

DUBLIN CITY UNIVERSITY

Breaking Language Barriers: Reimagining Machine Translation as Style Transfer

by

Sören Dréano, MSc

A Dissertation submitted in fulfilment of the requirements for the
award of Doctor of Philosophy (PhD)

in the

School of Electronic Engineering

Supervised by Prof. Derek Molloy
and Dr Noel Murphy

May 2025

Declaration of Authorship

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Doctor of Philosophy is entirely my own work, and I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: **Sören Dréano**

ID No: 19212555

Date: 20-05-2025

“- *Hello.*
- *With oil.*”

Mistranslated French joke

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Prof. Derek Molloy and Dr Noel Murphy. Four years ago, after a memorable interview in a small office on the top floor of the Stokes Building at the Glasnevin Campus of DCU, you entrusted me with the opportunity to start a PhD. You guided me through various challenges and changes in direction—from anomaly detection to object detection, and ultimately to unsupervised machine translation. Your unwavering support, whether in our weekly meetings—remote or in person—or through the many ups and downs, has been invaluable. Despite the challenging COVID-19 period, your mentorship kept me grounded. Thank you, Noel, for your insights into the subtleties of English grammar and Derek for helping me structure my ideas with clarity. Above all, I am deeply grateful for the excellent guidance and patience you both demonstrated throughout this journey.

To my ~~girlfriend~~ fiancée wife, Camille, thank you for being my anchor, my unwavering source of strength and support, whether I was in Dublin or back in France. Your serenity, your constant belief in me and your constant encouragement kept me going. Thank you for saying “yes” on the other side of the world and for reaffirming that commitment a year later. Most of all, thank you for Aurore.

To my family, your support over the years has been immeasurable. My parents, Yann and Florence, encouraged me to pursue my studies with dedication, providing me with everything I needed to succeed without ever letting me worry about anything else. Thank you, Florence, for your boundless affection and Yann, for instilling in me a sense of discipline and determination. To my brother Jäckel, who inspires me more than he realises and provided moments of respite during the intensity of this journey: thank you. To my extended family—my uncle, aunt, grandmother, cousins and my little cousin—I cherish our moments together and hope to see you all more frequently in the future.

To all those I met in Ireland, thank you for making this chapter unforgettable. Enric, thank you for our daily walks in the park and for your patient attempts to teach me guitar and singing. Joe, thank you for our exercise sessions. A heartfelt thanks to Amin, Carles, Fran, and Chris. Thank you Liam Meany and Sidra for being available when I needed you from France. I would also like to acknowledge the late Dr Kevin McGuinness for delivering the best Machine Learning course I have ever attended and for our enriching conversations afterward.

Lastly, to my friends—though the list could never be complete—thank you for being a part of my life and this journey. Ahmed, Boris, Jules, Baptiste, Maxence, Diego, and many others, you all hold a special place in this achievement.

Publications

We present a list of the works published during the course of this thesis. These references show the dissemination of the research findings and their contribution to the academic community in the field of [Natural Language Processing](#).

- Sören Dreano, Derek Molloy, and Noel Murphy. 2023. Tokengram_F, a Fast and Accurate Token-based chrF++ Derivative. In Proceedings of the Eighth Conference on Machine Translation, pages 730–737, Singapore. Association for Computational Linguistics.
- Sören Dreano, Derek Molloy, and Noel Murphy. 2023. Embed_Llama: Using LLM Embeddings for the Metrics Shared Task. In Proceedings of the Eighth Conference on Machine Translation, pages 738–745, Singapore. Association for Computational Linguistics.
- Sören Dreano, Derek Molloy, and Noel Murphy. 2024. CycleGN: A Cycle Consistent Approach for Neural Machine Translation. In Proceedings of the Ninth Conference on Machine Translation, pages 165–175, Miami, Florida, USA. Association for Computational Linguistics.
- Sören Dreano, Derek Molloy, and Noel Murphy. 2024. Exploration of the CycleGN Framework for Low-Resource Languages. In Proceedings of the Ninth Conference on Machine Translation, pages 756–761, Miami, Florida, USA. Association for Computational Linguistics.

Contents

Declaration of Authorship	iii
Acknowledgements	vii
List of Figures	xv
List of Tables	xxi
Abstract	xxxix
1 Introduction	1
1.1 Motivation	1
1.2 Research questions	2
1.3 Structure of the thesis	3
2 Literature Review	5
2.1 Fundamentals	5
2.1.1 Vectors, Matrices and Tensors	5
2.1.2 Machine Learning and Deep learning	6
2.1.3 Loss	6
2.1.4 Gradient Descent	8
2.1.5 Activation Function	12
2.2 Natural Language Processing	15
2.2.1 Tokenization	15
2.2.2 Categorical Variables	17
2.2.3 Training Word Vectors	19
2.2.4 Legacy Model Architectures	21
2.3 Transformer	25
2.3.1 Attention	26
2.3.2 Minor Improvements	28
2.4 Transformer Tasks	32
2.4.1 Sequence-to-Sequence	32
2.4.2 Language Modelling	33
2.4.3 Single-Sequence	34
2.4.4 Large Language Model	36
2.4.5 Vision Transformer and Multimodal Models	36
2.5 Machine Translation	37
2.5.1 Dataset Format	38

2.5.2	Evaluation	38
2.5.3	Recent Progress in Machine Translation	42
2.6	Generative Model Architectures	44
2.6.1	AutoEncoder	44
2.6.2	Variational AutoEncoder	45
2.6.3	Generative Adversarial Network	47
2.6.4	CycleGAN	48
2.7	Style Transfer	52
2.7.1	Image	52
2.7.2	Audio	55
2.7.3	Text	56
2.8	Quantisation	62
2.8.1	Precision	62
2.8.2	Computing Speed	65
2.8.3	Mixed Precision	66
2.8.4	Towards Integers	67
2.8.5	Scaling	68
2.8.6	Quantization-Aware Training	69
3	Machine Translation Evaluation	71
3.1	Introduction	71
3.2	Evaluation from Tokenization	72
3.2.1	A Deeper Dive into chrF++	72
3.2.2	Tokengram_F	75
3.2.3	Training	77
3.2.4	Results	79
3.2.5	Limitations	81
3.2.6	Future Work	81
3.3	Evaluation from Pre-trained Large Language Models	82
3.3.1	Embeddings Distance	82
3.3.2	Models	82
3.3.3	Distance Function	83
3.3.4	Embed_llama	85
3.3.5	Results	87
3.3.6	Limitations	91
3.3.7	Llama3	91
3.3.8	Future Work	93
3.4	Conclusion	94
4	Unsupervised Machine Translation	97
4.1	Introduction	97
4.1.1	A Cyclic Approach for Translation	98
4.1.2	Permuted and Non-Intersecting Dataset	99
4.1.3	Dataset	100
4.1.4	Tokenizer Training	101
4.1.5	Maximum Length of Sentences	101
4.1.6	Differences Between the English and German Corpora	103
4.1.7	Generator	104
4.1.8	Discriminator	107
4.1.9	Data types	109
4.1.10	Reproducible Build	110

4.2	Early Work	112
4.2.1	Baseline Results	112
4.2.2	Ideal implementation	113
4.2.3	Naive Implementation	115
4.2.4	Encoded Space	116
4.2.5	Pre-Trained Compressed Space	119
4.2.6	GPT2 with Language Modeling Head	121
4.2.7	BERT Generation	121
4.2.8	BERT and GPT2	122
4.3	CycleGN	123
4.3.1	Vocabulary Organisation	123
4.3.2	Dropping the Discriminator	124
4.3.3	Absence of Intermediate Evaluation	125
4.3.4	Escaping the Global Minimum	125
4.3.5	Pre-Training	126
4.3.6	Training	127
4.4	Challenges	128
4.4.1	Punishing the Model	130
4.4.2	Gradient Clipping	130
4.4.3	Batch Size	132
4.4.4	Per-Epoch Permutation	133
4.4.5	Size of the Training Dataset	134
4.4.6	Diminishing the Cost	135
4.4.7	Semantic Confusions	137
4.5	Results	142
4.5.1	Results on the English-German Language Pair	142
4.5.2	Results on the WMT24 Main Challenge	144
4.5.3	Low-Resource Languages	151
4.6	Comparison with Existing Frameworks	156
4.6.1	CycleGN Training	157
4.6.2	Reproducibility	159
4.6.3	Results	161
4.7	Limitations	162
4.8	Future Work	163
4.8.1	Gradient Accumulation	163
4.8.2	Model Architecture	163
4.8.3	Quantisation-Aware Training	164
4.8.4	Larger Datasets	164
4.9	Conclusion	165
5	Unsupervised Style Transfer	167
5.1	Introduction	167
5.2	Formality	168
5.3	Sentiment	172
5.4	Reflections and Future Directions in text style transfer	175
5.5	Limitations	175
5.6	Conclusion	175
6	Neural Text Compression	177
6.1	Introduction to Compression	177
6.1.1	Lossy Compression	178

6.1.2	Lossless Compression	179
6.2	Lossless text compression	180
6.2.1	Algorithms	180
6.2.2	General Benchmarks	181
6.3	Llamazip	183
6.3.1	Framework	184
6.3.2	Parameters	184
6.3.3	Algorithm	185
6.3.4	Similar Work	187
6.3.5	Results	188
6.3.6	Effects of the Final Algorithm	194
6.3.7	Speed	197
6.3.8	Memory Requirements	197
6.3.9	Optimal Parameters	198
6.3.10	Bytes Per Character	198
6.3.11	Other Usages of Llamazip	199
6.3.12	Limitations	203
6.3.13	Future Work	204
6.3.14	Conclusion	204
7	Conclusion	207
7.1	Answers to the Research Questions	207
7.2	Research Contributions	209
7.3	Future Research	211
7.4	Closing Remarks	212
A	Metric Results	215
A.1	Tokengram_F Results	215
A.2	Embed_llama Results	215
B	Publications	221
B.1	Tokengram_F, a Fast and Accurate Token-based chrF++ Derivative	222
B.2	Embed_Llama: Using LLM Embeddings for the Metrics Shared Task	230
B.3	CycleGN: A Cycle Consistent Approach for Neural Machine Translation	238
B.4	Exploration of the CycleGN Framework for Low-Resource Languages	249
	Bibliography	255

List of Figures

2.1	Rosenbrock function, obtained from [385]	8
2.2	Representation of the path of the gradient descent algorithm used to find the global minimum on the Rosenbrock function	10
2.3	3D representation of the path of the gradient descent algorithm used to find the global minimum on the Rosenbrock function	11
2.4	ReLU function, obtained from PyTorch documentation [280]	13
2.5	GELU function, obtained from PyTorch documentation [279]	14
2.6	SiLU function, obtained from PyTorch documentation [281]	15
2.7	Examples of Zipfian distributions, obtained from [241]	17
2.8	Representation of one-hot encoding, obtained from [114]	18
2.9	Architecture of a RNN feedback connection, obtained from Mishra et al. [237]	23
2.10	Architecture of an LSTM cell, obtained from [239]	24
2.11	Architecture of a GRU cell, obtained from [3]	25
2.12	Architecture of a Transformer model, in its original encoder-decoder version, obtained from [359]	26
2.13	Representation of the attention mechanism pipeline, which relies on the triplet of tensors (Q, K, V) to attend the elements of the sequence, obtained from [47]	28
2.14	Representation of the dropout mechanism, which disables random parameters during training, obtained from [334]	29
2.15	The effect of normalisation on different distributions, concentrating values around a mean, obtained from [46]	30
2.16	Layer Normalisation normalises across features within a single data point, ensuring consistent scaling for each input sample, while Batch Normalisation normalises across the batch dimension for each feature, stabilising feature distributions across the batch. In this Figure, obtained from [18], C represents the number of channels or the input and N represents the batch size.	31
2.17	Evolution of the number of parameters in LLMs, obtained from [228]	37
2.18	Architecture of the AutoEncoder model, in which an encoder compresses the input into a latent space and the decoder expands this vector into a vector which has the same dimension as the original input, obtained from Abdollahi et al. [2]	44
2.19	Deep Fake training process, in which two autoencoders sharing the same encoder are trained to reconstruct images of two different people, obtained from [227]	46
2.20	Difference between the inner workings of a VAE, in which the latent vector is sampled from a mean and a variance, and an AutoEncoder in which the latent vector is directly generated by the encoder, obtained from [105]	46

2.21	Representation of the training process of a GAN model, in which the generator generates data from random noise while the discriminator learns to distinguish original data from synthetic data, obtained from [325]	48
2.22	Images generated using a trained CycleGAN, obtained from [432] . .	49
2.23	CycleGAN training process, in which two GANs cooperate to generate credible synthetic data from synthetic and original data points, inspired from [392]. x and y are original images from the dataset, $fake\_xis$ obtained from $F_{Y \mapsto X}(y)$, $fake\_yis$ obtained from $G_{X \mapsto Y}(x)$, rec_x and rec_y are respectively obtained from $F_{Y \mapsto X}(fake\_y)$ and $G_{X \mapsto Y}(fake\_x)$ and id_x and id_y are respectively obtained from $F_{Y \mapsto X}(x)$ and $G_{X \mapsto Y}(y)$	51
2.24	Demonstration of the importance of the Identity Loss, obtained from [432]	51
2.25	Taxonomy of Style Transfer techniques, obtained from [161]. The taxonomy divides Neural Style Transfer methods into Image-Optimisation-Based Online Neural Methods and Model-Optimisation-Based Offline Neural Methods, further classifying them into parametric and non-parametric techniques, per-style and multiple-style models, and photorealistic and non-photorealistic applications	53
2.26	Style Transfer categories in image processing, obtained from [42] . .	54
2.27	Artistic styles applied to the same photography. The facades are the content of the picture, and the style is changed to transform a photographic image into a painting in the style of Van Gogh for example, obtained from [107]	54
2.28	Audio Style Transfer architecture, obtained from [116]	56
2.29	Taxonomy of text style transfer methods, obtained from [142]. This taxonomy categorises text style transfer techniques into parallel supervised, non-parallel supervised, and purely unsupervised methods. It further subdivides non-parallel supervised techniques into explicit and implicit style-content disentanglement methods, as well as those without disentanglement, highlighting strategies such as adversarial learning, attribute control generation, reinforcement learning, and entangled latent representation editing	58
2.30	Examples of sentences in the GYAFC dataset, which is a paired formal-informal text dataset, obtained from [64]	59
2.31	Format and bit attribution of floating-point IEEE 754, obtained from [121]	64
2.32	Performance improvements achieved by using TF32 and float16 precision on an NVidia A100, in comparison to float32 precision on an NVidia V100, obtained from [337]	65
2.33	Activation gradient magnitudes and the difference in representable range between float32 and float16, obtained from [247]	67
2.34	Speedup of Automatic Mixed Precision versus float32, obtained from [147]	68
2.35	Accuracy of the LLaMA3-8B model depending on the quantization type, obtained from [254]	69

3.1	Classification of text similarity functions, obtained from [368]. Text distance is subdivided into length-based, distribution-based, and semantic-based approaches, including Euclidean distance, KL divergence, and Word Mover’s Distance. Text representation methods range from string-based techniques (e.g., edit distance and Jaccard similarity) to corpus-based models (e.g., TF-IDF, Word2Vec, BERT), as well as semantic text matching and graph-structured representations like knowledge graphs and graph neural networks	84
3.2	Architecture of the LLaMA2 model, which is an open-source decoder-only LLM trained autoregressively, as displayed in the Huggingface library	86
3.3	System correlations with human rankings between the vector space of the reference and the hypothesis, depending on the last LLaMA layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks	88
3.4	Segment correlations with human rankings between the vector space of the reference and the hypothesis, depending on the last LLaMA layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks	88
3.5	System correlations with human rankings for each language pair in the WMT2022 between the vector space of the reference and the hypothesis, depending on the last LLaMA layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks	89
3.6	System correlations with human rankings of the English-to-Chinese language pair between the vector space of the reference and the hypothesis, depending on the last LLaMA layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks	90
3.7	System correlations depending on the LLaMA layer, using LLaMA3	91
3.8	Segment correlations depending on the LLaMA layer, using LLaMA3	92
3.9	System correlations for each language pair in the WMT2022 dataset depending on the LLaMA layer, using LLaMA3	93
3.10	System correlations of the English-to-Chinese language pair depending on the LLaMA layer and the dataset, using LLaMA3	93
4.1	Distribution of the number of tokens in the English dataset	104
4.2	Distribution of the number of tokens in the German dataset	104
4.3	Architecture of a Marian Encoder block as displayed in the Hugging-Face library	105
4.4	Architecture of a Marian Decoder block as displayed in the Hugging-Face library	106
4.5	Architecture of the CNN Discriminator	109
4.6	Generator training process, where $\neg y$ input ids refers to $\hat{y}$ and $\neg \neg x$ input ids refers to $\hat{x}$	114
4.7	Discriminator training process	115
4.8	Generator training process	116
4.9	Single encoder architecture	117
4.10	Dual encoder architecture	117

4.11	Generator training process. The encoder of $\mathcal{G}$ encodes x into zx . zx is then used as the input of the generator of $\mathcal{G}$ to obtain $\hat{z}y$. The generator of $\mathcal{F}$ transforms $\hat{z}y$ into $\hat{z}\hat{x}$ which is decoded by $\mathcal{F}$ into $\hat{x}$. The similarities between x and $\hat{x}$ on one side, and zx and $\hat{z}\hat{x}$ on the other, are used as loss functions. Another loss function is obtained through the discriminator $\mathcal{E}$. This process is repeated with y as inputs but was not represented for the sake of clarity	118
4.12	Discriminator training process. $\mathcal{D}$ takes as input zx , that is to say the output of the encoder of $\mathcal{G}$ given x and learns to classify it as a “real” sample. On the other hand, it learns to classify as a “fake” sample $\hat{z}x$, which is obtained from the generator of $\mathcal{F}$ given zy , itself obtained through the encoder of $\mathcal{F}$ given y . This training process is repeated for the discriminator $\mathcal{E}$ but is not shown for the sake of clarity.	119
4.13	Pre-training process	120
4.14	In the absence of a Discriminator $y \in \mathcal{Y}$, the CycleGN architecture will converge towards a state where no translation happens and still perfectly satisfy the Cycle-Consistency Loss function	125
4.15	Masked Language Modeling training process	126
4.16	CycleGN training process	127
4.17	Divergence in the Cycle-Consistency Loss of CycleGN training . . .	129
4.18	Divergence in the Cycle-Consistency Loss of CycleGN training . . .	131
4.19	Cycle-Consistency Loss of CycleGnN training with gradient clipping	132
4.20	Evolution of the COMET score during the permuted training of the small CycleGN framework on the WMT14 English-French language pair	137
4.21	Evolution of the COMET score during the permuted training of the small CycleGN framework on the WMT24 English-German language pair	138
4.22	Evolution of the COMET score during the permuted training of the small CycleGN framework on the WMT24 English-Spanish language pair	138
4.23	Evolution of the COMET score during permuted CycleGN training	143
4.24	Evolution of the COMET score during non-intersecting CycleGN training	143
4.25	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT24 English-Russian language pair	148
4.26	Evolution of the COMET score during the non-intersecting training of the CycleGN framework on the WMT24 English-Russian language pair	149
4.27	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT24 English-Chinese language pair	149
4.28	Evolution of the COMET score during the non-intersecting training of the CycleGN framework on the WMT24 English-Chinese language pair	151
4.29	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT24 English-Spanish language pair	151
4.30	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT24 English-Icelandic language pair	152
4.31	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 English-Assamese language pair	154

4.32	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 English-Khasi language pair	154
4.33	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 English-Manipuri language pair	154
4.34	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 English-Mizo language pair	154
4.35	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 Spanish-Aragonese language pair	155
4.36	Evolution of the Cycle-Consistency Loss during the permuted training of the CycleGN framework on the WMT24 Spanish-Asturian language pair	155
4.37	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT14 English-French language pair .	158
4.38	Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT16 English-German language pair	158
4.39	Linear scheduler with warm-up phase	164
6.1	Compression of alice29.txt depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping	188
6.2	Compression of the Frankenstein novel depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping	189
6.3	Compression of the FSDFSD novel depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping	191
6.4	Compression of the novel generated by LLaMA3 depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping	192
6.5	Compression of the novel generated by Mistral depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping	193
6.6	Percentage of correct predictions on the alice29 document depending on the quantisation and the context window	202
6.7	Percentage of correct predictions on the Frankenstein document depending on the quantisation and the context window	202
6.8	Percentage of correct predictions on the FSDFSD document depending on the quantisation and the context window	203

List of Tables

2.1	Example of a CBoW training dataset	21
3.1	Correlations between system and sentence translations with human ranking over all language pairs depending of the hyperparameters of Tokengram_F with a vocabulary size of 50,000	79
3.2	Tokengram_F results on the WMT20 dataset compared to chrF++	80
3.3	Tokengram_F results on the WMT21 dataset compared to chrF (chrF was used in place of chrF++ as chrF++ where the results were not reported)	80
3.4	Tokengram_F results on the WMT22 dataset compared to chrF++ (chrF was used in place of chrF++ as chrF++ where the results were not reported)	80
3.5	Wilcoxon signed-rank test results comparing Tokengram_F and chrF on WMT20, WMT21 and WMT22	81
3.6	Means and standard deviations of the system correlations with human rankings depending on the last LLaMA layer used, when WMT20, WMT21 and WMT22 are merged. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks	90
3.7	Means and standard deviations of the system correlations depending on the LLaMA layer when WMT20, WMT21 and WMT22 are merged	94
4.1	Representation of a parallel dataset	99
4.2	Representation of a permuted dataset	99
4.3	Representation of a non-intersecting dataset	100
4.4	Impact of the maximum sentence length on the percentage of untruncated sentences	103
4.5	Impact of the maximum sentence length on the maximum batch size	103
4.6	Impact of the size of the vocabulary on the number of unknown tokens	106
4.7	Examples of German sentences where a larger vocabulary size prevents out-of-vocabulary errors	106
4.8	Non-default parameters in the configuration of Marian Transformer models	107
4.9	COMET score of the baseline models	113
4.10	COMET score of the Helsinki models	113
4.11	Vocabulary organisation with a single tokenizer	123
4.12	Vocabulary organisation with two tokenizers	124
4.13	Vocabulary organisation with our tokenizer	124
4.14	Evolution of the translation of a test sentence	128
4.15	Evolution of the translation of a test sentence and its backtranslation obtained from Deepl.com	129
4.16	COMET score of the CycleGN framework depending on data augmentation	133

4.17	COMET score of the CycleGN framework depending the number of epochs	135
4.18	COMET score obtained by models trained using the CycleGN framework depending on the dataset and quantisation	136
4.19	Differences in the architecture of the default CycleGN model and the small CycleGN model	136
4.20	COMET score obtained by models trained using the CycleGN framework depending on the number of parameters	137
4.21	Examples of location confusion produced by CycleGN from German to English	139
4.22	Examples of name confusion produced by CycleGN from German to English	140
4.23	Examples of age confusion produced by CycleGN from German to English	141
4.24	Examples of date confusion produced by CycleGN from German to English	141
4.25	COMET scores of the CycleGN and baseline models	142
4.26	Generated development translations at specific batch indexes	145
4.27	Comparison between the number of sentences available in the WMT24 dataset and the number of sentences used to train the permuted models depending on the language pair	146
4.28	Comparison between the number of sentences available in the WMT24 dataset and the number of sentences used to train the non-intersecting models depending on the language pair	146
4.29	COMET scores obtained by CycleGN models depending on the language pair and the dataset type on the WMT24 dataset. Note that these are not official WMT24 results, as the official scores have not yet been released. The scores shown are based on a non-official test set extracted from the training data	147
4.30	Model performance across language pairs (caption not rotated)	150
4.31	Number of sentences for each language pair and number of epochs during training	153
4.32	Comparison between the BLEU scores obtained in the original XLM publication and the reproduction	160
4.33	Comparison between the BLEU scores obtained in the original MASS publication and the reproduction	160
4.34	BLEU scores obtained in the original CBD publication	161
4.35	Comparison between the number of parameters of CycleGN, XLM, MASS and CBD	161
4.36	Comparison between the COMET scores obtained by the CycleGN models and XLM, MASS and CBD	162
5.1	Scores obtained on the GYAFC dataset in the Formal to Informal direction depending on the model	170
5.2	Scores obtained on the XFORMAL French dataset in the Formal to Informal direction depending on the model	171
5.3	Scores obtained on the XFORMAL Italian dataset in the Formal to Informal direction depending on the model	171
5.4	Scores obtained on the Yelp dataset depending on the model. The results are averaged on both directions	173
6.1	Size of the compressed enwik9 dataset depending on the compression algorithm	182

6.2	Size of the compressed Silesia Corpus depending on the compression algorithm	183
6.3	Size of the compressed alice29.txt file depending on the compression algorithm	183
6.4	Size of the compressed alice29.txt file depending on the compression algorithm	189
6.5	Size of the compressed Frankenstein.txt file depending on the compression algorithm	190
6.6	Size of the compressed FSDFSD.txt file depending on the compression algorithm	190
6.7	Size of the compressed LLaMA3 novel file depending on the compression algorithm	192
6.8	Size of the compressed Mistral novel file depending on the compression algorithm	193
6.9	Size of the different compressed batches of the enwik9 dataset using Llamazip. For all algorithms other than Llamazip presented in this table, the compression ratios were sourced from [224], and the compressed sizes were extrapolated based on these ratios	194
6.10	Variations in the size of the compressed alice29 document depending on the quantisation and context window of Llamazip	194
6.11	Average size of the alice29 file over all context windows and quantisations depending on the final compression algorithm	195
6.12	Variations in the size of the compressed Frankenstein document depending on the quantisation and context window of Llamazip	195
6.13	Average size of the Frankenstein file over all context windows and quantisations depending on the final compression algorithm	195
6.14	Variations in the size of the compressed FSDFSD document depending on the quantisation and context window of Llamazip	196
6.15	Average size of the FSDFSD file over all context windows and quantisations depending on the final compression algorithm	196
6.16	Variations in the size of the compressed document generated by LLaMA3 depending on the quantisation and context window of Llamazip	196
6.17	Average size of the file generated by LLaMA over all context windows and quantisations depending on the final compression algorithm	196
6.18	Variations in the size of the compressed document generated by Mistral depending on the quantisation and context window of Llamazip	197
6.19	Average size of the file generated by Mistral over all context windows and quantisations depending on the final compression algorithm	197
6.20	Number of tokens compressed per second using Llamazip depending on the context window and the quantisation using a NVidia 4090	197
6.21	Size (in MiB) of the memory required to compress a file using Llamazip depending on the context window and the quantisation	198
6.22	Number of bytes necessary to represent one character depending on the document and the compression algorithm	199
6.23	Percentage of correct predictions depending on the document and the context window using float16 quantisation	200
6.24	Size of the books from MIT Press depending on the parameters from Llamazip	201
A.1	Pearson correlations (r) and Kandall correlations (τ) obtained by Tokengram_F in the WMT20 challenge	215
A.2	Pearson correlations (r) and Kandall correlations (τ) obtained by Tokengram_F in the WMT21 challenge	215

A.3	Pearson correlations (r) and Kandall correlations (τ) obtained by Tokengram_F in the WMT22 challenge	216
A.4	Pearson correlations (r) for the WMT20 system dataset depending on the Llama layer. The findings pertaining to the second layer are shown in bold, as it represents the prevailing default layer count within the Embed_Llama	216
A.5	Kendall correlations (τ) for the WMT20 segment dataset depending on the Llama layer	217
A.6	Pearson correlations (r) for the WMT21 system dataset depending on the Llama layer	217
A.7	Kendall correlations (τ) for the WMT21 segment dataset depending on the Llama layer	218
A.8	Pearson correlations (r) for the WMT22 system dataset depending on the Llama layer	218
A.9	Kendall correlations (τ) for the WMT22 segment dataset depending on the Llama layer	219

Acronyms

AE AutoEncoder. [xv](#), [44–46](#), [51](#), [70](#), [120](#), [173](#)

AI Artificial Intelligence. [5](#), [6](#), [34](#), [36](#), [204](#)

BCEL Binary Cross-Entropy Loss. [7](#), [8](#), [45](#)

BERT Bidirectional Encoder Representations from Transformers. [xvii](#), [34–36](#),
[40](#), [70](#), [84](#), [87](#), [108](#), [121–123](#), [126](#), [134](#), [164](#)

BLEU Bilingual Evaluation Understudy. [xxii](#), [3](#), [39](#), [42](#), [60](#), [62](#), [70](#), [75](#), [81](#),
[112](#), [134](#), [138](#), [139](#), [144](#), [156](#), [159–162](#), [168–170](#), [175](#)

BoW Bag-of-Words. [19](#), [20](#), [82](#)

BPE Byte Pair Encoding. [16](#), [17](#), [76](#), [77](#), [82](#)

CBD Cross-model Back-translated Distillation. [xxii](#), [156](#), [157](#), [159–162](#)

CBoW Continuous Bag-of-Words. [xxi](#), [20](#), [21](#)

CCL Cycle-Consistency Loss. [xviii](#), [xix](#), [4](#), [50](#), [70](#), [98](#), [114](#), [118](#), [125](#), [127–](#)
[129](#), [131](#), [132](#), [153–155](#), [161](#), [208](#), [210](#)

CEL Cross-Entropy Loss. [8](#)

CLM Causal Language Modeling. [33](#), [34](#), [82](#), [157](#)

CNN Convolutional Neural Network. [xvii](#), [52](#), [55](#), [107–109](#), [168](#), [171](#)

COMET Crosslingual Optimised Metric for Evaluation of Translation. [xviii](#), [xix](#),
[xxi](#), [xxii](#), [40](#), [41](#), [70](#), [81](#), [95](#), [112](#), [113](#), [133](#), [135–139](#), [141–144](#), [147–](#)
[149](#), [151](#), [152](#), [158](#), [159](#), [161–163](#), [207](#), [212](#)

DL Deep Learning. [6](#), [16](#), [197](#)

DNN Deep Neural Network. [12](#), [107](#)

- DocMT** Document-Level Machine Translation. 43
- ELMo** Embeddings from Language Models). 36
- GAN** Generative Adversarial Network. xvi, 47–49, 51, 55, 61, 62, 70, 99, 107, 108, 119, 124
- GELU** Gaussian Error Linear Unit. xv, 13, 14
- GEMBA** GPT Estimation Metric Based Assessment. 41
- GPT** Generative Pre-trained Transformer. 33–36, 44, 70, 121–123, 134, 157, 164, 181
- GRU** Gated Recurrent Units. xv, 24, 25, 172, 173
- GYAFC** Grammarly’s Yahoo Answers Formality Corpus. xvi, xxii, 59, 168, 170–172, 210
- IL** Identity Loss. xvi, 50, 51, 114, 118, 127, 161
- LLaMA** Large Language Model Meta AI. xvi, xvii, xix, xxi, xxiii, 34, 35, 69, 70, 83, 86–95, 183, 184, 186, 187, 189–193, 195, 196, 199–201, 203, 204, 207–209, 211
- LLM** Large Language Model. xv, xvii, xxxi, 2–4, 33, 34, 36, 37, 41, 43, 61, 71, 72, 82, 83, 85, 86, 91, 94, 95, 97, 156, 157, 177, 183, 184, 187, 191, 199–202, 204, 205, 207–212
- LM** Language Modeling. 33, 35, 70
- LSTM** Long Short-Term Memory. xv, 24, 25, 27, 36, 70, 101, 169, 173, 181
- LZ77** Lempel-Ziv 1977. 180, 181
- LZMA** Lempel-Ziv-Markov chain Algorithm. 180, 181
- MAE** Mean Absolute Error. 7, 11
- MASS** MAsked Sequence to Sequence pre-training. xxii, 156, 157, 159–162
- ML** Machine Learning. 3, 5, 6, 12, 18, 19, 25, 44, 62, 64, 67, 69, 70, 75, 103, 110, 112, 153, 181, 211–213
- MLM** Masked Language Modeling. 33, 35, 126, 136, 156, 157, 208, 210

- MSE** Mean Square Error. [7](#), [45](#)
- MT** Machine Translation. [xxxi](#), [1–4](#), [15](#), [16](#), [21–26](#), [32](#), [37–44](#), [70–72](#), [75](#), [78](#), [95](#), [97–100](#), [121](#), [133](#), [134](#), [143](#), [146](#), [151](#), [155–157](#), [159](#), [161](#), [162](#), [165](#), [175](#), [184](#), [208–210](#), [212](#), [213](#)
- NER** Named Entity Recognition. [34](#), [36](#), [126](#), [141](#)
- NLP** Natural Language Processing. [ix](#), [1](#), [2](#), [4](#), [6](#), [12](#), [13](#), [15–17](#), [19](#), [21](#), [23](#), [25](#), [29](#), [32–37](#), [42](#), [57](#), [69](#), [70](#), [78](#), [101](#), [103](#), [126](#), [134](#), [164](#), [181](#), [209–212](#)
- NMT** Neural Machine Translation. [3](#), [22](#), [27](#), [32](#), [37](#), [43](#), [71](#), [97](#), [98](#), [112–116](#), [123](#), [128](#), [129](#), [134](#), [156](#), [167](#), [169](#), [177](#)
- NSP** Next Sentence Prediction. [35](#), [126](#)
- NTP** Next Token Prediction. [4](#), [33](#), [35](#), [36](#), [164](#), [181](#), [183–185](#), [187](#), [199](#), [200](#), [204](#), [205](#), [210](#)
- PINC** Paraphrase In N-gram Changes. [168](#), [169](#)
- PRNG** Pseudorandom Number Generator. [111](#)
- PTQ** Post-Training Quantization. [69](#)
- QAT** Quantization-Aware Training. [69](#), [164](#)
- RBMT** Rule-based machine translation. [21](#), [22](#)
- ReLU** Rectified Linear Unit. [xv](#), [12–14](#)
- RLE** Run-Length Encoding. [180](#), [181](#)
- RNG** Random Number Generator. [110](#), [191](#)
- RNN** Recurrent Neural Network. [xv](#), [22–24](#), [27](#), [32](#), [55](#), [70](#), [101](#)
- RoBERTa** Robustly Optimized BERT Approach. [35](#), [36](#), [126](#)
- sBLEU** source Bilingual Evaluation Understudy. [60](#), [61](#)
- seq2seq** Sequence-to-Sequence. [32](#), [33](#), [37](#), [115](#), [116](#), [121](#), [126](#), [167](#)
- SiLU** Sigmoid Linear Unit. [xv](#), [14](#), [15](#)
- SMT** Statistical Machine Translation. [22](#), [70](#)

- TF-IDF** Term Frequency-Inverse Document Frequency. 20, 84
- VAE** Variational AutoEncoder. xv, 45–47, 62, 120, 121
- WMT** Workshop on statistical Machine Translation. xvii–xix, xxi–xxiv, 41, 43, 78–81, 86–94, 100, 112, 113, 134, 136–138, 142, 144–149, 151–155, 158, 159, 162, 163, 207, 215, 216
- WO** Word Overlap. 60, 61
- XLM** Cross-lingual Language Model Pretraining. xxii, 156, 157, 159–162, 170

À Aurore

DUBLIN CITY UNIVERSITY

Abstract

School of Electronic Engineering

Breaking Language Barriers: Reimagining Machine Translation as Style Transfer

by Sören Dréano, MSc

Machine Translation (MT) has traditionally relied on parallel corpora, posing significant challenges for low-resource languages. This thesis reimagines **Machine Translation** as a style transfer problem, introducing a novel framework, **CycleGN**, to enable both translation and style transfer without requiring parallel data sets. Using monolingual corpora, the proposed method seeks to broaden equitable access to language translation systems, while advancing theoretical insights into style transfer.

Another contribution of this work is the development of **Tokengram_F**, a novel metric that extends n -gram analysis to better capture linguistic and contextual nuances in translation evaluation and can estimate the quality of machine-generated sentences in more than 200 different languages. Furthermore, **Embed_llama** leverages pre-trained **Large Language Model (LLM)** embeddings to enhance semantic alignment and evaluation accuracy, deepening the work on transfer learning.

This thesis also explores text compression through the development of **Llamazip**, a lossless compression algorithm that uses the predictive capabilities of **LLMs**. Beyond achieving excellent compression ratios, Llamazip demonstrates innovative applications, such as identifying training set membership of given target text and benchmarking predictive performance.

The research presented in this thesis has led to the publication of four peer-reviewed publications, the submission of another one, and we are also in the process of writing a further paper. Ultimately, this work seeks to **democratise access to translation technologies** by broadening the scope of accessible training data. It aims to contribute to the evolution of language technologies in a multilingual world.

Chapter 1

Introduction

1.1 Motivation

Currently, the datasets required to train neural networks applied to translation need to be aligned in pairs, that is, two sentences—one in each language—with effectively the same meaning need to be available in two different languages, so that the model can start training to establish links between the languages. Many sources of this type exist, for example, supranational institutions such as the European Union or the United Nations make their working documents, agreements, and other texts available. Because they bring together countries with different languages and because these agreements can have the force of law, the translations are among the best in the world. These datasets are essential for training traditional [Machine Translation \(MT\)](#) models, but are costly and often unavailable for low-resource languages. This limitation excludes many languages from benefiting from advances in [Machine Translation](#), effectively creating an accessibility gap in multilingual communication.

The goal of this research is to find a training architecture such that parallel datasets are not necessary. Rather than viewing [Machine Translation](#) solely as a traditional [Natural Language Processing \(NLP\)](#) challenge, this research reimagines it as a problem of style transfer. This perspective takes advantage of advances in unsupervised learning and generative modeling techniques to propose a novel approach to text-to-text translation. CycleGAN is an architecture that revolutionised style transfer for images, as it trains a model pair to learn how to shift from one domain to another and vice versa, without using any paired examples. The postulate is that applying the same underlying principles to text would result in models that can accurately translate text, even when they have never been exposed to the exact translation of any input sentence. To be more specific, the aim is to train [Machine Translation](#) models from a dataset that contains many sentences in natural languages, without the need for the correspondences between sentence pairs in the different languages to be established. The implications of this work extend beyond conventional language pairs, as this approach can be adapted to address stylistic transformations within the same language, such as the transition between, say, technical and literary English.

To be able to dispense with parallel corpora and simply require unpaired sentences in each of the languages, i.e., two unmatched bodies of text, would significantly open up the possibilities offered to translation. Indeed, there are many languages that are not yet translated, such as Linear A of the Minoans, which could be if such a method existed. As long as we collect enough sentences from an untranslated language, then translating this language is just a matter of training a model using a non-parallel dataset, created from the untranslated language and another “known” language.

This work also focusses on evaluating the quality of machine-generated translations. The metrics for translation evaluation are pivotal for the advancement of translation methods, as metrics with stronger correlations with human judgments enable more accurate and meaningful comparisons between models. There is in fact a dual need: on the one hand for cost-effective metrics, and on the other, for leveraging transfer learning to repurpose large, computationally expensive metric models.

Finally, within the broader framework of [Natural Language Processing](#) and transfer learning, this research also explores the generalisability of [Large Language Model \(LLM\)](#)s to tasks beyond text generation, with particular attention to lossless text compression. Although lossless text compression and [Machine Translation](#) are distinct tasks, they share underlying principles related to information representation and efficient encoding. Both require models to capture and manipulate linguistic structures, with translation focusing on meaning preservation across languages and compression emphasising compact text representation. By examining the capacity of [Large Language Models](#) to predict units of text, this work provides insights into their capacity for semantic understanding.

Ultimately, this research seeks not only to address the practical limitations of existing [Machine Translation](#) paradigms, but also to expand the theoretical understanding of domain adaptation in [Natural Language Processing](#). By broadening the scope of accessible training data and improving the inclusivity of [Machine Translation](#) technologies, this work aims to contribute meaningfully to the evolution of language technologies in a multilingual world.

1.2 Research questions

We outline the fundamental inquiries that guide this thesis. These questions are designed to address the key challenges and gaps identified in the field of [Natural Language Processing](#), particularly in unsupervised [Machine Translation](#), text compression, and evaluation metrics. By framing the scope and direction of the research, this subsection establishes the foundation for the subsequent exploration of novel methodologies and their implications.

Hypothesis 1: Traditional non-neural evaluation metrics for [Machine Translation](#) lose critical information when they only compare whole words. This loss is particularly evident when considering synonyms and words that share the same root.

Hypothesis 2: [Large Language Models](#) encode rich semantic representations due to their exposure to massive amounts of data. These representations can be leveraged to

assess the semantic similarity of sentences across languages, improving [Machine Translation](#) evaluation.

Research question 1: How can non-neural evaluation metrics and [Large Language Models](#) be leveraged to improve the assessment of machine translation quality while maintaining efficiency?

Hypothesis 3: If a [Neural Machine Translation](#) model can be trained to compare a sentence with its translation, it should also be possible to train a model without direct translation pairs, learning from a dataset where no sentence in one language directly corresponds in meaning to any sentence in the other language.

Hypothesis 4: [Machine Translation](#) and text style transfer share fundamental similarities. Techniques from image style transfer, such as CycleGAN, can be adapted to train MT models on non-parallel datasets.

Research question 2: How can techniques from image style transfer, such as CycleGAN, be adapted for unsupervised [Machine Translation](#) in the absence of parallel datasets?

Hypothesis 5: A technique originally developed for image style transfer and adapted for unsupervised [Machine Translation](#) can also be applied to text style transfer.

Research question 3: How can style transfer methods originally designed for images be adapted for text style transfer tasks?

Hypothesis 6: Any model that predicts the next token based on a sequence of previous tokens operates similarly to a Markov chain, a principle used in text compression. Consequently, such models may be repurposed for more efficient lossless text compression.

Hypothesis 7: The performance of a model varies with quantisation, with greater variations observed for data that were included in the training set.

Research Question 4: How can [Large Language Models](#) be adapted for both efficient text compression and dataset membership inference through quantisation analysis?

1.3 Structure of the thesis

In Chapter 2, we provide the technical background necessary for this thesis. We address the foundations of [Machine Learning](#) (ML), [Machine Translation](#), style transfer and generative architectures.

In Chapter 3, we present our work on [Machine Translation](#) evaluation. The Token-gram_F metric is introduced here as a refined approach for evaluating translations, combining the granularity of n -gram-based metrics with enhanced contextual sensitivity. This metric is designed to address the limitations of traditional evaluation methods such as [BLEU](#), providing a more nuanced assessment of translation quality. A second contribution in the “metric” space is Embed_llama, which integrates large-scale language embeddings into the translation framework. Using Llama embeddings, this approach enhances the semantic representation of text, allowing the model to better capture the nuances of meaning across domains and languages.

In Chapter 4, we focus on the development of our unsupervised [Machine Translation](#) technique, CycleGN, based on an image style transfer technique. In this chapter, we demonstrate that the use of a [Cycle-Consistency Loss \(CCL\)](#) is effective in training a model even on non-intersecting datasets, a specific case of non-parallel datasets in which there are no sentence pairs sharing the same meaning between the two languages.

In Chapter 5, we evaluate whether the technique developed in Chapter 4 is also effective for text style transfer, using non-parallel datasets.

In Chapter 6, we engage with lossless text compression based on pre-trained [Large Language Models](#). The author presents Llamazip, an algorithm that relies on the [Next Token Prediction \(NTP\)](#) capacities of [LLMs](#) to store only mispredictions. Furthermore, this algorithm can be used to detect whether a text document was part of the training set of the underlying [LLM](#).

Finally, in Chapter 7, we conclude this thesis by presenting an overview of the research conducted, answering the research questions, and proposing future directions in the [Natural Language Processing](#) domain.

Chapter 2

Literature Review

Any sufficiently advanced
technology is indistinguishable
from magic

Arthur C. Clarke

2.1 Fundamentals

Although [Artificial Intelligence](#) is a rapidly expanding field, as evidenced by the ferment of ideas in the contemporary Zeitgeist, it is also a poorly understood, almost magical field, with unknown inner workings and “infinite” capabilities.

This introduction presents some key concepts for understanding the field, as well as an example that will show how an algorithm can improve by multiplying iterations on a dataset.

2.1.1 Vectors, Matrices and Tensors

Linear algebra is the branch of mathematics that focusses on vector spaces and linear transformations. A vector space is a collection of vectors which can be added together and multiplied by a scalar. These vectors possess both magnitude and direction and are usually represented as one-dimensional arrays of numbers of n components. The meaning of a vector depends on a context. For example, it can represent the geometric translation of a point from a set of coordinates to another, a physical quantity such as gravity, the evolution of a value over time, or a word vector, as discussed in [Section 2.2.2](#).

Matrices are two-dimensional arrays that contain m rows and n columns. Depending on the context, a matrix can, for example, represent the concatenation of n vectors of m elements, the mathematical operator to transform a vector or another matrix, a set of weights in [Machine Learning \(ML\)](#) or the evolution of a vector over time.

Finally, tensors are the generalisation of vectors and matrices in higher dimensions, thus vectors and matrices are specific cases of tensors and matrices of order 1 and 2 respectively. As is the case with vectors and matrices, tensors serve as a versatile instrument, including a means of representing data. While a greyscale image can be represented by a matrix of $m \times n$ pixels where the value of each element denotes the luminosity of the corresponding pixel, a colour image requires another dimension to encode the added information. Thus, a full HD image in RGB can be represented as a tensor of dimension (3, 1080, 1920), where the quantity of blue for the pixel at position (100, 0) is the element at index (2, 100, 0). A colour video being an ordered set of colour images, another dimension is needed to represent the evolution through time. Thus, a dimension (100, 3, 1080, 1920) tensor can adequately represent 100 frames of full HD video in RGB, where the element at index (8, 1, 25, 25) represents the amount of green in the ninth frame at position (25, 25).

2.1.2 Machine Learning and Deep learning

[Machine Learning](#) is a field of [Artificial Intelligence](#) that uses systems that improve their performance over iterations by learning from data. It is a broad field that includes algorithms such as decision trees and support vector machines that rely on statistical methods and human fine-tuning to make predictions or decisions. [Deep Learning](#) is a subset of [Machine Learning](#) that relies on artificial neural networks to automatically learn hierarchical representations of data. In order to qualify as “deep”, these models require multiple layers of neurons. [DL](#) models can automatically learn relevant features from raw data, particularly in tasks such as image recognition or [Natural Language Processing](#).

2.1.3 Loss

In [Machine Learning](#), a loss function is a method to quantify the difference between the predictions of a model and the reference data. The objective of [Machine Learning](#) is to minimise this difference, that is, train a model so that its predictions are indistinguishable from the dataset.

The loss function is at the heart of [Machine Learning](#), as measuring the error in the predictions and providing feedback to a model is required to train the parameters. Many loss functions exist depending on the task and they can be broadly classified into regression losses and classification losses.

Regression

Regression is a statistical method that models the relationship between a dependent variable and one or more independent variables. Although regression has its use in inferring causal relationships, it is most commonly used in [Machine Learning](#) for predictions. For example, from a dataset that contains the growth of a plant depending on

the amount of water it receives per day, a regression analysis can be used to predict the growth for an untested water quantity either by interpolating if the quantity belongs to the interval present in the dataset, or extrapolating if it is not. Although regressions can be linear, as is the case with variables that are linearly correlated, the previous example is an example of a nonlinear regression, as both the absence of water and an excessive amount lead to the death of the plant.

The **Mean Square Error (MSE)**, also known as the squared L2 norm, is the average of the squared difference between the predicted values and the observed values and its formula is:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

where Y_i is the value of the i -th element in the dataset and $\hat{Y}_i$ is its predicted value.

A close alternative to **MSE** is **Mean Absolute Error (MAE)**. Unlike **MSE**, **MAE** calculates the average of absolute differences between predicted and actual values, without squaring the errors. Consequently, **MAE** is less sensitive to large errors and does not penalise them as much as **MSE** does.

Classification

Although regression applies to continuous variables, classification is the task of classifying discrete values, that is, predicting a label given a set of variables. Classification involves predicting a discrete label or category given a set of input variables, such as classifying pictures depending on the most visible item.

Binary Classification As its name suggests, binary classification is a subcategory of classification where there are only two classes, which corresponds, for example, to a classification of the presence or absence of cancer or to detect whether an email attachment contains a virus. Using this last example, the **Binary Cross-Entropy Loss (BCEL)** calculates a loss based on the probability that a file is indeed malicious. By definition, entropy is the measure of uncertainty. In information theory, entropy is related to the quantity of information, or bits, required to describe data. If all the files are malicious, there is no uncertainty associated with a given file and, as such, the associated entropy would be zero. On the other hand, if half of the files contain viruses, the entropy is maximal and is equal to $\log(2)$, as there are only two classes possible.

The formula of **BCEL** is :

$$BCEL = - \sum_{i=1}^n Y_i \times \log(\hat{Y}_i) + (1 - Y_i) \times \log(1 - \hat{Y}_i)$$

where $Y_i \times \log(\hat{Y}_i)$ computes the loss if the item belongs to the first class and $(1 - Y_i) \times \log(1 - \hat{Y}_i)$ computes the loss if the item belongs to the second class.

Multi-Class Classification [Cross-Entropy Loss \(CEL\)](#) is a generalisation of [BCEL](#) that supports multi-class classification. For each sample, the loss is computed as the negative log-probability of the true class. The overall [CEL](#) is the average of these losses across all samples in the dataset or batch.

As is the case with all loss functions, a higher value indicates that the model does not accurately classify the data. In fact, as in $\log(1.0) = 0$, a model attributed to a probability of input of 1.0 to the correct class will result in a loss of 0.0. On the other hand, if the model mistakenly assigns a probability of 0.0 to the correct class, the result of the loss function is $-\infty$ (although $\log(0)$ is not mathematically defined, PyTorch interprets the result as $-\infty$).

2.1.4 Gradient Descent

The objective of loss functions is to determine how accurate a model is, and gradient descent is the process of optimising the parameters of the model to minimise the value of the loss function. The aim of training is to reduce this loss function and the process at the heart of this training is gradient descent.

To facilitate understanding, the Rosenbrock function, shown in Figure 2.1, will be used as an example in its 2D version, as it has been used as a performance test problem for optimisation [302]. It is a nonconvex function with a global minimum inside a long, narrow, parabolic valley, and its formula is $f(x, y) = (1 - x)^2 + 100 \times (y - x^2)^2$. The objective of the gradient descent process is to find the global minimum of the function, which is in location (1, 1).

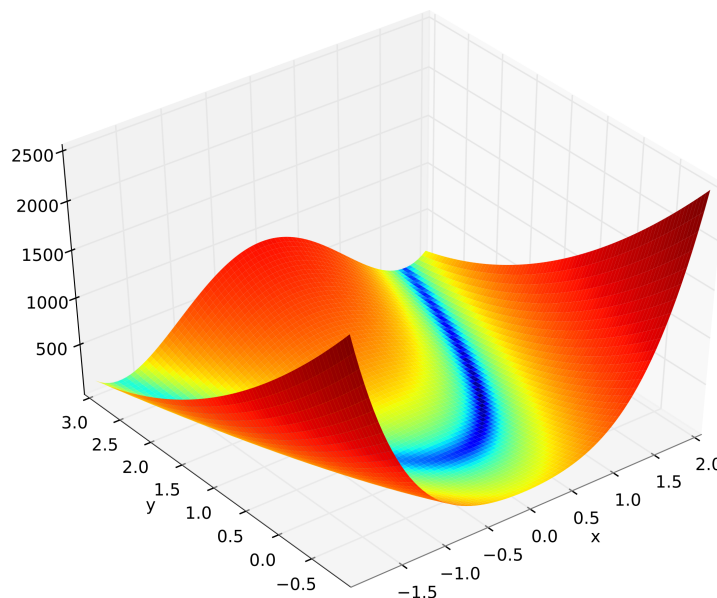


FIGURE 2.1: Rosenbrock function, obtained from [385]

From the Derivative

The formula of the derivative of the Rosenbrock function is as follows:

$$\begin{aligned}\frac{df}{dx} &= -2 \times (1 - x) - 400 \times x \times (y - x^2) \\ \frac{df}{dy} &= 200 \times (y - x^2)\end{aligned}$$

The gradient descent algorithm is an iterative process in which at each step the weights are updated by computing the gradients of the functions. The gradient of each parameter is multiplied by a hyper-parameter, the learning rate, in order to only slightly change the weights.

```

def rosenbrock(x, y):
    return (1 - x)**2 + 100 * (y - x**2)**2

def rosenbrock_gradient(x, y):
    return -2 * (1 - x) - 400 * x * (y - x**2), 200 * (y - x**2)

lr = 0.001
iterations = 10000
param_x = 2.0
param_y = 0.0

for _ in range(iterations):
    dx, dy = rosenbrock_gradient(param_x, param_y)
    param_x -= lr * dx
    param_y -= lr * dy

```

CODE LISTING 2.1: Python implementation of the gradient descent algorithm for the Rosenbrock function

In real-life conditions, the weights would be randomly sampled from a known distribution, such as a Gaussian distribution. However, in order to demonstrate the efficiency of the gradient descent process, values that converge slowly have been selected. Figures 2.2 and 2.3 display the evolution of the Rosenbrock function result during training and have been inspired by [276]. Before training, the model obtains $f(2,0) = 1601$. At the end of the training process, the weight values are 0.993 and 0.985 and the model value is $f(0.993, 0.985) = 5.381e - 5$, indicating that the algorithm has successfully minimised the function.

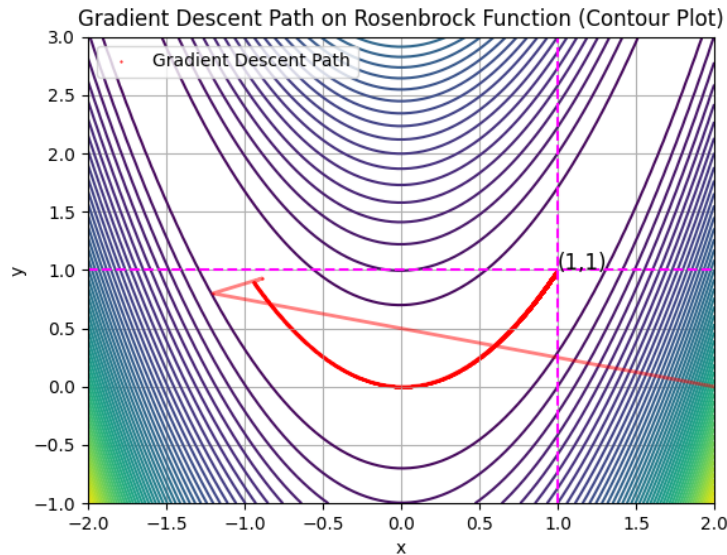


FIGURE 2.2: Representation of the path of the gradient descent algorithm used to find the global minimum on the Rosenbrock function

From a Loss Function

Unlike the previous example, which required knowledge of the function to be optimised to calculate its derivative, the use of a loss function obviates the need for this derivative

Gradient Descent Path on Rosenbrock Function (3D)

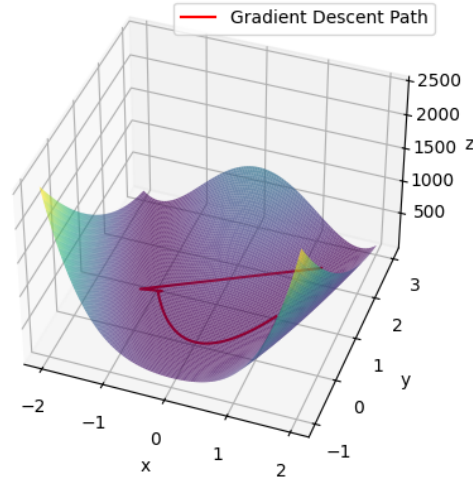


FIGURE 2.3: 3D representation of the path of the gradient descent algorithm used to find the global minimum on the Rosenbrock function

calculation. At each step, MAE is computed between the result obtained with the current parameters and the global minimum of the function, which is zero. Variables not redefined, such as the learning rate, are identical to those in the previous example.

After an equal number of iterations, the parameters obtained have the same values as in the previous example, namely 0.993 and 0.985, respectively.

```
import torch
param_x = torch.tensor(2.0, requires_grad=True)
param_y = torch.tensor(0.0, requires_grad=True)
loss_fct = torch.nn.L1Loss()
opt = torch.optim.SGD([param_x, param_y], lr=lr)

for _ in range(iterations):
    opt.zero_grad()
    result = rosenbrock(param_x, param_y)
    loss = loss_fct(result, torch.tensor(0.0))
    loss.backward()
    opt.step()
```

CODE LISTING 2.2: PyTorch implementation of the gradient descent algorithm for any function

Further Optimisations of Gradient Descent

Although gradient descent is an effective method to find the optimal parameters, it does not represent a single method, but, in fact, a family of algorithms [304]. While stochastic gradient descent updates the weights after each training example, which is what is computed in the example above, batch gradient descent computes the gradient over the entire dataset before updating the weights, and mini-batch gradient updates the weights every n samples.

In addition, gradient descent has been further improved. Momentum [284] can be added to dampen oscillations during the training process by increasing or decreasing the learning rate. Rather than using a single learning rate for all parameters, Adagrad [87] calculates a different learning rate for each parameter based on the cumulative sum of the previous gradients squared. Adam [175] extends Adagrad by incorporating two additional components: a first moment estimate, which is an exponentially decaying average of past gradients (non-squared), and a second moment estimate, which is an exponentially decaying average of past squared gradients. This allows Adam to balance the benefits of momentum and adaptive learning rates, mitigating the issue of vanishing learning rates in Adagrad.

Gradient accumulation is a technique used in training machine learning models, particularly deep neural networks, when the available computational resources (such as GPU memory) are insufficient to process a large batch of data at once. This technique allows simulation of training with large batch sizes without requiring the corresponding large memory.

During training, for each mini-batch, the gradients are computed, but instead of updating the model parameters at the end of the batch, these gradients accumulate over several iterations. After processing a predefined number of minibatches, the model parameters are updated using the accumulated gradients. In this way, the batch size can be increased, which could improve accuracy and stability.

We have explored the fundamentals of [Machine Learning](#) and showed how, from a simple set of algorithms, it is possible to create complex models. These complex models can even describe images or generate music. The branch of [Machine Learning](#) that focusses on language, and which will be our main interest in this thesis, is [Natural Language Processing](#).

2.1.5 Activation Function

The activation function in machine learning, especially in neural networks, plays a crucial role in determining the output of a node or neuron. It is a mathematical function that introduces non-linearity into the network, enabling it to learn and perform more complex tasks that linear functions cannot handle. Most real-world separations between classes are non-linear, meaning it cannot be separated using straight lines. Activation functions allow neural networks to process and learn from these non-linear data by introducing non-linear properties into the network. Moreover, activation functions are required for [DNNs](#), since without this added non-linearity, having multiple layers would be equivalent to having a single larger layer. Finally, activation layers such as [ReLU](#) limit the risk of gradients vanishing.

ReLU

The [Rectified Linear Unit \(ReLU\)](#) activation function is a widely used activation function in neural networks, particularly in the context of deep learning. Mathematically, it is defined as $ReLU(x) = \max(0, x)$.

Due to its simple max function, [ReLU](#) is computationally efficient compared to other activation functions such as sigmoid or tanh, which is partly why it was chosen for this work. It was also widely used in [Natural Language Processing](#), but it seems that other activation functions have been favoured recently, such as [Gaussian Error Linear Unit \(GELU\)](#), which is now the default activation function in HuggingFace. Figure 2.4 is a plot of the [ReLU](#) function from -6 to 6.

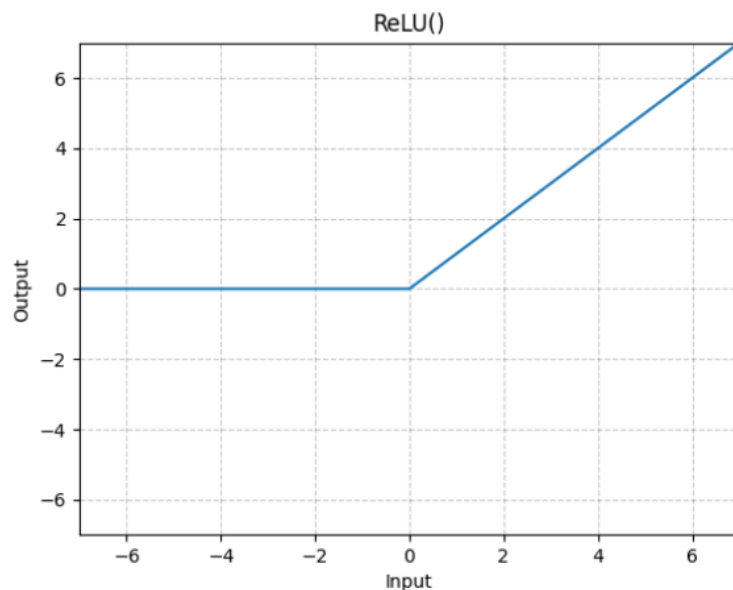


FIGURE 2.4: [ReLU](#) function, obtained from PyTorch documentation [280]

GELU

The [GELU](#) activation function [132] is a notable activation function in the field of machine learning, particularly within deep neural network architectures. The [GELU](#) activation function has gained significant attention due to its efficacy in various neural network models. Its mathematical definition is $GELU(x) = x \times \phi(x)$

At its core, the [GELU](#) function approximates a [ReLU](#) for positive values while dampening the outputs for negative values in a manner influenced by the Gaussian distribution. This unique characteristic enables it to manage the vanishing gradient problem more effectively than some traditional activation functions. The application of the [GELU](#) function has been shown to have remarkable success in transformer-based models, such as GPT and BERT. In these contexts, [GELU](#) has been instrumental in improving the models' learning capabilities. Its ability to balance between linear and nonlinear transformations makes it particularly suitable for complex tasks that require sophisticated feature extraction and representation.

One of the key advantages of the **GELU** function is its facilitation of smoother gradient flow during backpropagation. This attribute helps to alleviate issues related to training deep neural networks, such as vanishing or exploding gradient problems. Moreover, its probabilistic approach to activation, inspired by the Gaussian distribution, provides a more dynamic and adaptive activation landscape compared to traditional functions such as **ReLU** or sigmoid. Figure 2.5 is a plot of the **GELU** function from -6 to 6, obtained from PyTorch documentation.

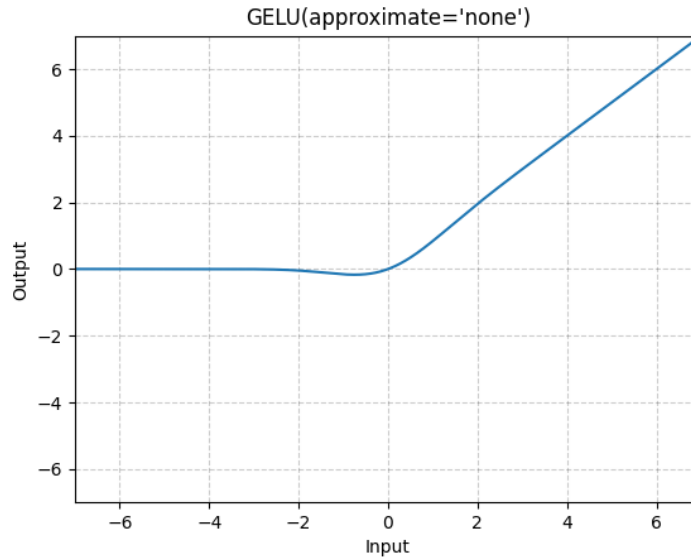


FIGURE 2.5: **GELU** function, obtained from PyTorch documentation [279]

SiLU

The **Sigmoid Linear Unit (SiLU)** activation function [90], also known as the Swish function, is a relatively recent addition to the family of activation functions used in neural networks. The **SiLU** function has gained attention for its performance in deep learning models. Mathematically, the **SiLU** function is defined as $SiLU(x) = x \times \sigma(x)$, where $\sigma(x)$ is the logistic sigmoid.

The sigmoid part of **SiLU** acts as a gate for the input, controlling its flow through the function. This self-gating property enables the function to adaptively scale the output based on the input, which can help in learning complex patterns in the data.

SiLU has shown promising results in deep learning models, often outperforming traditional activation functions such as **ReLU** in certain tasks. This performance enhancement is attributed to its ability to capture more complex dynamics due to its non-linear and self-gating nature. Figure 2.6 is a plot of the **SiLU** function from -6 to 6, obtained from the PyTorch documentation.

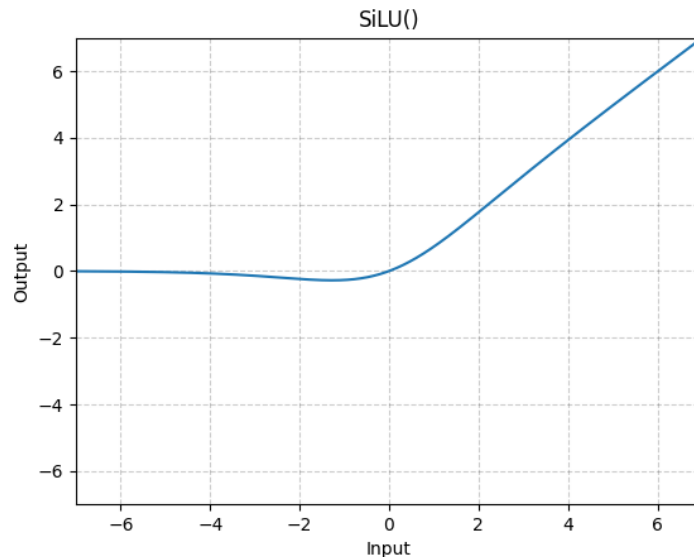


FIGURE 2.6: SiLU function, obtained from PyTorch documentation [281]

2.2 Natural Language Processing

As [Natural Language Processing \(NLP\)](#) became a study domain in the 1940s [164], multiple definitions have been presented over the decades, such as “the intersection of artificial intelligence and linguistics” [240] and “the computerized approach to text analysis based on both a set of theories and a set of technologies” [203].

[Natural Language Processing](#) aims to enable computers to interpret and manipulate human language and covers a variety of methods and activities to analyse and process written texts and spoken words. [Natural Language Processing](#) includes tasks such as [Machine Translation \(MT\)](#), speech recognition, named entity recognition, and text generation [152].

2.2.1 Tokenization

As text cannot be directly processed by Machine Learning algorithms, it first has to be converted into a numerical representation. Tokenization splits the text into smaller character sequences, including, but not limited to, phonemes, syllables, letters, words, or base pairs, collectively named *tokens*. The tokenization process also often includes adding special tokens, such as the unknown $\langle unk \rangle$ token to represent never seen characters, or the end of sequence $\langle eos \rangle$ token that is used to pad the sentence to a fixed length. Each token can be converted to and from a unique identifier, which is typically an integer, using a predefined vocabulary.

General Principle

Segmenting a corpus into individual words constitutes a basic form of tokenization, while dividing text at the character level represents an alternative approach. Although

these methodologies possess their respective utilities, more sophisticated algorithms have been devised to encapsulate greater semantic depth. In instances where a consecutive sequence of words frequently recurs within a language, akin to a prevalent phrase, assigning a singular token to this sequence facilitates the model’s ability to discern the unique connotation embodied by this specific amalgamation of words. Conversely, subword elements, such as morphological roots, can convey analogous semantic significance across diverse terms and are thus beneficial to tokenize accordingly.

The choice of the tokenization algorithm can significantly impact the performance of [Machine Translation](#) systems, depending on the target language pair [81] [417] [59].

[Byte Pair Encoding](#) and Unigram are two of the most common modern tokenization algorithms. Section 3.2.2 presents a deeper dive into the inner workings of these algorithms, along with tokenization examples.

Byte Pair Encoding

Subword-based tokenization creates subwords from words according to the number of occurrences of continuous character chains in the training data. Subwords can be combined to represent less frequent words or even words that were not present in the training data. [Byte Pair Encoding \(BPE\)](#) [317] was used for data compression long before it was applied in tasks related to [Natural Language Processing](#). After first counting the frequency of each unique word in the training data, [BPE](#) merges frequent occurrences of subword-pairs until it reaches the desired vocabulary size.

Unigram

Instead of starting from a small vocabulary representing the set of unique words and growing in size from there as in [BPE](#), Unigram [188] initialises its base vocabulary to a large number of symbols and then trims it down to the desired size. It is analogous to factor analysis, since at each step it calculates the loss of information that would be induced by removing each token and then erases the less important ones from its vocabulary.

Vocabulary Size

Words in a large enough dataset resemble a Zipfian distribution, also related to the discrete Pareto distribution and the Riemann zeta distribution, that is, “in any natural language corpus, we observe that the rank of a type is roughly inversely proportional to its frequency” [115], where a “type” refers to a unique word or token in a text corpus. Figure 2.7 shows examples of this distribution. Such a distribution causes both a class imbalance, which is a common problem in [Deep Learning](#) [163] and a risk of unseen vocabulary, as rare types might not be present in the training dataset [115].

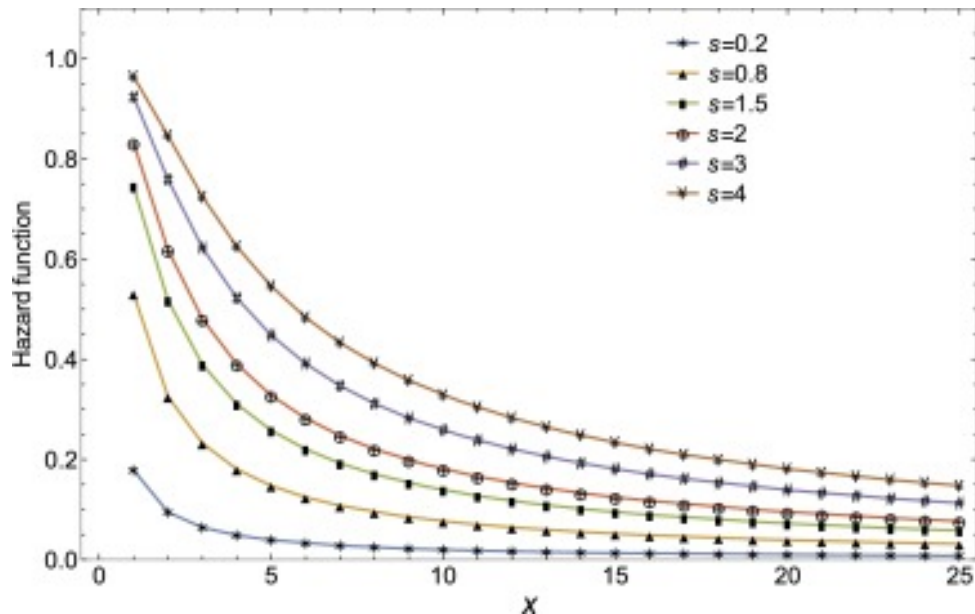


FIGURE 2.7: Examples of Zipfian distributions, obtained from [241]

Both BPE and Unigram expose a single hyper-parameter, the vocabulary size, which determines how many tokens exist in the vocabulary. The size of the vocabulary determines the size of the input embedding layer, which directly affects the computational requirements of the model. The embedding layer is a layer dedicated to word vectors, discussed in Section 2.2.2. As the impact of this hyper-parameter is often not well understood, the size of the vocabulary is often chosen arbitrarily [306]. In a Transformer model, a vocabulary size smaller than the width of the network constitutes a bottleneck that will reduce its performance [378]. However, this is not a problem in Natural Language Processing, since vocabulary sizes usually comprise at least 8,000 tokens [55].

Vocabularies of more than 30,000 tokens only obtain better results when the dataset is large. Vocabularies can be so large that it decreases the performance of a model [115], probably because some classes do not have enough examples in the training dataset, and a small vocabulary size also leads to a decrease in performance [195].

A useful heuristic is to “use the largest possible BPE vocabulary such that at least 95% of the classes have 100 or more examples in training” [115]. The optimal vocabulary size also depends on the language, “as a more morphologically rich language, needs a large vocabulary for the word model” [395].

In summary, the vocabulary size needs to be large enough to limit the impact of out-of-vocabulary tokens, while small enough to obtain enough examples for every token in the training dataset.

2.2.2 Categorical Variables

While the tokenization step transforms strings into integers, the integers do not form continuous variables. Indeed, the tokenizer does not order the tokens by semantic proximity or indeed any proximity at all, thus two completely unrelated tokens can have

adjacent identifiers. Tokens are a type of nominal variables, that is, categories that do not have any mathematical meaning and are without ordering. By way of distinction, even if age can be expressed as a discrete numeric variable [13], it is “continuous” in the sense that since it can take any numeric value, has meaningful intervals, and a true zero point.

Machine Learning is based on mathematical operations such as addition and multiplication, which require continuous variables. Since models expect to receive continuous variables, they interpret integers as ordered elements, which in the case of tokens would imply that the token $n+1$ is “greater than the token” n . In order to represent categorical variables and use them as model inputs, there are two main techniques for encoding them in a digital format: one-hot encoding and embeddings.

One-Hot Encoding

In one-hot encoding, each token in the sequence is represented as a binary vector with a length equal to the vocabulary size. All elements of the vector are 0, except for a single element that takes the value 1: the element that has the token identifier as its index. Figure 2.8 shows the representation of the sentence “The cat sat on the mat” where the vocabulary is “(cat, mat, on, sat, the)”. This approach is known as one-hot encoding.

		cat	mat	on	sat	the
the =>		0	0	0	0	1
cat =>		1	0	0	0	0
sat =>		0	0	0	1	0
...						

FIGURE 2.8: Representation of one-hot encoding, obtained from [114]

One-hot encoding is wasteful as the vectors created are sparse and mostly empty. While a sequence of 10 tokens with a vocabulary size of 10,000 necessitates 40 bits, or 5 bytes, when encoded as unsigned int32, its one-hot encoded representation would require 100,000 bits, that is, 12,500 bytes. Doubling the size of the vocabulary also doubles the memory footprint. Moreover, one-hot encoding is not a trainable technique and thus cannot be tailored to specific tasks.

Word Vectors

Word vectors are a crucial element of [Machine Learning](#) that encodes categorical elements into continuous values. In distinction to one-hot encoding, a word vector is a dense vector of fixed length that does not depend on the vocabulary size. This fixed length, also known as the embedding dimension, can vary depending on the specific task and usually varies between 512 [\[359\]](#) and 3072 [\[252\]](#) for [Natural Language Processing](#). Embeddings are tensor-based structures of dimension $(vocab\_size, embed\_size)$ similar to lookup tables, where the index of a word serves as the key to retrieve its corresponding word vector.

Rather than letting humans decide the attributes of each dimension, word vectors are computed by embedding layers that have trained weights that efficiently learn the semantic information appropriate for the task. The tokens are projected to this embedded space where similar tokens are positioned closer together than dissimilar ones.

Multiple methods of training word vectors have been developed over the years.

2.2.3 Training Word Vectors

Word vectors, which are not simply vectors of arbitrary values, but instead contain crucial information about the semantic meaning of words or tokens, can be generated using various training techniques. While [Bag-of-Words](#) is based on the frequency of occurrence of terms, more recent methods allow word vectors to be trained at the same time as the model, without having to go through an intermediate stage dedicated to them.

Bag-of-Words

[Bag-of-Words \(BoW\)](#) is a simple technique to extract features in order to create machine representation of text [\[283\]](#). A [BoW](#) model represents a document as a sparse vector, where each dimension corresponds to a word in the vocabulary, and the value represents its frequency in the document. This allows textual data to be transformed into a numerical form suitable for machine learning models. A [BoW](#) contains two arrays:

- The first is the vocabulary and it contains the complete list of unique words in the dataset;
- The second, which depends on the first, is an array of integers and contains the number of occurrences for each word in the vocabulary.

Since each document is represented as a vector of word frequencies, BoW disregards word order and syntactic structure, focusing solely on word occurrence patterns. This information can be mapped as a `HashMap`, a key-value data structure in which the keys are unique. The “dict” object is the native implementation of a `HashMap` in Python.

By default, **BoW** models are only a measure of the frequency of each word where all the weights are equal, i.e., all words are equally important. As a consequence, conjunctions such as “and”, “or” and “but” are similarly important as nouns and verbs. In order to take into account the relative importance of each word, **Term Frequency-Inverse Document Frequency (TF-IDF)** decreases the weights of words the more frequent they are. Many different weighting schemes exist to calculate the frequency of words, such as the relative frequency. Let t be the term, d the document, and $f_{t,d}$ the number of occurrences of the term t in d , the formula of the raw frequency is:

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}.$$

Neural Pre-Training

The meaning of a word can drastically change depending on its surrounding context, a property known as polysemy. As **BoW** only considers sentences as distinct words without any relationships between them, it fails to capture the different meanings of a single word. Neural Word Embeddings such as GloVe [269] and Word2Vec [234] improve upon **BoW** by encoding semantic relationships between words based on their co-occurrence patterns in large corpora. However, these models assign a single static vector representation to each word, meaning they do not explicitly capture polysemy. As a result, a word with multiple meanings, such as “bank” (financial institution or riverbank), is represented by the same vector regardless of context. Word2Vec is a group of models employing different techniques to extract representations of words:

- **Continuous Bag-of-Words (CBoW)** consists in predicting a word depending on its neighbours. This approach is faster to train [190];
- Skip-gram is the opposite task, in which the context has to be predicted from a word, a technique preferred for smaller datasets.

Given the sentence,

From ancient grudge break to new mutiny

and a span of 2 words, the construction of the **CBoW** dataset is represented in Table 2.1. The skip-gram dataset can be obtained by swapping the input and target.

Word2Vec uses a neural network-based predictive model that focusses on local context by predicting a word given to its neighbours (**CBoW**) or predicting neighbouring words given a target word (skip-gram). In contrast, GloVe uses a count-based model that leverages a global word-word co-occurrence matrix to capture statistical relationships between words and provides embeddings that reflect broader contextual relationships, albeit at a higher computational cost.

TABLE 2.1: Example of a CBoW training dataset

Input	Target
(ancient, grudge)	From
(from, grudge, break)	ancient
(From, ancient, break, to)	grudge
(ancient, grudge, to, new)	break
(grudge, break, new, mutiny)	to
(break, to, mutiny)	new
(to, new)	mutiny

Automatic Feature Learning

Instead of initially pre-training the embeddings, freezing the weights, and then integrating them into a neural network, it is feasible to train an embedding layer concurrently with the model. Consequently, the learning process for embeddings is directly influenced by the model’s tasks, such as translation, text generation, or classification, allowing the embeddings to capture the relevant semantic relationships and syntactic structures.

Training the embedding as part of the model rather than pre-training it separately is particularly effective when the training dataset accurately represents the data that the model will encounter and there is no intention to repurpose the embeddings for other tasks in the context of transfer learning.

In PyTorch, the *nn.Embedding* layer is initialised by default with random numbers between -1 and 1, sampled from the uniform distribution [282]. Depending on the task, it is also possible to use other initialisation functions [19] such as the Gaussian initialisation, the Xavier initialisation [109], which performs best with the activation functions of sigmoid and tanh, or the Kaiming initialisation [129] which is optimised for the activation functions of the ReLU family.

Similarly to other layers, the weights are then learnt through backpropagation of the entire network.

2.2.4 Legacy Model Architectures

Advancements in [Natural Language Processing](#) has been substantial in recent decades. The transition from rule-based methods and statistical methods to neural networks equipped with memory cells has drastically increased the performance of the model. Before examining the transformative impact of the Transformer architecture, it is important to review the previous [Machine Translation](#) techniques.

Rule-Based Machine Translation

[Rule-based machine translation](#) techniques were among the first [Machine Translation](#) methods to emerge [231]. These algorithms rely on dictionaries and sets of rules to

achieve translations. Although there are different approaches, the [RBMT](#) methods usually involve identifying grammatical elements such as nouns, verbs, determiners, adjectives, etc. The words are then mapped to the target language using the dictionary, and a set of rules is applied to change the order of the words if required.

For example, to convert the English sentence “The table is large” into French, it is first determined that it contains a determinant, a noun, a verb and an adjective. The words are then assigned to their French counterparts “La table est grand” and finally, since French is a gendered language, the rules would determine that “table” is a feminine word and as such require the adjective “grand” to agree in gender to the table and as such be converted to “grande”, resulting in the correct translation: “La table est grande”.

Statistical Machine Translation

[Statistical Machine Translation \(SMT\)](#) methods were dominant in the [Machine Translation](#) domain before recent [Neural Machine Translation \(NMT\)](#) advancements [366] [422]. These methods rely heavily on statistical models that do not require neural networks. Statistical methods operate based on the principle of analysing and interpreting large datasets to find patterns or statistical regularities, typically involving building models that learn the probability distributions of words, phrases, or sentences [40] [183] [57].

A fundamental approach in [SMT](#) is the Noisy Channel Model, where translation is framed as a probabilistic process: given a source sentence, the system searches for the most likely target sentence using Bayes’ Rule. [SMT](#) consists of two main components:

- the Translation Model, which estimates the probability of a source sentence being translated into a target sentence by analysing large-scale parallel corpora. It involves aligning words or phrases in a source language with their translations in a target language and then using these alignments to translate new sentences.
- the Language Model, which ensures fluency in the target language by modeling the probability of word sequences based on monolingual text.

Although statistical methods are effective to some extent, they have limitations, especially in handling the complexities and nuances of natural language. They often struggle with long-range dependencies in text [366] and idiomatic expressions.

Google adopted [Neural Machine Translation](#) in 2016 [395] [355] in their popular “Translate” product, and, in general, [Neural Machine Translation](#) models have been rapidly adopted because they excel in capturing context[370].

Recurrent Neural Network

A [Recurrent Neural Network \(RNN\)](#) [137] is a neural network architecture specifically designed for the processing of sequential data through the inclusion of feedback connections. Unlike feedforward neural networks that facilitate unidirectional information flow

from input to output, RNNs include connections that enable cyclic information flow, thus facilitating the maintenance of an internal memory or state. This connection is shown in Figure 2.9.

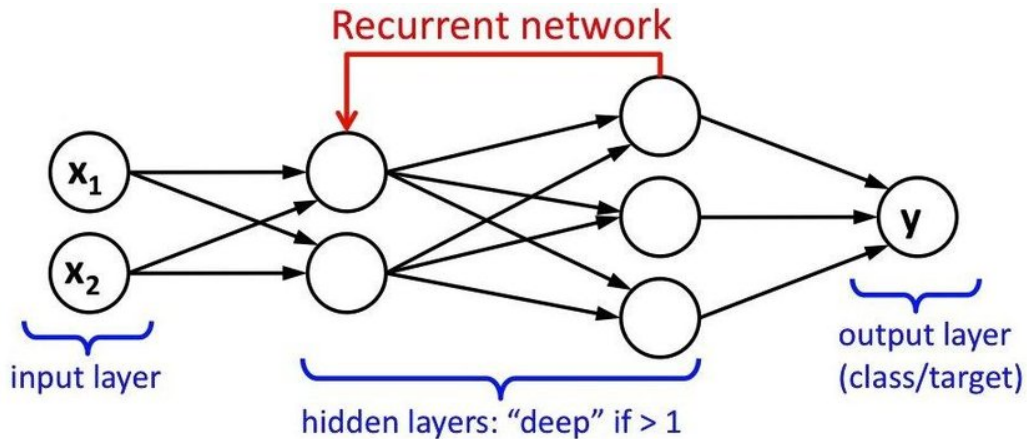


FIGURE 2.9: Architecture of a RNN feedback connection, obtained from Mishra et al. [237]

The fundamental concept underlying RNNs is their ability to handle input sequences of variable lengths and to acquire the capacity to identify temporal dependencies or patterns within the data. RNNs find widespread application in tasks involving sequential or time-dependent data analysis [370] [221] [315] [415], including Natural Language Processing domains [168] [397] such as speech recognition [233] [138] [323] [309] and Machine Translation [58] [327] [212].

In a recurrent neural network, information cycles through a loop. When a RNN makes a decision, it considers the current input and also what it has learnt from the inputs it received previously. This is crucial in tasks where context is important for understanding an input fully. In language, for example, the meaning of a word can depend greatly on the words that come before it (or indeed the words that come after it).

The most important part of a RNN is the hidden state, which captures some information about a sequence and is used as a form of memory by the model. At each time step in the sequence, the RNN updates its hidden state using both the new input and the previous hidden state. This process is repeated for each element in the input sequence, and the hidden state effectively transmits information throughout the processing of the sequence.

The calculations performed by an RNN in each step are generally quite simple. The network typically combines the current input with its current state (a summary of previous input) through a set of weights (parameters) that are learnt during training. This combination, usually involving operations like addition and multiplication, produces a new state and a new output.

RNNs have a significant limitation: As they process longer sequences, they can suffer from short-term memory. If a sequence is too long, they might have trouble carrying

information from earlier time steps to later ones, a problem known as “vanishing gradients”, which arises when the derivatives from the activation functions become close to zero, resulting in numerically small or zero small gradient values.

Long Short-Term Memory

Long Short-Term Memory (LSTM) [136] neural networks are a type of RNN that was specifically designed to handle long-range dependencies. As they retain information for a longer period and “forget” “unnecessary” data, they are adapted for the Machine Translation domain [360], where they perform better than RNNs [327].

LSTMs contain one or multiple cells, which themselves use multiple gates and a cell state, that is, memory of the previous state. The *input* gate controls what new information will be stored in the cell state, while the *forget* gate determines what information from the previous state should be discarded. Without that *forget* gate, the cell would grow too large and contain outdated information. Finally, the *output* gate controls which part of the cell state will be transferred to the next layers of the neural network. Figure 2.10 shows the internal architecture of an LSTM cell.

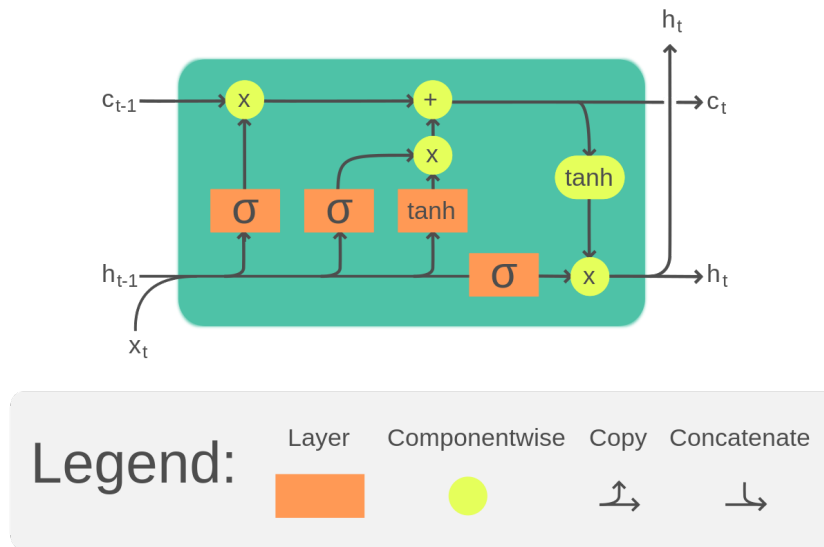


FIGURE 2.10: Architecture of an LSTM cell, obtained from [239]

During training, the LSTM adjusts the parameters of these gates to optimise the flow of information. This process is guided by backpropagation through time, where the network’s predictions are compared to the actual results, and the difference (the error) is used to update the weights in the network, improving its performance over time.

Gated Recurrent Unit

The Gated Recurrent Units (GRU) [58] architecture simplify the structure of LSTM cells, making them more efficient in certain scenarios such as polyphonic music modeling [63], especially when dealing with smaller datasets or when computational efficiency is

a key concern. [GRU](#) have also been used in [Machine Translation](#) [199] [70] [420] [324] [320].

In a [GRU](#) cell, there are only two gates: the update gate and the reset gate. These gates help the model to decide which information should be passed to the output. One of the key differences between a [GRU](#) and a [LSTM](#) cell is that the [GRU](#) does not have a separate memory cell, only the hidden state. The schematic representation of a [GRU](#) in Figure 2.11 shows how closely related [GRU](#) and [LSTM](#) cells are.

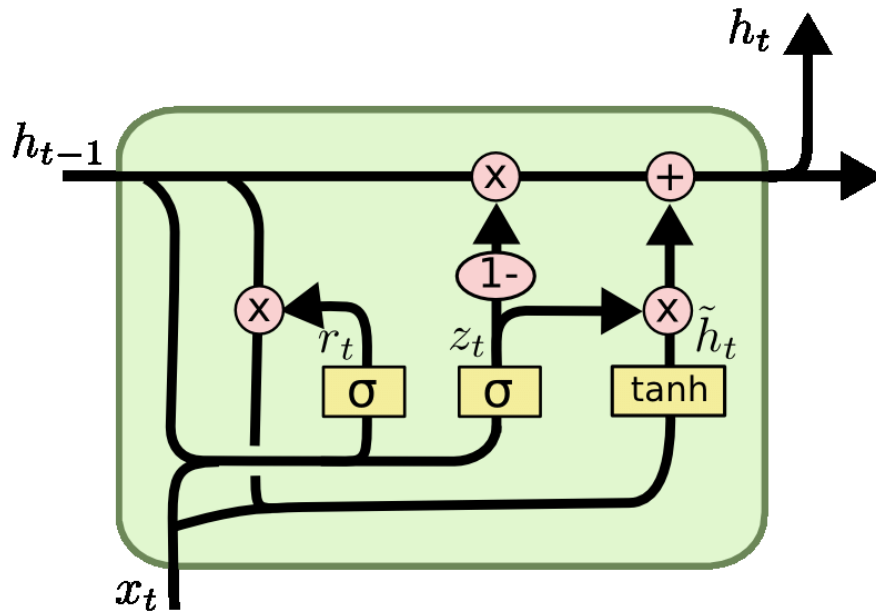


FIGURE 2.11: Architecture of a [GRU](#) cell, obtained from [3]

We have explored how neural architectures, such as [LSTM](#) or [GRU](#), are capable of processing sequences of data, using dedicated vectors used as a form of memory. However, research and industry trends have increasingly favoured the Transformer architecture due to its ability to better parallelise computations, which facilitates scaling to larger models with more parameters.

2.3 Transformer

In 2017, the “Attention is All You Need” study by Google [359] introduced a novel [Machine Learning](#) architecture in the field of [Natural Language Processing](#), the Transformer model, described [34] as one of the “foundation models” in [Machine Learning](#).

As shown in Figure 2.12, the transformer architecture consists of two main components which can be employed independently or in combination: the encoder and the decoder. The encoder consists of a stack of identical layers. Each layer has two sub-layers: a multihead self-attention mechanism and a position-wise feed-forward neural network. The decoder also consists of a stack of identical layers, but with an additional sublayer that performs multihead attention over the encoder’s output. This allows the decoder to focus on different parts of the input sequence during the decoding process.

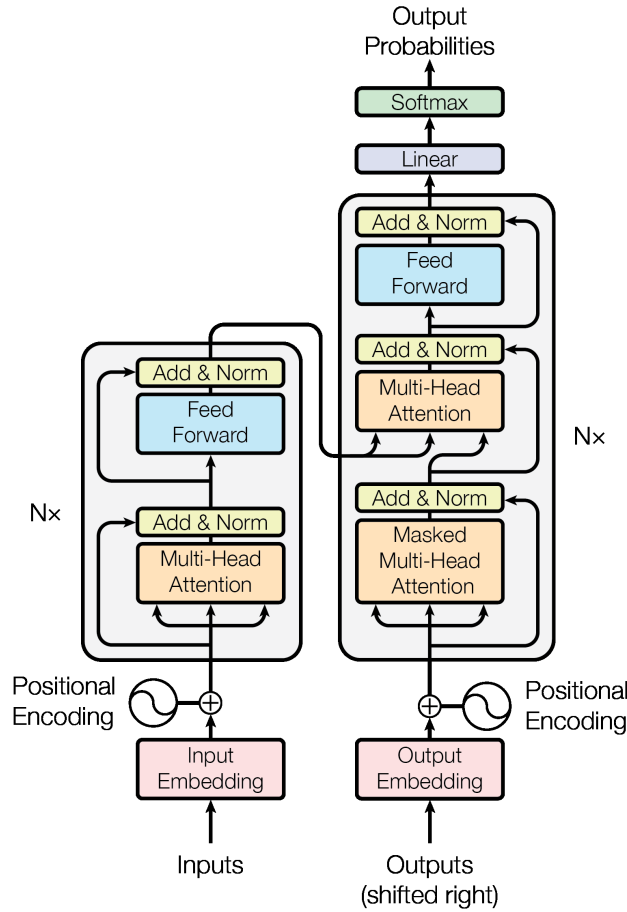


FIGURE 2.12: Architecture of a Transformer model, in its original encoder-decoder version, obtained from [359]

The input tokens are fed to the encoder, which will then output a vector of the same length as the input. The connection between the encoder and the decoder is taken care of by the cross-attention block. The decoder then auto-regressively predicts the target sequence, token by token, either using the output of the encoder in an encoder-decoder transformer model, or from its embedding layer in a decoder-only model. The encoder-decoder architecture is particularly effective in tasks such as translation, as the encoder transforms the input sentence into an abstract representation, and the decoder will be in charge of generating a sentence in the target language.

Since the creation of the Transformer architecture, it has become a dominant architecture in the [Machine Translation](#) world [206]. For example, the Transformer model is used in Co-DETR [437], which is the current State-of-the-Art in Object Detection on the COCO dataset [207], in HR-CEUTrack-Large [434] for Multi-Object tracking on the COESOT dataset [345], and in Transformer (cycle) [344] on the WMT2014 English-German dataset [223]

2.3.1 Attention

The key idea behind the transformer architecture is the use of self-attention mechanisms to capture relationships between all different words in the input sequence. This contrasts

to previous [Neural Machine Translation](#) architectures, such as [RNNs](#) and [LSTMs](#) in which words are only connected to their nearby neighbours.

Rather than employing fixed embeddings for individual tokens, the mechanism of self-attention calculates a weighted average of each embedding using dot products between the embeddings, which is a measure of similarity. This product can be scaled [\[38\]](#) to avoid the absolute value of the dot products becoming so large that applying softmax leads to vanishing gradients [\[31\]](#). This allows the model to weigh the importance of different words or tokens in the input sequence when processing each word or token, and enables it to capture the dependencies between tokens regardless of their relative distance. Moreover, as the embeddings depend on the embeddings of all other words in a sequence, the representation of polysemous words can vary according to the context.

The attention mechanism involves three vectors for each token in the input sequence: query (Q), key (K), and value (V). In order to explain what these vectors represent, the analogy obtained from [\[408\]](#) illustrates it as follows: Someone in a library has a specific question (Q) and is looking for an answer. They skim through the titles (K) of the books, and using these titles, they decide how relevant each book is and how much attention to give it. Finally, the information (V) is obtained by reading the selected books in order to answer the question. The self-attention mechanism takes its name from the fact that queries, keys, and values are obtained from the same source [\[46\]](#), as they are obtained from the tokenized input sequence. However, they are obtained through different embeddings with different weights.

For each element in the input, an attention score is computed by taking the dot product of its Q vector and the K vectors of the other elements. The key vectors help the model to “understand” the relationship of each token with the input. More attention is paid to similar QK pairs, as the dot product is a measure of similarity between two vectors. The softmax function converts these attention scores into a probability distribution whose total sum is 1. Then it generates the weighted sum associated with each word, where the weights are determined by the relevance or importance of each word to the others. This calculation is shown in [Figure 2.13](#).

The attention mechanism is computed by “heads” to capture contextual relationships between words. Attention scores determine how much focus or “attention” should be given to other parts of the sequence when encoding or representing each specific element.

An advantage of the self-attention mechanism is its ability to process all elements of the input at the same time, unlike [RNN](#) and [LSTM](#) which process the input sequentially, where the first token is processed to obtain the first hidden state, which is used with the second token to derive the second hidden state. Parallel processing in transformer networks speeds up computation and allows the model to capture long-range dependencies in the data more effectively.

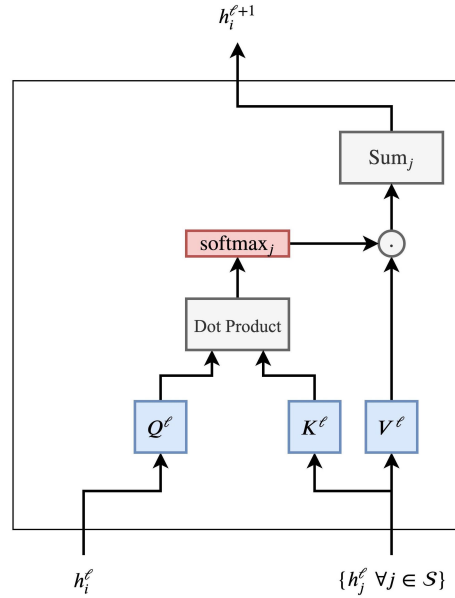


FIGURE 2.13: Representation of the attention mechanism pipeline, which relies on the triplet of tensors (Q, K, V) to attend the elements of the sequence, obtained from [47]

Multi-Head

The use of multiple heads allows the model to simultaneously examine several positions at once, expanding the capacity of the model to understand the sequence in different ways. Indeed, by computing multiple sets of (Q, K, V) points of view, some heads can focus on semantic relationships, others concentrate on syntactic relationships, and others again might focus on certain types of words like nouns or verbs, without loss of information. As there are multiple heads, each one of them can have a unique perspective on the relationships between words. This is analogous to having multiple pairs of eyes, where each set focusses on a different part of the input, or on interpreting it in a different way.

After all heads have processed the input in parallel, their outputs are concatenated. This concatenated output ensures that the information captured by each head is combined and used in subsequent layers of the transformer.

Cross-Attention

The cross-attention mechanism is at the intersection of the encoder and the decoder. It is a variant of self-attention, where the three tensors (Q, K, V) are not obtained from the same source. In fact, the key and value vectors are gathered from the final encoder layer, while the query tensor is obtained through the previous layer in the decoder.

2.3.2 Minor Improvements

Although the self-attention mechanism is at the heart of the Transformer and is the major innovation of this architecture, other important techniques are at work to make

the Transformer so effective. Already used in other [Natural Language Processing](#) models, these operations have been repeated to improve training stability and learning speed.

Skip Connections

Skip connections, also called residual connections, were introduced for the first time in the ResNET [130] architecture. A connection of this type allows the output (or part of the output) of a layer to skip at least one layer and then to merge with the output of the skipped layers. This merging can take multiple forms, such as concatenation, addition, multiplication, or taking the average. Deep networks can suffer from vanishing gradients [264] and also usually improve the “flow” of gradients [395]. In Transformer models, skip connections can be used to preserve the original sequence to ensure that no word is left behind or forgotten by the self-attention mechanism.

Dropout

Dropout [334] is a technique designed to prevent overfitting, which is when a model gives excellent predictions in the training dataset, but obtains poor results in the test set. This can happen when a complex model adapts to the noise in the training data but also when a large model has enough parameters to remember the entire training data.

To quote the authors, “Applying dropout to a neural network amounts to sampling a ‘thinned’ network from it”. By randomly dropping units, that is to say removing the connections towards and from these cells, the remaining cells have no choice but to learn how to generalise features in the presence of unreliable cells. Figure 2.14 displays the effect of applying dropout on linear layers.

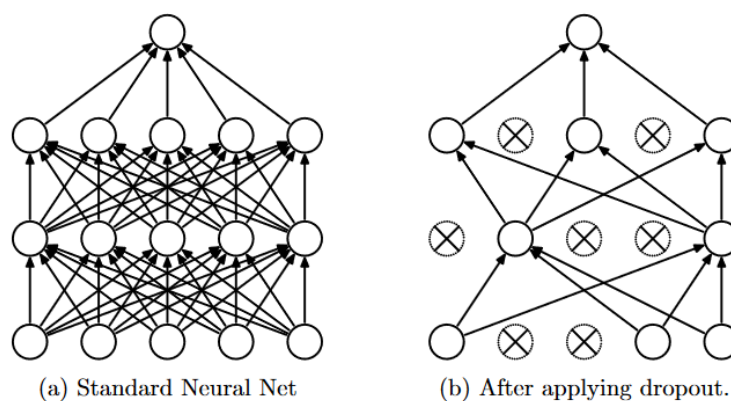


FIGURE 2.14: Representation of the dropout mechanism, which disables random parameters during training, obtained from [334]

It is crucial to note that this dropout mechanism is only activated during training. At inference time, the dropout rate, that is, the probability of a cell to be dropped, is set to 0.0. In this way, there is no risk of a particularly important cell disappearing when it is most needed, i.e. at inference time.

Normalisation

In order to make the inputs to a neural network more consistent, normalisation techniques are applied. Data are often noisy and come in varying scales and ranges. Normalisation helps adjust the scale and range of the data, thereby promoting uniformity across the dataset. As an example obtained from [62], two different features can have drastically different ranges, such as age, which usually varies between 0 and 100, and income, which can be in the billions. When two features have such large range disparities, the performance of models trained on these data can be degraded. Similarly, when the dataset is made up of images, the colour values are often normalised and scaled so that, regardless of the luminosity and intensity of the images, they are standardised to a stable format and to reduce internal covariate shift.

This normalisation is generally applied upstream of the neural network during pre-processing. In this way, the model does not know that the colours vary other than between 0 and 1. Many different mathematical equations can be used to normalise the data, depending mainly on the desired range, such as:

- Z-score normalisation: To obtain a distribution towards a mean of 0 and a standard deviation of 1, the transformation $x' = \frac{x-\mu}{\sigma}$ is used, where x is the original value, μ is the mean of the original data and σ its standard deviation. A small constant ϵ is often added to the denominator to increase numerical stability;
- Min-Max scaling: The application of rescaling compresses the data into a fixed range. The formula for obtaining a range of $[0, 1]$ is $x' = \frac{x-\min(x)}{\max(x)-\min(x)}$, where x is the original value, $\min(x)$ is the smallest value and $\max(x)$ the maximum.

Figure 2.15 shows the effect of Z-score normalization on different distributions.

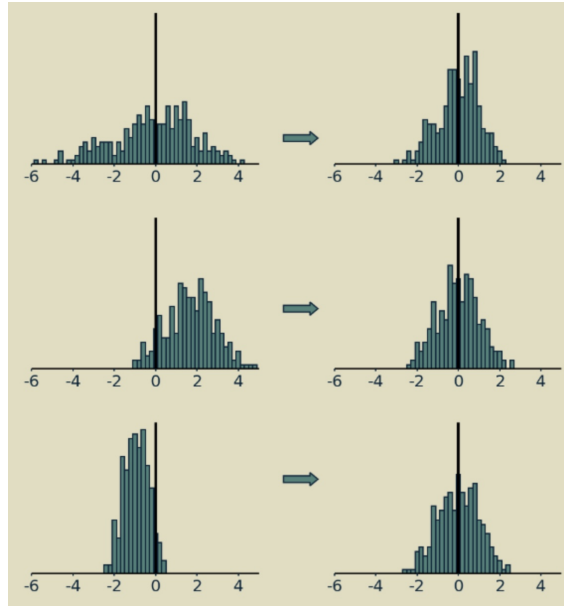


FIGURE 2.15: The effect of normalisation on different distributions, concentrating values around a mean, obtained from [46]

Normalisation steps can also be integrated directly into networks, such as batch normalisation [154]. Since neural network learning requires the dataset to be broken down into batches, if the inter-batch variation is significant, massive differences in performance can occur between batches. To remedy this problem, the Batch Normalisation layers record running means and running deviations of the batches during training, before normalising them, which has the effect of standardising across the batches. Batch normalisation is widely used in Computer Vision neural networks, such as ResNET [130], VGG [326], MobileNet [139], and YOLO [292].

Another type of intra-network normalisation is Layer Normalisation [18]. While Batch Normalisation applies normalisation over the entire batch, Layer Normalisation normalises all features within each sample. Figure 2.16 shows how batch normalisation and layer normalisation differ, and how Layer Normalisation removes the dependence on batches. Layer normalisation has been crucial in the success of the Transformer architecture [400], as the added stability of the distribution accelerates training and the effectiveness of this architecture.

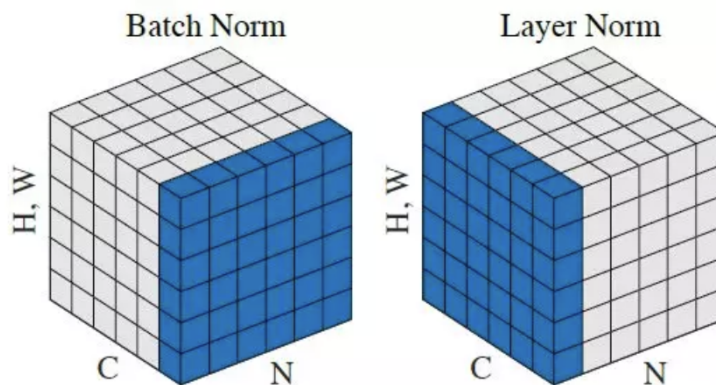


FIGURE 2.16: Layer Normalisation normalises across features within a single data point, ensuring consistent scaling for each input sample, while Batch Normalisation normalises across the batch dimension for each feature, stabilising feature distributions across the batch. In this Figure, obtained from [18], C represents the number of channels or the input and N represents the batch size.

Teacher Forcing

Normally, a decoder generates a sequence element by element, using its previous outputs as inputs for subsequent predictions. Teacher forcing [390] [108] is a technique that involves substituting the predicted token with the actual ground truth at each stage of the decoding process. This approach is designed to mitigate the cascading impact of early erroneous predictions in the sequence.

Instead of using the model's own predictions at each step, the correct output from the training data is provided as the next input. For instance, in a task where the model is

translating from English to French, for each prediction the model is given the true next token in the French sentence during training instead of the token it just predicted.

This technique has two main benefits: it prevents error accumulation during training by continually guiding the model in the correct sequence, and it accelerates the training process by providing the model with direct and accurate feedback. Teacher forcing has been adopted by very popular architectures in [Neural Machine Translation](#), such as T5 [288].

The Transformer architecture is a highly efficient encoder-decoder model, with an attention mechanism that makes it particularly efficient for tasks in the domain of [Natural Language Processing](#), from text translation to summarisation or question answering. We will now explore how this base architecture can be modified depending on the task.

2.4 Transformer Tasks

As well as being particularly computationally optimised due to its parallel calculations and high performance due to its self-attention and multi-head mechanism, the transformer architecture has another massive benefit: modularity. Although it is initially an encoder-decoder architecture, depending on the task, a model may only use the encoder, the decoder, or both. So, a Transformer can be adapted differently depending on whether you want to obtain a translation model, a classification model, or a text generation model.

2.4.1 Sequence-to-Sequence

[Sequence-to-Sequence \(seq2seq\)](#) [343] initially refer to an architecture containing an encoder, which is responsible for reading and processing the input sentence, and a decoder, which focusses on generating an appropriate output. The [seq2seq](#) architecture is well adapted for [Machine Translation](#), speech-to-text and text summarisation. Both the encoder and decoder are implemented using multiple chained [RNNs](#). Attention mechanisms have been implemented in [seq2seq](#) models [20].

However, the meaning of [seq2seq](#) has evolved and is now more general, encompassing any task that involves the transformation of a sequence from a source distribution to a sequence of a target distribution.

Transformers can be configured to work similarly to the [seq2seq](#) models with some key advantages. In fact, the original architecture [seq2seq](#) could only process the input sequentially, as it relied on [RNN](#) cells. Modern architectures rely on Transformer models, which can process the input tokens in parallel. The original Transformer model [359], which contains both an encoder and a decoder, is a [seq2seq](#) model.

2.4.2 Language Modelling

Language Modeling (LM) involves predicting the probability distribution of a sequence of words. For this task, the model must be able to predict a token according to the other tokens, i.e., sort the tokens in its vocabulary to find the one most likely to correspond to the specified position. There are two main forms of **LM**:

- **Causal Language Modeling (CLM)**, also known as autoregressive **LM**, is a training method in which the model is tasked with predicting the next token given the preceding tokens. In this condition, the model cannot attend to tokens that follow the token to predict, adhering to the forward-flowing causality of natural language. This type of training is particularly useful for text generation, as each generated token can only be predicted from preceding parts of the sequence. **Next Token Prediction** is similar to **CLM**, but can also be bidirectional, that is, the model has access to both previous and future tokens;
- **Masked Language Modeling (MLM)** involves reconstructing an input sentence where at least one token was replaced by a special `<mask>` token. The model may or may not have access to the entire sequence to predict the masked tokens.

GPT

The first **Generative Pre-trained Transformer (GPT)** [286], introduced in 2018 by OpenAI, is usually recognised as the first **LLM**. **LLMs** are discussed in more details in Section 2.4.4. **GPTs** are a class of **Natural Language Processing** models, based on the Transformer architecture, that are pre-trained on a single-token generation task.

As an autoregressive model, that is, each prediction depends on the input and the previously generated tokens, the **GPT** models are trained on unlabelled datasets in an unsupervised manner on **CLM**. During training, a random sentence is drawn from the dataset: the first n tokens are used as input, and the output will be compared to the $n+1$ token of the original sentence to obtain a loss and compute the gradients.

At the time of generation, each predicted token is concatenated with the list of input tokens. This step is repeated until a predefined size limit is reached, or until the most likely predicted token corresponds to a special token signifying that the answer is complete.

Unlike traditional transformer models, the **GPT** models are based on a decoder-only architecture. The absence of a receiver reduces the number of parameters in the architecture and simplifies the design, while not hindering its generative capabilities. Although an encoder is required in the **seq2seq** tasks, the autoregressive nature of **GPT** means that each token is only predicted from the previous ones and not from a contextual representation of the input generated by an encoder.

A notable example of a **GPT**-based application that has recently gained popularity is ChatGPT, a conversational system built on a proprietary **LLM** fine-tuned for tasks

related to answering user questions. The underlying model was fine-tuned with instruction tuning, using datasets containing human-written responses to align its outputs with user instructions. This increased interest [213] in generative AI was an important showcase of recent innovations in Natural Language Processing for the general public.

Although GPTs can be used as text generation models, this pre-training also allows them to be efficiently fine-tuned for different Natural Language Processing-related tasks such as semantic similarity assessment [286] and text classification [286] [287].

LLaMA

The Large Language Model Meta AI (LLaMA) [353] is a LLM developed by Meta and released in 2023 in four different versions, varying in the number of parameters: 4, 13, 33 and 65 billion. The LLaMA2 [354] model was also released in 2023, in versions of 7, 13, and 70 billion parameters. The LLaMA model is a Transformer architecture using RMSNorm [419] as pre-normalisation and Rotary Position Embedding [338]. As a Transformer model pre-trained on CLM, LLaMA should be considered as part of the GPT family. The 7 billion parameter model encompasses an embedding layer, 32 blocks, and a projection layer. Building upon the previous iterations, Meta released LLaMA3 [86], with models comprising 8 billion and 70 billion parameters. These models were pre-trained on approximately 15 trillion tokens, with instruction-tuned versions fine-tuned on over 10 million human-annotated examples.

2.4.3 Single-Sequence

Single-Sequence tasks focus on analysing an input sentence and deriving a fixed output. These tasks involve for example text classification and sequence labeling. Text classification assigns a single label to an entire input sequence. Examples include sentiment analysis, topic classification, or spam detection, where the model predicts one category for the whole text. Sequence labeling assigns labels to individual tokens or spans within the input. Tasks such as Named Entity Recognition (NER) fall into this category, as they require each word in a sequence to be tagged accordingly.

Within the domain of the Transformer architectures, single-sequence tasks predominantly only rely on the encoder. The encoder processes the input sequence through multiple layers of self-attention mechanisms and feedforward neural networks, generating a collection of context-rich embedding.

BERT

Bidirectional Encoder Representations from Transformers (BERT) [80] is a pre-trained model based on the Transformer architecture, developed by Google and released in 2018. Unlike the GPT models, BERT is an encoder-only architecture. Rather than focussing on being a generative architecture, BERT is targeted at capturing a nuanced

understanding of a sentence. As a bidirectional model, it simultaneously processes the input from left-to-right and right-to-left to integrate context before and after each token.

[BERT](#) has been used in a wide variety of [Natural Language Processing](#) tasks [301] [186] such as sentiment analysis [45] [134] [9] [332], question answering [416] [285] [377] and text ranking [205], where the objective is to retrieve a list of texts from a corpus in response to a query.

Unlike [GPT](#), [BERT](#) was not trained on [NTP](#) but rather on two different tasks, [Next Sentence Prediction \(NSP\)](#) and [Masked Language Modeling \(MLM\)](#).

NSP Unlike [NTP](#), [Next Sentence Prediction \(NSP\)](#) is a classification task in which the model predicts whether two spans of text appear sequentially in the dataset. [NSP](#) requires four special tokens: the [CLS] symbol, which indicates that the query is a [NSP](#) task, the [SEP] token, which indicates the end of the first span of text and the beginning of the second, and [IsNext] and [NotNext], the two possible outputs of the models. The examples in the following are obtained from [384].

- Given "[CLS] my dog is cute [SEP] he likes playing" the model should output token [IsNext];
- Given "[CLS] my dog is cute [SEP] how do magnets work" the model should output token [NotNext].

MLM The [Masked Language Modeling](#) task involves selecting a set percentage of tokens in the input sentence and replacing them by a [MASK] token or a random word token, or not replacing them at all, and training the model to recover the original sentence.

Even if [BERT](#) is not primarily designed for text generation, as it is entangled with a [LM](#) task that involves predicting tokens, it could be categorised with other [LM](#) models alongside [GPT](#) and [LLaMA](#). However, since [BERT](#) is an encoder-only architecture and is used to understand and analyse input, our opinion that it is more appropriate to categorise it in the Single-Sequence category.

RoBERTa

[Robustly Optimized BERT Approach \(RoBERTa\)](#) [214] is an architecture derived from [BERT](#) designed for single-sequence tasks. In their study, the authors found that the reduction of the [NSP](#) task did not degrade downstream performance; thus, they only trained [RoBERTa](#) on [Masked Language Modeling](#). [RoBERTa](#) was also trained on more data using larger batches, allowing this architecture to perform better than its predecessor in downstream tasks such as text classification [4] [37] [71] [335].

ELMo

Embeddings from Language Models) (ELMo) representation [270] are embeddings generated by a bidirectional LSTM that was trained on two different tasks:

- Forward Language Model, which involves predicting the next token based on the previous ones, a task also known as NTP, similarly to GPT;
- Backward Language Model, that is to say predicting the previous token given the following ones.

This bidirectional nature allows ELMo to encode richer contextual information, capturing both past and future dependencies in a sequence. The representations obtained from ELMo have been used for tasks such as NER [5], text classification [216] [314] and sentiment analysis [89] [41].

Contrary to Word2Vec and GloVe, embeddings generated by BERT, RoBERTa and ELMo produce different vector representations for the same word depending on its context, which helps capture polysemy.

2.4.4 Large Language Model

A Large Language Model (LLM) is a Natural Language Processing model that relies on the Transformer architecture and comprises a large number of parameters, usually in billions. Although they were initially only trained on text datasets, they have since been adapted to work with many data types, including images and videos. LLMs can perform diverse Natural Language Processing-related tasks, such as text generation, summarization, translation, or answering questions. They usually have a large context window, which means that they can handle a large input. For example, LLM chatbots can take into account the entire history of a conversation to answer more precisely, rather than only using the last question as input. A Foundation Model refers to a large-scale, pre-trained AI model that serves as a general-purpose system, which can be fine-tuned to a wide range of downstream tasks. Using this definition, LLMs are a subset of foundation models dedicated to Natural Language Processing and can even have multi-domain capabilities.

As shown in Figure 2.17, the number of parameters in LLMs has drastically increased in the last few years, some models even exceeding a trillion parameters [85] [297].

2.4.5 Vision Transformer and Multimodal Models

Although vision transformers and multimodal transformers represent significant advances in transformer architecture, their application areas diverge from the core focus of this work. Vision Transformers adapt the transformer model for image processing tasks,

2.5.1 Dataset Format

[Machine Translation](#) models are most commonly trained using “parallel” datasets, which are structured collections of pairs of text. Each pair comprises a segment of text in a source language and its translation in the target language. By providing direct translations, the models learn the correspondences between text units to map the source language to the target language.

As the quality of the datasets significantly affects the performance of [Machine Translation](#) models, parallel datasets are carefully curated to ensure that the semantic and contextual integrity of the original content is maintained in the target sentences.

The characteristics of datasets may exhibit variability not only in size and quality, but also in thematic scope. Indeed, whereas certain corpora strive to encompass a broad spectrum of content, aspiring towards comprehensive linguistic diversity, others delineate a narrower focus, concentrating on highly specialised subjects such as legal, medical, or technical discourses. These corpora facilitate the fine-tuning of models, enhancing their precision in executing tasks within specific domains.

Parallel corpora also serve in the evaluation of [Machine Translation](#) models. By comparing a generated translation of a source sentence with its target sentence in a parallel test dataset, it is possible to evaluate the quality of the [Machine Translation](#) system.

2.5.2 Evaluation

Evaluation metrics are critical for assessing the quality of machine-generated outputs. By providing an objective and reproducible evaluation of generated translations, they enable comparisons between datasets and architectures.

Models are usually tested at set intervals during the training process, and this ongoing assessment of performance facilitates the cessation of training upon satisfactory results, or the determination of when training does not further improve the accuracy of the model.

During the development of [Machine Translation](#) systems, metrics help to determine the optimal parameters and diagnose the limitations of an architecture. Depending on the approach on which they rely, metrics have different profiles and characteristics and can capture different aspects of translation. Determining whether a system preserves meaning at the cost of lexical accuracy can be crucial in orienting further research.

Non-Neural Metrics

Although the following metrics are not the most recent, traditional metrics still provide valuable insight. In fact, they are still used as a point of comparison for newer metrics and [Machine Translation](#) systems, and are often used in conjunction with more modern evaluation methods, as they are computationally inexpensive. Since they do not require

training, they are compatible with a wide range of languages, which is particularly important when evaluating translations from low-resource languages.

BLEU The score [Bilingual Evaluation Understudy](#) [260] evaluates the quality of [Machine Translation](#) by comparing the [Machine Translation](#) output to reference translations. It measures how well the [Machine Translation](#) output matches the references using n -grams, which are contiguous sequences of n text items, these items being words in the context of [BLEU](#). [BLEU](#) calculates the precision for each length of n -gram, usually 1 to 4, and combine them into a final score considering brevity.

Although [BLEU](#) was widely used, it has limitations, as it focuses primarily on exact matches and does not address issues such as word order and meaning.

ROUGE Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [204] is a metric commonly used to evaluate the quality of text summarisation. In its default form, the ROUGE score is calculated by dividing the number of matching words in the reference sentence (or summary) by the total word count in that reference. Variations include ROUGE-L, which focusses on the longest common subsequence, and ROUGE-N, which is similar to [BLEU](#) as it also measures the overlap of n -grams but relies on recall instead of precision.

Although this metric is not used primarily in [Machine Translation](#), it provides useful information by focussing on recall instead of precision.

METEOR The Metric for Evaluation of Translation with Explicit ORdering (METEOR) [25] metric relies on the harmonic mean of precision and recall for unigrams (a specific case of word n -grams where $n = 1$). Additionally, it incorporates traditional exact word matching along with unique features not present in previous metrics, including stem (the root form of words) and synonym matching.

The design aim of METEOR is to address the issues identified in the widely used [BLEU](#) metric and it aims to achieve a strong alignment with human evaluations, particularly at the sentence or segment level.

TER Translation Edit Rate (TER) [329] is a [Machine Translation](#) score that measures the minimum number of operations, such as deletions, substitutions, or insertions, that have to be applied to the output of a system to obtain the reference translation.

As TER directly measures the cost of correcting a machine-generated output rather than assessing quality, it is usually used in conjunction with other metrics.

chrF and chrF++ Instead of operating at the word level, the character n -gram F-score (chrF) [273] relies on character n -grams in order to decrease its sensitivity to exact word matches. The F-score is the harmonic mean of the precision and the recall. The general formula is:

$$ngrF = (1 + \beta^2) \frac{(ngrP \times ngrR)}{(\beta^2 \times ngrP + ngrR)}$$

where the factor β represents the ratio of recall importance to precision importance. By default, chrF uses $\beta = 3$, placing more emphasis on recall.

ChrF++ [274] is an evolution of chrF that also takes into account the word n -grams for greater macro-level coherence analysis.

Neural Metrics

Neural metrics for [Machine Translation](#) evaluation use neural machine learning to assess the quality of translations beyond simple lexical matches. Although non-neural techniques rely on word or character overlap, neural metrics can provide a more nuanced and comprehensive evaluation.

BERTScore BERTScore [424] is an evaluation metric based on [BERT](#) that uses contextual embeddings to compute token similarity, using the cosine distance. The BERTScore leverages this representation, which enables it to not rely on exact matches but instead takes advantage of the fact that semantically close words and concepts are positioned closely together in the embedding space.

This is analogous to the Word2Vec models discussed in Section 2.2.3, which are networks trained specifically to produce word embeddings that capture proximity in the meaning of words. By comparing the distance between the latent representation of a reference and the generated sentence, BERTScore can estimate the quality of the translation. The key difference between BERTScore and Word2Vec in machine translation evaluation lies in their embedding representations. As mentioned in Section 2.4.3, [BERT](#) generates dynamic, contextualised embeddings, where the representation of a word depends on its context. In contrast, Word2Vec relies on static embeddings, assigning each word a fixed representation regardless of the surrounding tokens. This distinction makes BERTScore more suitable for translation evaluation, as it captures semantic similarity and contextual nuances, whereas Word2Vec may misinterpret words with multiple meanings.

BLEURT BLEURT [313] is a reference-based learnt evaluation metric based on [BERT](#), similarly to BERTScore 2.5.2, which was pre-trained on synthetic data to help its generalisation faculties. This pre-training involves multiple tasks: a mask filling task, similar to the training in [BERT](#), backtranslation, and predicting dropped words. Although both BERTScore and BLEURT rely on [BERT](#) as the underlying model, BLEURT is reference-free and is explicitly fine-tuned for machine translation evaluation using synthetic and human-annotated data.

COMET [Crosslingual Optimised Metric for Evaluation of Translation \(COMET\)](#) [294] is a neural framework for [Machine Translation](#) evaluation. Pre-trained models

are provided, which vary in their approach and depend on the target language. Although the default COMET22 [295] model employs a reference-based regression approach, CometKiwi [296] does not use any reference to evaluate a Machine Translation-generated sentence. Instead of comparing the machine-generated output to a reference translation, CometKiwi predicts the quality of a translation based solely on the source sentence and the generated translation. CometKiwi achieves this by leveraging a multilingual pre-trained encoder, which is then fine-tuned it on human-annotated quality estimation datasets. The model learns to assign a quality score based on features extracted from the source and translated sentences, capturing semantic adequacy and fluency without relying on a gold-standard reference.

At the time of the publication of the results of the metric shared task of the WMT22 [99], COMET22 is the best published and open evaluation metric, and is second overall, behind MetricX XXL, which has not been made available. COMET is the default model of MATEO [358], an online tool that aims to “improve digital literacy of non-expert users by allowing them to easily evaluate machine-generated translations.”

xCOMET [117] integrates sentence-level evaluation and error-spanning detection capabilities. Using both the source sentence and reference translation enhances the accuracy of the default model. The authors demonstrate that the use of a multilingual embedding space benefits the evaluation process. Although using an ensemble of xCOMET-XL and xCOMET-XXL is the best performing metric on the metric shared task of WMT23 [100], COMET22 was selected as the primary evaluation metric for the work conducted in Chapter 4 of this thesis. This choice was motivated by its faster computational efficiency and its widespread use in academic literature, which enhances the reliability of comparisons.

Direct Scoring Approaches with Large Language Models Decoder-only LLMs are autoregressively trained on vast amounts of data and can subsequently be fine-tuned to enhance their ability to follow instructions and answer questions. GPT Estimation Metric Based Assessment (GEMBA) [179] is a metric that relies on the capabilities of these models without requiring additional training. Through optimised prompt engineering techniques designed to identify the most suitable templates, an LLM can assess translation quality, either with or without a reference. The authors explored seven versions of GPT models and demonstrated that ChatGPT-3.5 achieved state-of-the-art accuracy in the WMT22 Metrics Shared Task.

AutoMQM [95] is a similar metric in that it also relies on a fine-tuned LLM—specifically, PaLM-2 [14]—to evaluate translation quality. However, instead of directly asking the model to score a translation, AutoMQM prompts the LLM to generate a list of errors in the hypothesis and classify each error accordingly. The evaluation process then assigns penalty points based on the severity of each detected error. The authors showed that this approach outperforms directly requesting a score from the LLM when using PaLM models.

2.5.3 Recent Progress in Machine Translation

As in most domains of [Natural Language Processing](#), the Transformer architecture is currently dominant in [Machine Translation](#), usually in its encoder-decoder form. This is not surprising given that the original work on this architecture focused on improving the [BLEU](#) scores on [Machine Translation](#) tasks. Since then, the research has continually progressed, building on the tremendous success of the Transformer architecture.

Multilingual Models

As the name suggests, a multilingual [Machine Translation](#) model is a model that is not limited to translation from one specific language to another. A model of this type can translate different languages into different languages, enabling not only transfer learning, but also considerable savings in language training. In fact, the formula for calculating permutations without possible repetitions is $\frac{n!}{(n-r)!}$, where n is the number of different languages that have to be translated and r is equal to two since a translation is from one language to another. Thus, six different bilingual models would be required to be able to translate English, French, and German interchangeably (EN→FR, EN→DE, FR→DE, FR→EN, DE→EN, DE→FR), 90 models would be required with 10 languages and 9900 for 100 languages.

T5 [288], which stands for Text-To-Text Transfer Transformer, is a versatile Google model that aims to unify all text-to-text tasks in [Natural Language Processing](#), such as text summarisation, text translation, or question answering. T5 relies on varied training on a wide range of different tasks to take advantage of transfer learning. mT5 [405] is a variant of T5 trained on multiple different language pairs and covers 101 languages. However, mT5 cannot be used as is for [Machine Translation](#), as it has to be fine-tuned using parallel datasets.

One of the main multilingual models dedicated to translation is Many-to-Many, or M2M, [94]. This Facebook model is capable of translating more than 100 different languages in different language families and alphabets. It was trained using different parallel datasets along with mining parallel data [298] [356], which is a process in which sentences with similar meanings across different monolingual datasets are extracted to create parallel datasets. Language-Agnostic Sensitivity Representations, or LASER¹, is another Facebook model that is dedicated to creating text embeddings. The calculation of the embedding distance of two sentences in different languages was one of the methods used to extract parallel datasets, resulting in the construction of the CCMatrix dataset [312].

No Language Left Behind, or NLLB [346], is another multilingual [Machine Translation](#) model from Meta dedicated to low-resource languages, which are natural languages where the datasets are non-existent, low quality or scarce. The underlying dataset was constructed by interviewing native speakers.

¹<https://github.com/facebookresearch/LASER>

KG-MT [68] is a multilingual method which relies on a retrieval mechanism to extract relevant information from a multilingual knowledge graph. By incorporating external knowledge and cultural context, this method improves the translation quality, particularly in texts containing entity name.

Large Language Models

Although LLMs are not specifically trained in translation-related tasks, they have demonstrated robust multilingual Machine Translation capabilities [93] [433]. The gigantic number of parameters in these models and the vastness of the datasets used to train them mean that they are capable of translating different languages and even surpassing traditional Neural Machine Translation models for specific language pairs.

It was observed in the General Machine Translation Task of the 2024 WMT [182] that most submissions consisted of fine-tuned LLMs. Notably, the best-performing system overall was a proprietary LLM, Claude 3.5 Sonnet.

Document-Level Machine Translation

Although most of the research on Machine Translation is based on sentence-by-sentence translation, translation of complete documents, or Document-Level Machine Translation, is a rapidly growing field. Taking advantage of the coherence and style of a text at the level of an entire document, models can achieve better performance than trying to separate sentences and translate in isolation [423] [141]. Since large context windows make it possible to read and capture the essence of a larger part of the document than smaller ones, LLMs have also demonstrated excellent performance for DocMT [394] [73].

Additional Advancements

Recent research on enhancing tokenization algorithms [146] has demonstrated that leveraging longer tokens through Word Pair Encoding, combined with a dedicated attention mechanism, enables the extension of traditional transformer models and improves performance in Neural Machine Translation.

A common challenge in Multilingual Neural Machine Translation is the incorporation of additional languages into a model already trained on a set of languages. Recent research [211] suggests that instead of fine-tuning the existing model, which risks catastrophic forgetting and performance degradation in previously learnt languages, it is more effective to identify and prune the least essential parameters for existing tasks. This reduction creates a more compact model, into which new external parameters can then be introduced, specifically dedicated to learning additional languages.

A similar pruning mechanism has also been used [126] to identify in a trained model the optimal sub-network for each language pair. This can be used to extract smaller and models without altering the quality of the generated translations.

We have explored how the Transformer architecture is used in [Machine Translation](#), as well as how machine-generated translations can be automatically evaluated by neural or non-neural algorithms, to enable in-depth comparisons between models. However, in order to better understand Chapter 4, it is essential to understand how machine-generated architectures other than those based on the Transformer architecture, such as [GPT](#), are designed and trained.

2.6 Generative Model Architectures

Generative architectures in [Machine Learning](#) encompass a range of models and algorithms specifically conceived to create synthetic data points that reflect the characteristics of the training dataset. The fundamental concept underpinning these architectures is the ability to discern and replicate the distribution patterns of the training data, thereby facilitating the generation of new data entities.

2.6.1 AutoEncoder

An [AutoEncoder \(AE\)](#) is an encoder-decoder architecture in which the encoded representation of the input data is of a lower dimension. The original idea for [AutoEncoder](#) dates at least from 1987 [311] [418] [24] and possibly from 1986 [305].

The encoder maps the input data to a latent space, that is, a compressed representation, progressively reducing the dimensionality of the input with multiple layers. The latent space captures the most important features of the input data, which the decoder uses to reconstruct the original input data by progressively upsampling the latent vector. The latent space corresponds to the layer labeled “code” in Figure 2.18.

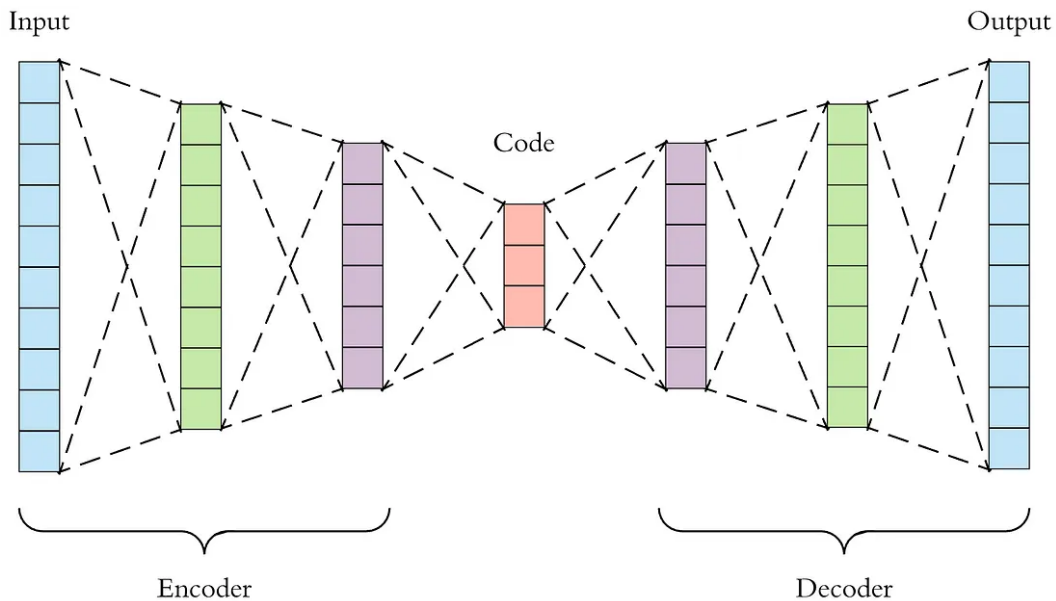


FIGURE 2.18: Architecture of the AutoEncoder model, in which an encoder compresses the input into a latent space and the decoder expands this vector into a vector which has the same dimension as the original input, obtained from Abdollahi et al. [2]

AutoEncoders have been successfully used for anomaly detection [56] [12] [428] [430] (an anomaly is raised if the **AutoEncoder** cannot successfully reconstruct the input data), image denoising [111] [225] [22] (noise is added to the input data and the loss compares the original unaltered input data to the output), and dimensionality reduction [369] [376] [373] [299], which can be used to lower the number of parameters in another model, as instead of directly using the input data, the trained encoder from an **AutoEncoder** can reduce the dimensionality with little loss, lowering the complexity of the subsequent algorithms applied to the relevant data.

The role of the encoder is to compress the input into a lower-dimensional representation, while the decoder is trained to reconstruct the original input from this representation. Training an **AutoEncoder** involves the definition of a loss function such as **Mean Square Error** or **Binary Cross-Entropy Loss**, to measure the dissimilarity between the input data and the reconstructed output.

By comparing the original input with the generated output, the decoder gradually learns how to reconstruct the input distribution from a compressed representation z , where z is the output of the encoder and the input of the decoder, and named “code” in Figure 2.18, while the encoder learns to retain the most important information.

AutoEncoders have been used to generate molecules [160], peptides [75], text [316], images [403], and videos [409]. The process of generating “deep fakes” [227], that is, believable synthetic images in which the face of one person is replaced by another, often a celebrity, is a good example to explain the generative capabilities of **AutoEncoders**. The training phase consists of obtaining a dataset containing images of celebrity A and another dataset containing photos of celebrity B . An encoder E and two decoders, X and Y are then initialised. All images from the two datasets first pass through the encoder E , which maps them to a lower dimensionality distribution Z . Thus, E learns the mapping $E_{A \cup B \rightarrow Z}$.

The decoder X learns to reconstruct the photos of personality A , and is never exposed to photos of B , while decoder Y is only used to reconstruct the images of person B and consequently is never trained on photos of individual A , in order to obtain the mappings $X_{Z \rightarrow A}$ and $Y_{Z \rightarrow B}$.

To graft the face of person B onto a photo of person A , one has to encode an image of A , then give this latent representation to decoder Y , which will transform this vector into an image very similar to the input, but with the face changed. Figure 2.19 presents both the training and the generative process of using **AutoEncoders** for deep fakes.

2.6.2 Variational AutoEncoder

The **Variational AutoEncoder (VAE)** is a type of generative model that combines ideas from traditional **AutoEncoder** and probabilistic graphical models. The aim of **VAEs** is to learn a probabilistic mapping between the input data and a latent space, allowing the generation of new data points. The key innovation of **VAEs** lies in the ability to model

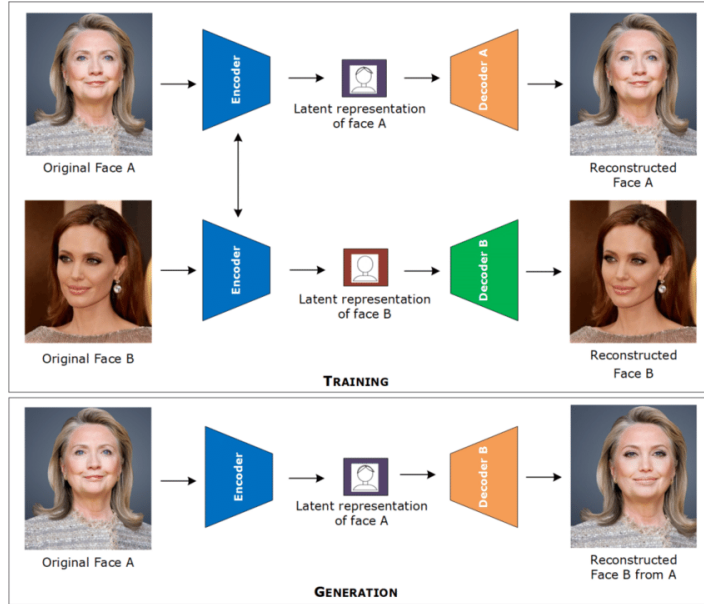


FIGURE 2.19: Deep Fake training process, in which two autoencoders sharing the same encoder are trained to reconstruct images of two different people, obtained from [227]

probability distributions over the latent space, enabling them to generate diverse and realistic samples.

The encoder of a VAE, as distinct from a traditional [AutoEncoder](#), produces not just a deterministic representation z , but also the parameters of a multivariate Gaussian distribution representing the latent space. The parameters obtained by the encoders are μ and σ^2 , respectively, the mean and variance vectors. The decoder, in turn, takes a sampled point from the latent space and tries to reconstruct the input based on that sample point. This difference is illustrated by Figure 2.20, where (a) is a VAE while (b) is an [AutoEncoder](#).

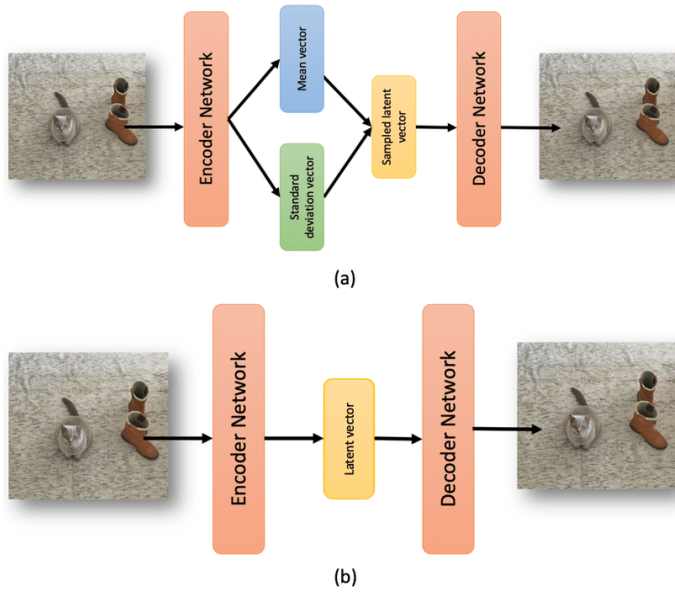


FIGURE 2.20: Difference between the inner workings of a VAE, in which the latent vector is sampled from a mean and a variance, and an [AutoEncoder](#) in which the latent vector is directly generated by the encoder, obtained from [105]

This architecture forces the encoder to model the distribution of the training dataset as a Gaussian distribution, and as such, it allows drawing samples from that distribution. However, directly sampling from this distribution using the $\mathcal{N}(\mu, \sigma^2)$ distribution introduces randomness, which is incompatible with the backpropagation process, as it is a non-differentiable operation. In fact, randomness means that there is no well-defined mathematical relationship (gradient) between the parameters (μ and σ) and the sampled value z . The “reparameterisation trick” consists in generating an auxiliary variable ϵ from the $\mathcal{N}(0, \mathcal{I})$ distribution. Then, z can be obtained as $z = \mu + \sigma \odot \epsilon$, where $\odot$ is the element-wise product.

Generating a sample using a trained VAE simply involves generating a latent vector from the learnt parameters μ and σ^2 . This latent representation is then fed to the decoder, which reconstructs this input into a new data point that is similar to the dataset on which it was trained.

2.6.3 Generative Adversarial Network

At its core, a Generative Adversarial Network (GAN) [112] is a deep learning framework consisting of two neural networks that compete in a zero-sum game, that is, a mathematical situation where the gain of one network is exactly balanced by the loss of another network.

These two networks, the Generator and the Discriminator, have two very distinct functions. The Generator G produces realistic synthetic data, while the role of the Discriminator D is to determine whether its input has been generated by G , i.e., a *fake*, or is *real* data. The goal of this framework is to generate synthetic data that is indistinguishable from the dataset, and GANs are usually trained to produce outputs that resemble known distributions such as images, sounds, or texts. In more mathematical terms, the role of the Generator is to map the data from one distribution X to another distribution Y , such that $\forall x \in X, G_{X \rightarrow Y}(x) \in Y$.

GANs have been used in anomaly detection [32] [76] [396], super resolution [29] [261] [375], video prediction [157] [193] and image-to-image translation [11] [201].

Training Process

The GAN training proceeds in alternating periods, where the Generator network will train on a certain number of examples while the Discriminator is kept constant, before swapping the network being trained. Depending on the specific case, the period can go from a single example to multiple epochs.

Figure 2.21 offers a schematic view of the inner workings of the training process of GANs.

The traditional GAN does not incorporate a mechanism to influence the synthetic output, since the data are generated from a random noise vector. Rather than only relying

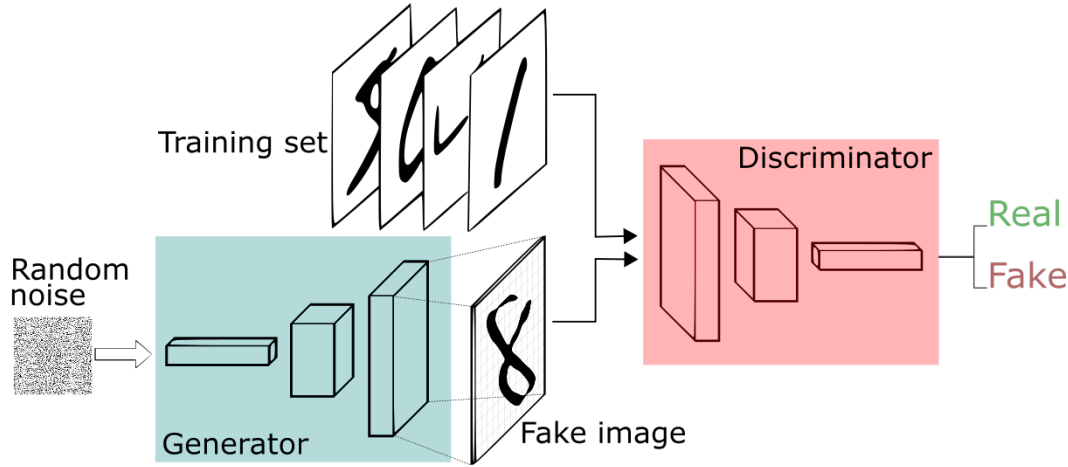


FIGURE 2.21: Representation of the training process of a GAN model, in which the generator generates data from random noise while the discriminator learns to distinguish original data from synthetic data, obtained from [325]

on this noise vector, Conditional GANs [236] [200] [401] [328] incorporate additional variables to direct the generation process, such as labels, tags, or descriptions.

Challenges

Training GANs is a notoriously difficult process [113] [54] [421], for a variety of reasons.

Mode Collapse The generator only produces a limited range of outputs rather than generating the entire data distribution. This has been greatly improved by using the Wasserstein distance [15] [118], which is a measure of the difference between two probability distributions, used to quantify the cost of optimally transporting the mass from one distribution to another.

Convergence The generator and discriminator can continue oscillating forever and never reach an appropriate equilibrium. The Discriminator feedback gets less meaningful over time as the Generator improves, since it will end up only being correct half the time;

Computationally Expensive Since a GAN requires two distinct neural networks for its training, but only one is used during inference, this framework is costly;

Training Instability If one network starts to outperform the other too quickly, further performance improvements will stagnate.

2.6.4 CycleGAN

A CycleGAN [432] is a type of GAN that aims to learn the mapping between the two datasets' distribution without requiring any paired example, that is, two data points that

represent the same content but do not share the same domain. Translating a sentence from one language to another and then back should result in the original sentence, which is a property named “cycle consistency”. CycleGANs have been used for different image-to-image translation tasks, such as object transformation [432], face generation [219], image dehazing [92], style transfer [432] and even steganography [61].

The first and most famous example of image-to-image translation is shown in Figure 2.22. By training a CycleGAN on a dataset consisting only of pictures of horses on one side and pictures of zebras on the other, the network is then able to “horsify” a zebra, that is, transform a zebra in a picture into a horse with minimal changes to the rest of the image.

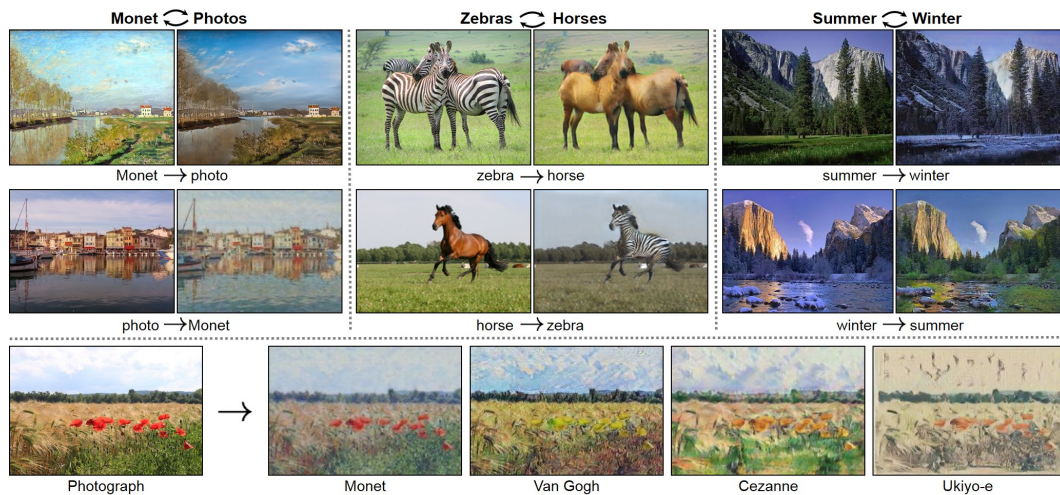


FIGURE 2.22: Images generated using a trained CycleGAN, obtained from [432]

Although a GAN already contains two different neural networks, a CycleGAN doubles this number, as it contains two discriminators and two generators. The two datasets are usually symmetrical, i.e., the data type is the same between the datasets, but the datasets need to have different distributions so that the CycleGAN can learn the particular features and similarities of each of these distributions.

The role of the Discriminator essentially stays the same, namely distinguishing a real example from a fake one in a specific domain. However, the Generator differs very significantly. Because it has to map one distribution to another, its input is henceforth the same dimension and type as its output and not a random vector, as is the case in a traditional GAN.

Training Process

As the CycleGAN training process is rather complex, definitions are crucial. Let X and Y be the two distributions that should be cross-mapped by the two generators G and F , G mapping domain X to domain Y while F maps domain Y to domain X . x is a training example from distribution X and y is a training example from distribution Y .

D is the discriminator that learns to distinguish between what belongs to domain X and what does not, while E is the discriminator that categorises between domain Y and everything else.

$\hat{x}$ is the output of generator F given the input y and $\hat{y}$ is the output of the generator G given the input x . $\hat{\hat{x}}$ comes from F given $\hat{y}$ and $\hat{\hat{y}}$ from G given $\hat{x}$.

Finally, id_x is the output of generator F given the input x and id_y is the output of G from y .

In mathematical terms:

- $x \in X$;
- $y \in Y$;
- $G_{X \mapsto Y}(x) = \hat{y}$, with $\hat{y} \in Y$;
- $F_{Y \mapsto X}(y) = \hat{x}$, with $\hat{x} \in X$;
- $G_{X \mapsto Y}(\hat{x}) = \hat{\hat{y}}$, with $\hat{\hat{y}} \in Y$;
- $F_{Y \mapsto X}(\hat{y}) = \hat{\hat{x}}$, with $\hat{\hat{x}} \in X$;
- $G_{X \mapsto Y}(y) = id\_y$, with $id\_y \in Y$;
- $F_{Y \mapsto X}(x) = id\_x$, with $id\_x \in X$;
- $x = \hat{\hat{x}}$;
- $y = \hat{\hat{y}}$.

Figure 2.23 only shows the training of Generators and not Discriminators, which indicates the complexity of the entire training process.

During training, two loss functions cooperate: the [Cycle-Consistency Loss \(CCL\)](#) and the [Identity Loss \(IL\)](#).

Cycle Consistency Loss The objective of a CycleGAN is to obtain a mapping such that:

$$G_{X \mapsto Y}(F_{Y \mapsto X}(y)) = y \text{ and } F_{Y \mapsto X}(G_{X \mapsto Y}(x)) = x$$

The [Cycle-Consistency Loss](#) aims to enforce this mapping by comparing the initial input x to the generated output $\hat{\hat{x}}$ and by comparing initial input y to the generated output $\hat{\hat{y}}$. This loss function prevents the model from producing incorrect translations, penalising the network when the original input and generated output differ.

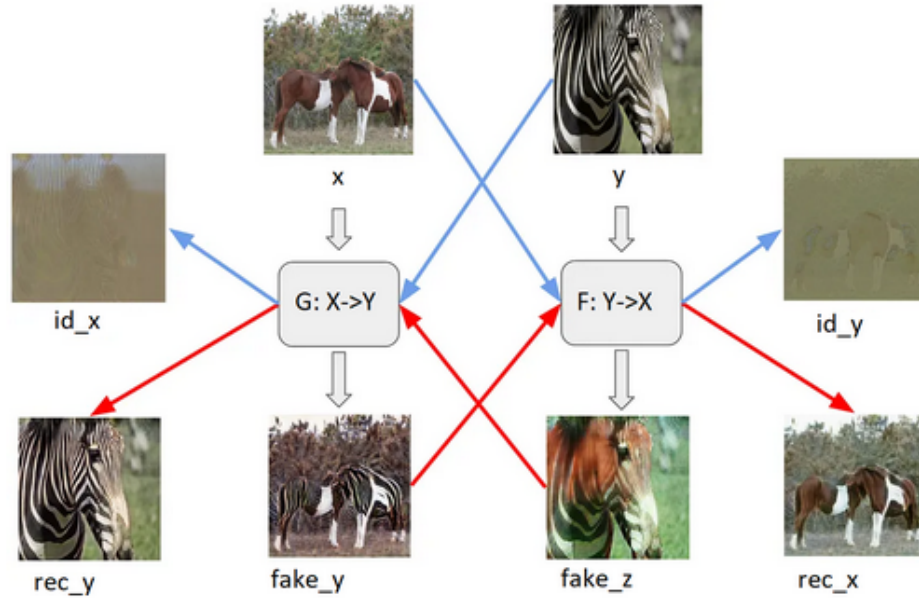


FIGURE 2.23: CycleGAN training process, in which two GANs cooperate to generate credible synthetic data from synthetic and original data points, inspired from [392]. x and y are original images from the dataset, $fake\_x$ is obtained from $F_{Y \rightarrow X}(y)$, $fake\_y$ is obtained from $G_{X \rightarrow Y}(x)$, rec_x and rec_y are respectively obtained from $F_{Y \rightarrow X}(fake\_y)$ and $G_{X \rightarrow Y}(fake\_x)$ and id_x and id_y are respectively obtained from $F_{Y \rightarrow X}(x)$ and $G_{X \rightarrow Y}(y)$

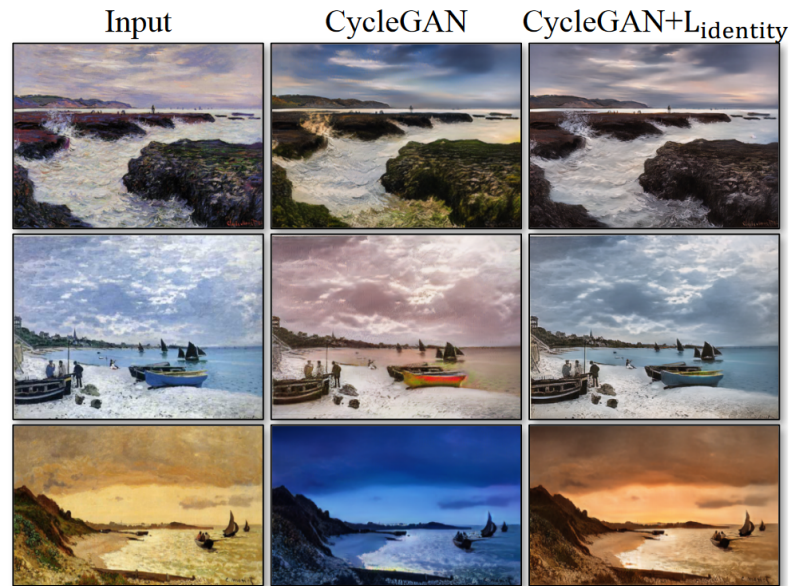


FIGURE 2.24: Demonstration of the importance of the Identity Loss, obtained from [432]

Identity Loss When a Generator receives data from its target domain, it should not apply any changes to its input, as the data already belong to the right distribution. Thus, the objective of the Identity Loss is to enforce $G_{X \rightarrow Y}(y) = y$ and $F_{Y \rightarrow X}(x) = x$. Figure 2.24 shows the impact of adding this loss function.

We explored the way in which models were able to generate synthetic data, whether by compressing and then reconstructing the data, such as AutoEncoder, or by using

competing models, one attempting to generate credible synthetic data, the other discriminating between data belonging to the dataset and synthetically generated data. These models also have other uses than data generation, such as style transfer.

2.7 Style Transfer

Style Transfer refers to the set of techniques used to alter the aesthetic or thematic presentation of data while preserving its underlying information or meaning. The essence of Style Transfer lies in distinguishing between the “style” and the “content” of the data, which can be quite nuanced and varies with the type of data in question. Understanding Style Transfer requires a thorough understanding of these concepts [159]. Figure 2.25 shows a proposed neural style transfer taxonomy.

2.7.1 Image

In image processing, the style transfer process involves the integration of the stylistic attributes exhibited by a reference image, typically a painting or specific artistic style, onto a target image while simultaneously retaining the inherent content of the target image. As previous work defines it, “this process modifies the texture characteristics of the content image without altering its content, based on the texture information of the style image” [42].

The content encapsulates the high-level information and structural characteristics of the target image, encompassing objects, shapes, and their arrangement. In contrast, the style captures the artistic elements and textures from a reference image, including colours, strokes, and patterns. Typically, Style Transfer involves the use of two main components: a content loss process and a style loss process. Figure 2.26, shows the Style Transfer categories in image processing.

Pretrained Convolutional Neural Network

The first approach to Neural Style Transfer for images [106] was performed on a VGGnet [326], pre-trained on a comprehensive dataset for image recognition tasks. This is a popular approach that was replicated [326], using other architectures such as ResNET [130] [10] [42] [220].

Deep layers, referring to hidden neurons, that capture and learn complex and abstract representations of input data, are used as a feature extractor, extracting both the content and style information from the input images. Using these features, a loss function quantifies the disparities between the generated image and both the target image (with respect to content) and the reference image (regarding style). CNNs have demonstrated their ability to extract meaningful features from images [161]. Through iterative optimisation, the algorithm progressively modifies the generated image to increasingly align its content and style features with the intended result by minimising the defined loss

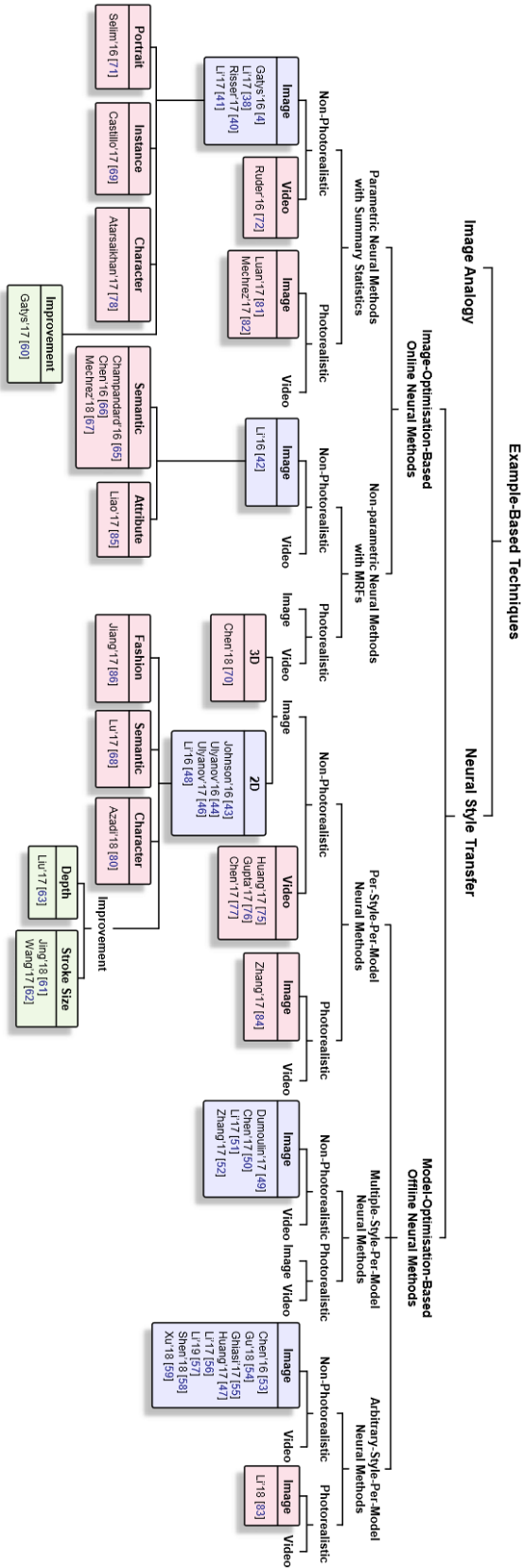


FIGURE 2.25: Taxonomy of Style Transfer techniques, obtained from [161]. The taxonomy divides Neural Style Transfer methods into Image-Optimisation-Based Online Neural Methods and Model-Optimisation-Based Offline Neural Methods, further classifying them into parametric and non-parametric techniques, per-style and multiple-style models, and photorealistic and non-photorealistic applications

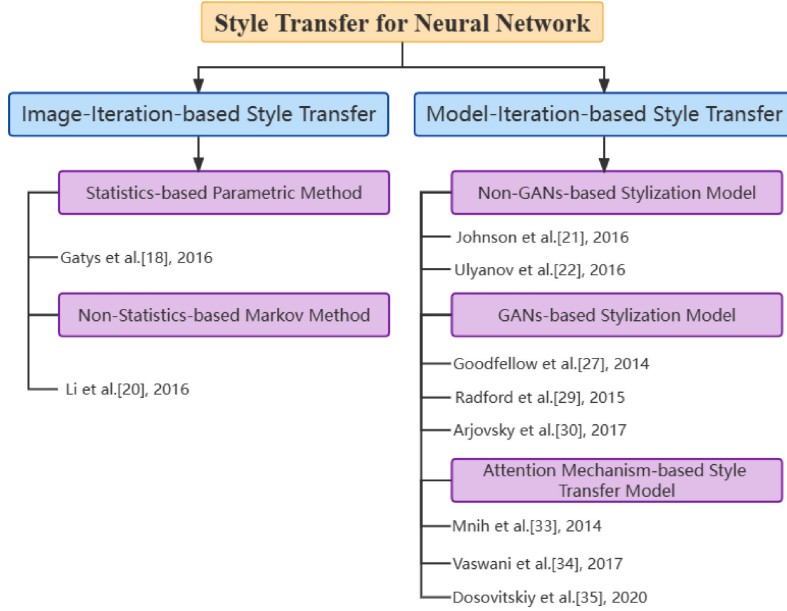


FIGURE 2.26: Style Transfer categories in image processing, obtained from [42]

function. By optimising these two loss functions, the algorithm adjusts the target image to combine the content of one image with the style of another, ultimately producing a new image that combines the best of both worlds. This technique is widely applied in artistic rendering, photo enhancement, and creative image synthesis, allowing for the transformation of ordinary images into visually compelling representations inspired by well-known artistic styles.



FIGURE 2.27: Artistic styles applied to the same photography. The facades are the content of the picture, and the style is changed to transform a photographic image into a painting in the style of Van Gogh for example, obtained from [107]

Figure 2.27 illustrates the difference between style and content. The image A is used as content, while the paintings (shown as thumbnail images) are used to apply their style. Images B, C and D show how applying a style to content produces a result that takes on the style's genre and characteristics while applying it to the content's shapes.

CycleGAN

Rather than relying on a pre-trained network, Style Transfer techniques using the CycleGAN framework, discussed in Section 2.6.4, have emerged [101] [351] [28] [268] [26] [51] [371]. CycleGANs do not require a specific style image and a content image, but are rather trained on an unpaired dataset of two different distributions and can then migrate the style of an image from one distribution to the other, preserving the content.

One of the key advantages of using a CycleGAN for Style Transfer-related tasks is the bidirectionality provided by the framework, that is, the ability of transferring the style of an image from one distribution to another and vice versa. Unlike pre-trained convolutional networks, where it is essential to manually choose the number of pre-trained layers to serve as the backbone, no similar work is involved here. The networks are generally trained from scratch and in parallel.

Pre-trained CNNs often preserve content structure better since they explicitly match high-level content features between images. They also benefit from feature extractors that have already been trained on large datasets, making them faster and more stable. On the other hand, CycleGANs does not directly use a style loss but instead learns mappings between domains, which may lead to more flexible but less consistent style adaptation and they require training from random initialisation, often needing significantly more computational resources and training data to reach comparable quality.

2.7.2 Audio

In the realm of sound and music, style transfer techniques have emerged to manipulate audio data. Although it is a relatively newer area compared to image style transfer [361], the principles are similar. Researchers have explored deep learning models like CNNs [52] [151], RNNs [300] [122] and GANs [171] to capture the musical essence of a specific artist or genre and apply it to other audio files. These models learn the specific patterns, rhythms, and tones of the target style and then recreate them in new audio compositions. In a similar manner to image style transfer, a pre-trained network extracts features from an audio file that will be used as a style. An iterative process is then used to apply these features to a content source until the result is deemed close enough to the target style.

The heart of style transfer lies in the recombination phase. As illustrated in Figure 2.28, the model combines the content features of the first audio with the style features of the second. This is a complex task, as the model must ensure that the content remains recognisable while seamlessly integrating the new style.

During the process, multiple challenges must be addressed, such as maintaining the naturalness of the audio, ensuring temporal coherence, and dealing with variable audio lengths and structures. Techniques such as feature disentanglement [185] are often used to enhance the process.

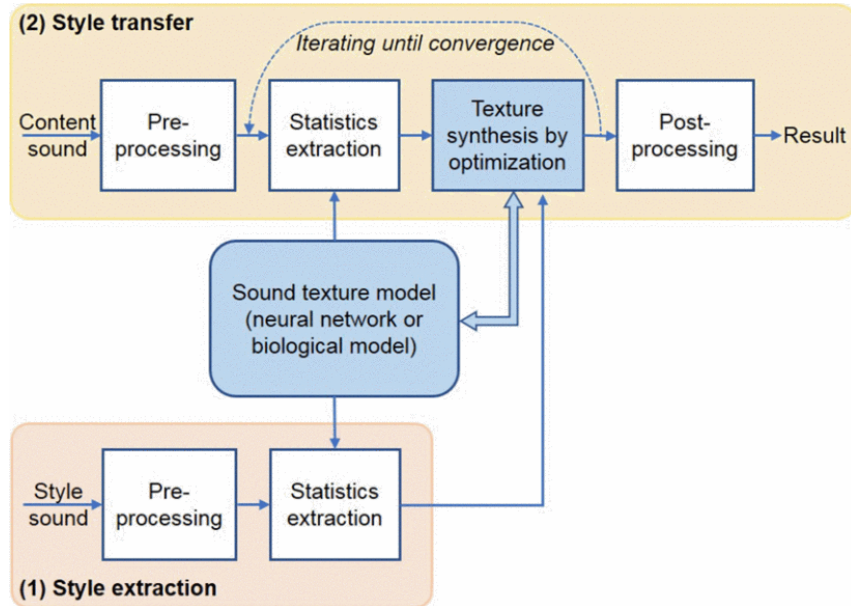


FIGURE 2.28: Audio Style Transfer architecture, obtained from [116]

2.7.3 Text

The style transfer applied to text aims to modify the stylistic attributes, such as sentiment, of the text while retaining the content that is unaffected by style within its context. For the sake of clarity, different citations from previous work are added: “text style transfer is defined as a task of adapting and/or changing the stylistic manner in which a sentence is written, while preserving the meaning of the original sentence” [352], “The goal of TST is to automatically control the style attributes of text while preserving the content” [159], “Style is an intuitive notion that involves the manner in which something is said” [229], “Style is an integral component of a sentence indicated by the choice of words a person makes” [352], “A crucial component of deep style transfer models is the part that generates the output sentence based on the representation of the content of the original sentence and the corresponding styles.” [352].

A recent review of the literature on text style transfer [352] lists the different text styles and identifies six of them. The examples given are also obtained from this work.

Personal Style

The way an individual expresses itself reveals information about him. In fact, there are differences in expression depending on an individual’s age, gender, or cultural background or level of education, for example, in the tendency to use emojis, the vocabulary used, or the way they construct sentences.

- My lord, the queen would speak with you, and presently. (shakespearean english)
- My lord, the queen wants to speak with you right away. (contemporary english)

Formality

In order to give a precise definition of formality, it is generally associated with the use of the passive voice. It is precise, logically rigorous, with words drawn from a subject-specific vocabulary, and often contains longer sentences or paragraphs. A typical example of formal writing is scientific publications, while informal language is less constrained with more non-standard vocabulary. The use of style transfer techniques could help a writer achieve the desired level of formality.

- Gotta see both sides of the story. (informal)
- You have to consider both sides of the story. (formal)

Politeness

Politeness is a spectrum, a high level of which enables us to maintain a “positive face” during social interactions, while a low level may be linked to a significant level of frustration.

- Send me the data. (non-polite)
- Could you please send me the data? (polite)

Offensiveness

In [Natural Language Processing](#), hate speech detection is an area that is being studied, particularly aimed at mitigating offensive content on social media platforms. Style transfer techniques could be employed to automatically rewrite offensive comments and posts, transforming them into more neutral or non-offensive versions while preserving their original intent.

- I hope they pay out the ***, fraudulent or no. (offensive)
- I hope they pay out the state, fraudulent or no. (non-offensive)

Genre

The genre of the text is determined by its purpose and its target audience. News organisations and advertisements do not use the same tone or words because they do not have the same objectives. The transfer style applied to the genre could, for example, enable non-experts to more easily understand a report written for experts.

- Many cause dyspnea, pleuritic chest pain, or both. (expert)
- The most common symptoms, regardless of the type of fluid in the pleural space or its cause, are shortness of breath and chest pain. (layman)

Sentiment

Detecting emotions associated with language makes it possible, for example, to predict public opinion on a given topic, political sentiment, and even sales.

- Great food, but horrible staff and very very rude workers! (negative)
- Great food, awesome staff, very personable and very efficient atmosphere! (positive)

Disentanglement

Disentanglement in text style transfer is the process of separating the content and the style. Depending on how models approach disentanglement, the process can be explicit or implicit. While disentanglement is an effective approach in text style transfer, some methods do not require explicit disentanglement. Figure 2.29 shows a taxonomy of text style transfer methods.

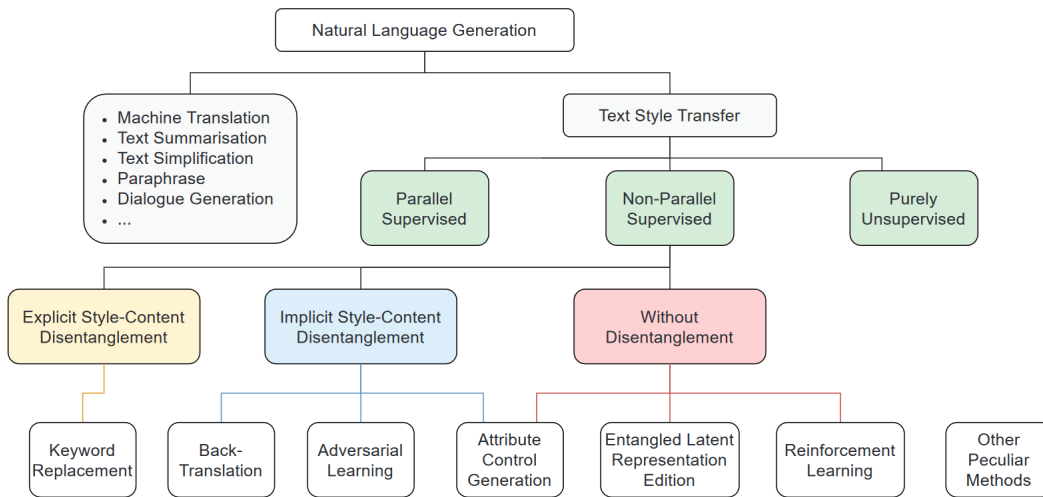


FIGURE 2.29: Taxonomy of text style transfer methods, obtained from [142]. This taxonomy categorises text style transfer techniques into parallel supervised, non-parallel supervised, and purely unsupervised methods. It further subdivides non-parallel supervised techniques into explicit and implicit style-content disentanglement methods, as well as those without disentanglement, highlighting strategies such as adversarial learning, attribute control generation, reinforcement learning, and entangled latent representation editing

Explicit Disentanglement Explicit disentanglement is achieved through an intentional configuration of model architecture, loss function, and training methodology to isolate content and style into separate representations. A common approach [142] would be to label specific words associated with style, such as “bad” and “good” in the context of sentiment transfer, in order to specifically target and replace them [196] [341] [399].

Implicit Disentanglement On the other hand, implicit disentanglement does not require any explicit mechanism to learn disentanglement. Although the training does

not involve a specific disentanglement process, the model will learn how to differentiate style and content as a by-product of the entire training process. Adversarial learning, discussed in Section 2.7.3, is based on such disentanglement [322] [217] [191].

Without Disentanglement Recent methods [142] have demonstrated that disentanglement is not necessary for text style transfer and that techniques such as adversarial learning and reinforcement learning [339] [74] can be used to transfer the style of a sentence without ever learning how to separate the style and the content.

Datasets

Within the domain of textual style transfer, two predominant categories emerge. The first category pertains to the modulation of a text’s emotional valence, such as altering a text’s sentiment from positive to negative, or vice-versa. The second category focusses on the conversion of textual expressions between informal and formal registers.

Yelp The Yelp dataset [413] contains restaurant reviews from users of the Yelp platform. Each review includes text, the review itself, and a rating from 1 to 5. Also included in this dataset for each review, for example, are the number of people who found the review useful, the date on which the review was posted, the author ID and the ID of the restaurant concerned, but these data are not used in the transfer style. The dataset has both “small” and a “big” versions.

Having a simple rating between 1 and 5 for each restaurant, accompanied by a sentence or set of sentences, is an effective way of creating a non-parallel dataset consisting of positive and negative reviews. The Yelp dataset has been used many times in text style transfer tasks [322] [197] [258] [210] [350] [66].

GYAFC Yahoo! Answers L6 corpus [407], itself based on the Yahoo! Answers [406], includes all the questions and their corresponding answers that were posted on the Yahoo! Answers website. [Grammarly’s Yahoo Answers Formality Corpus](#) [291] is a subset of the L6 dataset that was thoroughly curated to ensure its quality. It contains more than 100,000 paired examples of formal/informal sentences. Figure 2.30 shows examples that belong to this dataset.

Informal: *I’d say it is punk though.*

Formal: *However, I do believe it to be punk.*

Informal: *Gotta see both sides of the story.*

Formal: *You have to consider both sides of the story.*

FIGURE 2.30: Examples of sentences in the [GYAFC](#) dataset, which is a paired formal-informal text dataset, obtained from [64]

XFORMAL XFORMAL [36] evaluates multilingual formal style transfer, focussing on reformulating informal texts into a more formal style in the Brazilian Portuguese, French and Italian languages. The main new feature of this dataset is that it focusses on languages other than English.

Evaluation

As discussed in Section 2.7.3, text style transfer tasks involve the transformation of a sentence from a source distribution into a target distribution. Models can, for example, move a sentence from a distribution of negative feelings to positive feelings, or from a discourse in formal language to an informal discourse. In order to evaluate this task, it generally involves three different measures: measuring the preservation of the content, measuring the intensity of the style transfer, and measuring the fluency [235].

Content Preservation Preserving content in text style transfer means retaining the semantic meaning and intent of the source sentence, even if this requires paraphrasing or substituting words to achieve the target style.

- Most published work relies on the [BLEU](#) score to assess content preservation [258] [197] [210] [350] [102]. If we were to focus on preserving meaning, it is likely that evaluations based on neural networks would perform better, since they can “understand” the semantic proximity of sentences that have no words in common. However, using such measurement algorithms would in no way encourage models to preserve content. In addition to enabling the model to preserve content, the use of the [BLEU](#) score also limits the collapse mode, which would be a form of “cheating”. This score punishes the network that converts any sentence in the source domain to a single sentence in the target domain. It is therefore through methods based on the overlap between the source and target sentence that we can best determine the extent to which a model manages to retain the original words during its generation;
- When access to a parallel dataset is available, the generated sentence is compared to the reference. However, in the absence of a parallel dataset, the generated translation is contrasted with the source, leading to the utilisation of the [source Bilingual Evaluation Understudy](#) score [143], as opposed to the traditional [BLEU](#) score;
- As discussed in Section 2.5.2, the cosine similarity between the embedding representation of the source and the generated sentences can measure how semantically similar the two sentences are;
- [Word Overlap \(WO\)](#) [162] counts the unigram [Word Overlap](#) rate of the source and generated sentence, while excluding stop words and style-attributed words

Fluency The Kneser-Ney trigram language model [177] is a common approach to measuring the fluency of a sentence [143]. The Kneser-Ney language model, after being pre-trained, gauges the frequency of trigrams (a specific case of word n -grams where $n = 3$) within a training corpus. The perplexity score (PPL) for a new sentence is then determined by comparing its trigrams against the predictions of this model. A lower perplexity score indicates better alignment with the corpus, suggesting greater linguistic fluency in the generated sentence. Fluency is also known as *naturalness* [235].

Fine-tuned LLMs can also be used to evaluate the fluency in text style transfer evaluation [257]. Zero-shot prompting often outperforms traditional automated metrics in term of fluency, and prove that LLMs can also be used as content preservation and transfer strength metrics.

Transfer Strength Assessing a model solely based on its ability to conserve content could yield an unexpected outcome: the system does not undergo transformation and merely echoes the input sentence verbatim, since the congruence with the original sentence is inherently perfect.

Similarly to how a discriminator within a GAN verifies the transition of input from one distribution to another (as discussed in Section 2.6.3), the style assessment forces the network to alter the input, assuming that it does not already align with the desired style.

For this purpose, a pre-trained classifier such as fastText [165] or TextCNN [174] is used, a method also called “accuracy” (ACC). This facilitates the verification that the synthesised sentence indeed belongs to the intended domain. The classifier is employed to quantify the number of sentences whose style has been successfully altered by the model. Prior to style transfer, the classifier should assign all sentences in the dataset to their original category. The goal is that, following the style transfer process, the highest possible number of sentences are reclassified into the target category. The Earth Mover’s distance measures the “cost” involved in moving from the generated distribution to the target distribution [235].

G-Score The aim of the G-Score [143] is to provide a simple metric that measures and encompasses the different aspects of the evaluation of text style transfer. The G-Score is the geometric mean of the accuracy, the sBLEU, the WO and the inverse of the perplexity score:

$$G-Score = \sqrt[4]{ACC \times sBLEU \times WO \times \frac{1}{PPL}}$$

H-Score The H-Score [143] on the other hand is the harmonic mean of these values:

$$H-Score = \frac{4}{\frac{1}{ACC} + \frac{1}{sBLEU} + \frac{1}{WO} + PPL}$$

Adversarial Learning

The [GAN](#) framework has demonstrated capabilities in text generation [414] [426] [119] [53] and text style transfer [191] [412].

The CycleGAN framework has already been used for text style transfer tasks. TextCycleGAN [218] is an encoder-decoder architecture based on LSTM layers (discussed in Section 2.2.4) which achieves excellent [BLEU](#) scores at the cost of a reduced accuracy, which is the evaluation of the target sentiment.

Another work relying on a CycleGAN [278] uses an encoder-decoder architecture, where the encoder encodes the input sentence into two latent spaces: one for the style and the other for the content. The style latent space is a 4-dimensional one-hot encoded vector, since four represents the number of styles used and is trained using a supervised manner, as the label is known during training. The content latent space is trained by the reconstruction of the input. The two latent spaces are then concatenated and used as input to the decoder. By cleverly adding a supervised loss to the CycleGAN framework, the model learns to transfer the style.

Other architectures such as [Variational AutoEncoders](#) have been used in Text Generation [35] [144] [318] and text style transfer [209] [202] [374] [398] [402].

We explored how it was possible to transfer the style of one document to another while preserving the content of the latter. We were mainly interested in style transfer applied to text, in particular the most commonly used datasets and the evaluation of style transfer. To close this chapter, we examine model quantisation, that is, the way in which model weights can be stored and “shortened”.

2.8 Quantisation

The foundation of [Machine Learning](#) consists in the multiplications and additions of weights (coupled with a non-linearity for “decision-making”). These weights, or parameters, each occupy a certain amount of computer memory, and thus limitations of hardware constrain the number of weights that can be loaded at the same time. While it is possible to scale the quantity of memory both vertically, by adding more memory to the compute nodes, and horizontally, by deploying more compute nodes, another method to increase the number of parameters is to fit more weights in the same memory space. This approach is facilitated by quantisation, and it focusses not only on memory space but also on training and inference speed. Its main objective is to trade some weight “quality” for more “quantity”.

2.8.1 Precision

The set of all decimal numbers $\mathbb{R}$ is infinite, as it encompasses $\mathbb{Z}$, the infinite set of all integers and because there are infinitely many decimal fractions between two integers.

Moreover, some decimal numbers are irrational, that is, numbers with an infinite decimal expansion that does not repeat, such as π [245]. By definition, an infinite set of numbers, some being themselves infinite, cannot be represented on a finite storage, that is to say, with a finite number of bits.

Precision indicates the precision and exactness of the representation of a decimal value in computing. Standards such as the IEEE 754 standard [250] have emerged to provide methods for representing and approximating numbers on predetermined numbers of bits.

Integer Numbers

In computing, multiple integer types exist depending on two factors: whether the type is signed or unsigned, and the number of bits to use. As the names suggest, *int8*, *int16*, *int32* and *int64* are respectively represented in 8, 16, 32, or 64 bits. As signed integers, they can represent both negative and positive values. While an integer *int8* can represent any integer in the range $[-128, 127]$, an integer *int64* has a range $[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]$. The general formula to obtain the range of a signed integer type using the number of bits n is $[-2^{n-1}, 2^{n-1} - 1]$.

Unsigned integer types can only represent a subset of the set of natural (i.e. positive) numbers $\mathbb{N}$. An unsigned integer *uint8* can represent any integer between 0 and 255. The range of integers that can be represented using n bits is $[0, 2^n - 1]$.

Although the term “precision” is not typically associated with integer numbers, since an integer is either in range and can thus be exactly represented, or out of range and thus cannot be represented, they are included in this section for clarity.

Fixed-Point Representation

The fixed-point representation is the most obvious method for storing a decimal number using a specified number of bits. The Fixed-Point name comes from the fact that the number of bits for the integer part is fixed, and the number of bits in the fractional part is also fixed. Using the $Qm.n$ format, with m bits dedicated to the integer part and n bits for the fractional part, n determines the precision. The resolution for the signed $Qm.n$ format is given by the formula 2^{-n} and the range by $[-2^{m-1}, 2^{m-1} - 2^{-n}]$. For example, the Q14.2 format number has a resolution of 0.25 and can represent values between $[-8192.0, 8191.75]$ in increments of 0.25.

Thus, while Q14.2 can exactly represent the value 13.75, it cannot exactly store the number 13.84, resulting in a loss of precision as the number gets rounded to the closest representable value.

Floating-Point Representation

The floating-point representation is another method of storing decimal numbers, which allows for wider ranges compared to Fixed-Point numbers using the same number of bits.

Multiple floating-point variants exist, depending on the number of bits used. The most common variants are double-precision, which requires 64 bits; single-precision, which uses 32 bits; and half-precision, which relies on only 16 bits. Although these notations have, respectively, been renamed binary64, binary32 and binary16 in the 2008 revision of the IEEE standard [153], in the present document we use the older float64, float32 and float16 terms interchangeably. Floating-points represent numbers using the triplet (s, e, m) where s denotes the sign, e the exponent and m the mantissa.

The value of a float32 number is $(-1)^s \times m \times 2^{e-127}$, where s requires 1 bit, e requires 8 bits and m requires 23 bits. The precision indicates how close the representation of a number is to its actual value, and is 24 in the case of a float32, since the first bit of the mantissa is implicitly 1 in normalised numbers.

Half-precision numbers use 1 bit for the sign, 5 for the exponent, and 10 bits for the fractional part. The precision is only 11, less than float32, and the range of representable values is narrower, but the storage requirements is halved. Figure 2.31 displays the different bit allocations of the IEEE 754 standards depending on the number of bits.

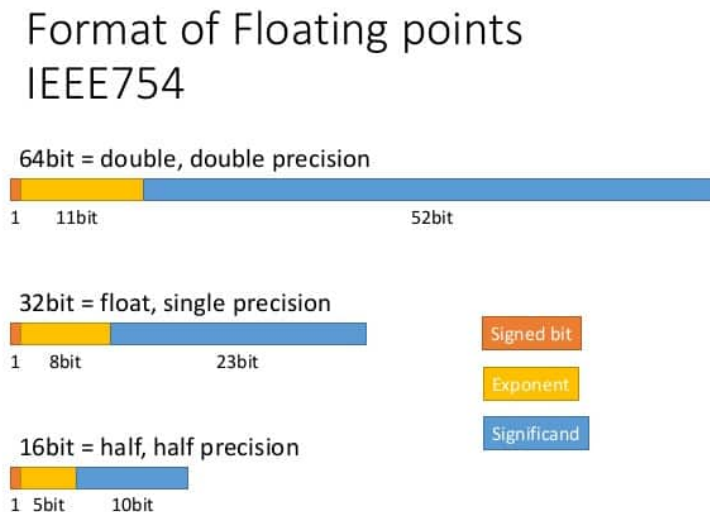


FIGURE 2.31: Format and bit attribution of floating-point IEEE 754, obtained from [121]

Bfloat and TF32

Bfloat16, or Brain float16, is a variation of binary16 dedicated to [Machine Learning](#). While binary16 allocates 5 bits to the exponent and 10 to the fraction, bfloat16 has an 8-bit exponent and a 7-bit mantissa. The immediate consequence of this choice is that

since bfloat16 allocates the same number of bits for the exponent as float32, their ranges are similar. However, it is clear that bfloat16 is not able to represent as many numbers as binary32 and thus fewer exact decimals can be represented. This representation has demonstrated excellent performance and some models trained using bfloat16 obtain the exact same performance as their float32 counterparts, while using less memory and training faster [167].

NVidia has developed its own floating point representation named TensorFloat32, or TF32 [337]. Despite what its name suggests, it uses 19 bits. 1 is dedicated to the sign, 8 to the exponent, similarly as float32 and bfloat16 and 10 bits are allocated to the mantissa, similarly to float16. TF32 is technically not a type, since it is never exposed to the user, and is only an operational mode, meaning that float32 parameters can be silently converted to TF32 before matrix multiplications in order to accelerate the computations while retaining most if not all of the required precision. Figure 2.32 shows a consistent 2-6 times speedup using TF32 compared to float32, although it should be noted that they did not keep the hardware constant in their comparison.

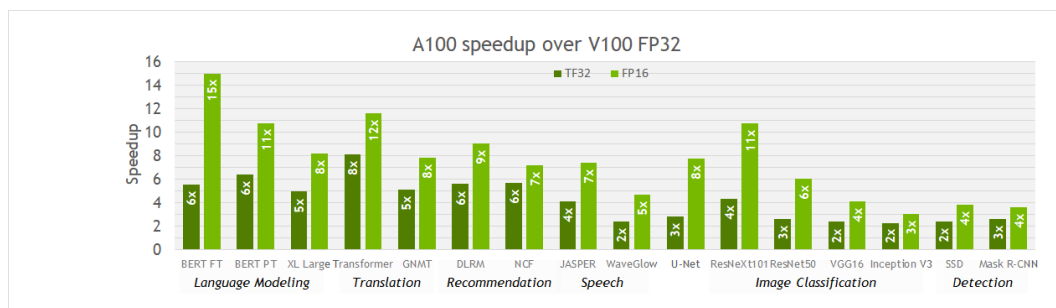


FIGURE 2.32: Performance improvements achieved by using TF32 and float16 precision on an NVidia A100, in comparison to float32 precision on an NVidia V100, obtained from [337]

2.8.2 Computing Speed

By requiring fewer bits to represent a number, float16 reduces memory usage compared to float32 or float64. Consequently, this allows larger models with more parameters to be accommodated using less memory [242] than would be required using float32. A smaller memory footprint also increases the throughput of computations by reducing the size of data transfers.

In recent years, manufacturers have included dedicated hardware to speed up float16 computation, which can provide order-of-magnitude performance improvements for machine learning workloads compared to traditionally used 32-bit floating point [135]. This increase can be demonstrated in real-world benchmarks, as confirmed by the PyTorch team which claims that peak float16 matrix multiplication and convolution performance is 16x faster than peak float32 performance on A100 GPUs [6].

A trivial benchmark to estimate the performance gain obtained by going from float32 to float16 is to compute matrix multiplications. In the code snippet below, two matrices of dimension (1024,1024) of randomly chosen float32 numbers between 0 and 1 are

generated. The time taken to execute the matrix multiplication of these two matrices 100000 times is then recorded. By varying the dtype of the tensors from *torch.float32* to *torch.float16* on a NVidia 4090, the computation time is divided by more than a factor of 3, decreasing from 9.64s to 3.16s. As a baseline, this same operation in double precision takes more than 170 seconds.

```
A = torch.rand((1024, 1024), dtype=torch.float32, device=device)
B = torch.rand((1024, 1024), dtype=torch.float32, device=device)
start = time.time()
for _ in range(100000):
    _ = torch.matmul(a, b)
end = time.time()
print(end-start)
```

CODE LISTING 2.3: PyTorch code used to compute matrix multiplications and measure the time taken depending on the tensor dtype

Float16 also reduces computation time by taking advantage of modern hardware’s ability, for some operations, to execute two float16 operations within the same cycle count required for one float32 operation. In fact, using half-precision is 21% faster than using single-precision when applying a (32, 64) Linear layer to a random input of dimension (16, 32).

```
A = torch.rand((16, 32), dtype=torch.float32, device=device)
B = torch.nn.Linear(32, 64, dtype=torch.float32, device=device)
start = time.time()
for _ in range(1000000):
    _ = B(A)
end = time.time()
print(end-start)
```

CODE LISTING 2.4: PyTorch code used to measure the time taken to apply a single linear layer to a random input depending on the tensor dtype

Although float16 computation is faster than float32, it inevitably introduces a reduction in precision since the number of representable numbers is considerably smaller. Numbers that can be represented in float32 but not in float16 are rounded to the closest numbers available in float16. By creating a random matrix of float32 numbers and copying it to a float16 representation, the difference between the original matrix and the copy results from this precision loss. For numbers in the [0, 1] range, the float16 numbers differ by approximately 1.7% from the original float32 numbers.

```
A = torch.rand((1024, 1024), dtype=torch.float32, device=device)
B = A.to(torch.float16)
C = (((A-B)/A)*100).abs()
print(C.mean())
```

CODE LISTING 2.5: PyTorch code estimating the average loss due to quantization

2.8.3 Mixed Precision

Since reducing the number of bits required to represent a decimal number accelerates the compute time at the cost of precision, a combination of both float32 and float16

can be used to increase the number of operations while retaining precision. Using different precision levels within a single operation is called mixed precision, also known as transprecision [44]. Critical operations requiring high precision are computed in single precision, while other operations are cast to half precision to reduce the compute time [246]. In [Machine Learning](#), the loss is dynamically scaled to avoid both vanishing gradients and exploding gradients. Indeed, as shown in Figure 2.33, more than half of the activation gradients would be directly rounded to 0 by naively converting float32 numbers to float16.

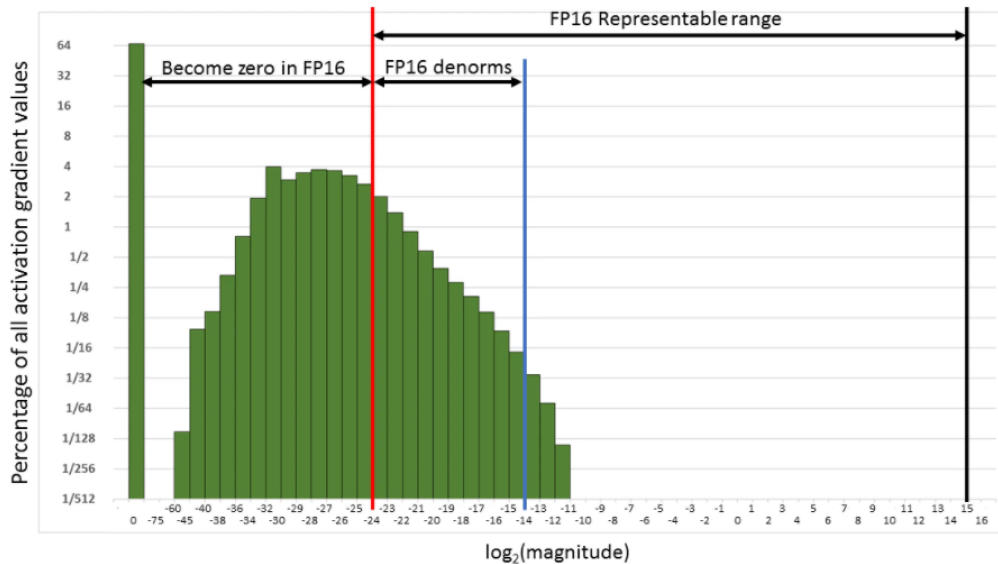


FIGURE 2.33: Activation gradient magnitudes and the difference in representable range between float32 and float16, obtained from [247]

The gradients accumulate in binary32 while most layers are stored and processed in binary16. Using automatic mixed precision, the framework itself chooses which operations are critical and which are not, automatically delivering quick and accurate results.

Figure 2.34 shows the speedup obtained by the PyTorch team over a variety of different models using Automatic Mixed Precision instead of float32 only. The performance gain obtained using Automatic Mixed Precision does not reduce the final accuracy of the trained model, as the model converges towards a similar state. Automatic mixed precision is a technique that, by automatically and appropriately choosing the level of precision required for each operation, makes it possible to significantly improve performance without compromising quality.

2.8.4 Towards Integers

Floating-point arithmetic is inherently more computationally expensive than integer arithmetic, since the underlying hardware units need to support operations such as rounding and avoiding overflow and underflow [110] [250]. This is why, in recent years, a considerable amount of work has focused on the possibility of using integers rather than decimal numbers for Machine Learning. By saving weights as integers and performing

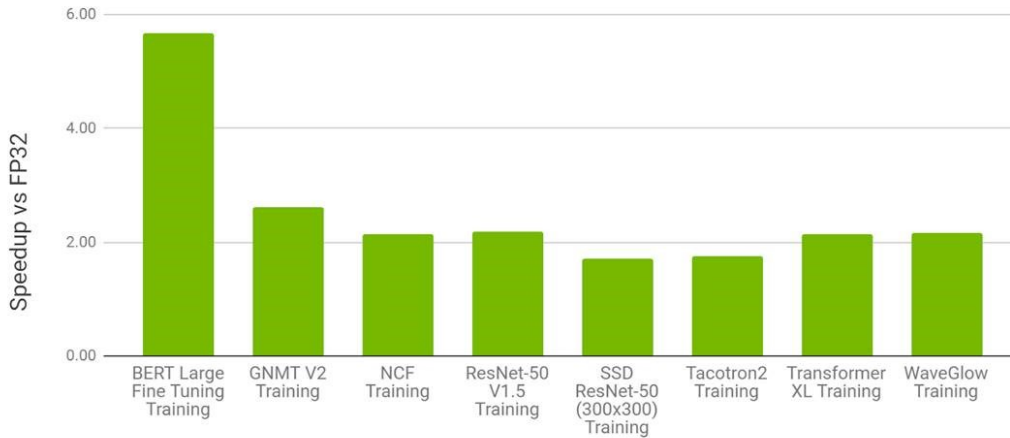


FIGURE 2.34: Speedup of Automatic Mixed Precision versus float32, obtained from [147]

all operations with integers, performance gains can be achieved over using floating-point numbers, at the possible expense of a reduction in quality.

Although it is possible to replace float32 with int32 to obtain an increase in performance, quantising most weights of a neural network to int8 while accumulating gradients in int32 obtains even better performance while still maintaining accuracy [155] [230] [78] [411] [173] [187]. Shorter bit-lengths, such as int5 quantisation [429] [372] and int4 quantization [230] [60] further increase throughput with minimal accuracy loss.

Quantising weights from float32 to int8 also massively decreases the storage requirements by a factor 4, which allows larger models to fit on the same hardware, or allows training using larger batch sizes.

2.8.5 Scaling

Quantizing weights as integers rather than floating-point numbers causes one major problem, and a second one being introduced by the switch from 16- or 32-bit storage to 8 bits or even less. The first is that the range of numbers that can be represented differs between floating point numbers and integers. A signed float32 has a maximum value of approximately 3.4×10^{38} , compared to 2.1×10^9 for a similarly sized int32, and only 127 for a smaller int8. Finally, dividing by 4 the number of bits used to represent a number induces a reduction in the number of representable digits. Thus, rounding has to occur, and some information is lost.

While there are many algorithms to transform float32 weights into int8 numbers, affine zero-point quantisation is among the easiest to understand. The first step is to determine the smallest and largest weights in a layer, min and max . The scale factor depends on the range of values that our destination format supports, which is 255 in the case of int8, and is calculated using the following formula: $scale = \frac{max-min}{255}$. The zero-point is the value in the quantised range that corresponds to the 0.0 value in the pre-quantised float32 state. It is calculated as: $zero = abs(round(\frac{min}{scale}))$ and clamped in the int8 representable range $(-128, 127)$. Finally, the weights are quantised using the formula:

$$\text{quantized\_weight} = \text{round}\left(\frac{\text{original\_weight}}{\text{scale}} + \text{zero}\right)$$

Dequantization is the process of obtaining the original weight from the quantised weight. Since information is lost during quantisation, the true original weight cannot be recovered.

$$\text{dequantized\_weight} = \text{scale} \times (\text{quantized\_weight} - \text{zero})$$

Previous work has shown that it is possible to quantify some binary32 models to int8 while only losing 1% of the accuracy [393].

2.8.6 Quantization-Aware Training

The quantisation of a model can be carried out after the end of its training, which is called [Post-Training Quantization \(PTQ\)](#). This method allows previously trained models to be used and is highly adaptable to existing environments to reduce computational requirements. Quantisation can be carried out dynamically at runtime, or in advance, and the model is saved with the new parameters converted. It is possible to convert only certain layers, depending on the reduction in precision obtained by quantisation, so that the precision of the weights is not reduced when the degradation in performance is deemed too great.

Another way in which quantisation is used is [Quantization-Aware Training \(QAT\)](#). Rather than quantising after a model has been trained, the aim is to predict future quantisation at an early stage. Training is carried out in non-quantised format and the error due to quantisation is simulated at each stage, in order to mitigate this error. Figure 2.35 shows how [QAT](#) reduces the degradation from quantisation compared to [PTQ](#) on the [LLaMA3-8B](#) model.

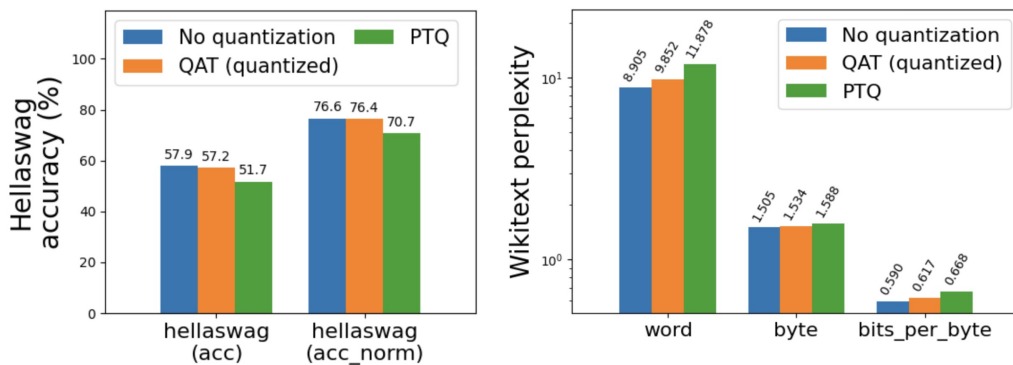


FIGURE 2.35: Accuracy of the [LLaMA3-8B](#) model depending on the quantization type, obtained from [254]

In this review of the literature, we provide a comprehensive overview of key concepts and advances in the field of [Machine Learning](#), with a particular emphasis on [Natural Language Processing](#), and the transformative impact of the Transformer architecture.

After introducing fundamental concepts in linear algebra, such as vectors, matrices, and tensors that serve as building blocks to represent and manipulate data, we introduce the basics of [Machine Learning](#). We explain the concept of a loss function as a measure of model performance and how gradient descent algorithms are used to optimise model parameters to minimise this loss.

We then outline key [Natural Language Processing](#) techniques, such as tokenization (the process of breaking down text into smaller units), Word Embeddings (methods for representing words as numerical vectors) and Legacy Model Architecture, such as [SMT](#) models, [RNN](#) models and [LSTM](#) models.

We discuss the pivotal role of the Transformer architecture, emphasizing its key features, particularly the self-attention mechanism. While the original Transformer model adopts an encoder-decoder structure, variants exist, including encoder-only architectures designed for representation learning and decoder-only architectures optimised for autoregressive text generation. We explain how this architecture can be used for Sequence-to-Sequence tasks, such as [Machine Translation](#) and [Language Modeling](#) using architectures like [GPT](#), [LLaMA](#), and [BERT](#). We explore specific aspects of [Machine Translation](#), such as the evaluation metrics, both non-neural, for example [BLEU](#) and [chrF](#), and neural, such as [BERTScore](#), [BLEURT](#), and [COMET](#).

We explore additional generative architectures and frameworks, such as the [AutoEncoder](#) (an encoder-decoder model which compresses and then reconstructs its input), the [GAN](#) (a training framework where a generator generates synthetic data and the discriminator tries to distinguish the data belonging to the dataset from the data synthetically created by the generator), and the CycleGAN (a framework consisting of two [GANs](#) where the output of one generator is used as the input of the other and vice versa, and relies on a [Cycle-Consistency Loss](#) to train the networks).

We also review style transfer techniques in various domains, such as image style transfer and audio style transfer, but we mainly focus on text style transfer methods, focussing on sentiment, formality, and other stylistic attributes.

Finally, we explore quantisation techniques used to reduce the memory footprint and computational cost of models while maintaining accuracy.

In this review of the literature, our objective is to establish a foundational understanding of core concepts and techniques in [Machine Learning](#), setting the stage for further investigation into specific [Natural Language Processing](#) tasks and advances.

Chapter 3

Machine Translation Evaluation

All translation is a compromise
– the effort to be literal and the
effort to be idiomatic

Benjamin Jowett

In Chapter 2, we provided a comprehensive review of [Neural Machine Translation](#), examining the evolution of various architectures with a particular focus on the transformative power of the transformer model. We also explored [Large Language Models](#) and highlighted how the pre-trained embedding distance can serve as a tool to assess the similarity of sentences. Building on these foundations, we introduce in this chapter two new metrics for evaluating the quality of machine-generated translations, which we developed and published. In this chapter, our aim is to answer the Research Question 1. How can non-neural evaluation metrics and [Large Language Models](#) be leveraged to improve the assessment of machine translation quality while maintaining efficiency?

3.1 Introduction

The recent development of automatic text translation systems has led to the emergence of numerous translation evaluation algorithms. The aim of these algorithms is to be able to evaluate automatically generated translations, called hypotheses, most often by comparing them with references, i.e. the ideal translation desired. The closer an algorithm’s ranking is to that of a native speaker or professional translator, the better. In this way, it is possible to compare translation models for a fraction of the price, making large-scale comparisons between systems that are faster and simpler.

In order to extend research in the field and to better understand the diversity of evaluation techniques with a view to creating an architecture for unsupervised training for [Machine Translation](#), which is at the heart of this thesis, two distinct [Machine Translation](#) evaluation tools have been developed and published.

The first is based on the idea that modern tokenisation algorithms, by extracting the most frequent subwords, add information that is not present in whole words. In this way, comparing subwords rather than whole words increases the granularity of the overlap analysis. The second algorithm is based on the use of embeddings and deeper layers from [Large Language Models](#) trained on tasks other than text translation, in order to study the generalisation of these models. By measuring the distance between the vectors generated by these models, it is possible to assess the semantic proximity of two sentences.

3.2 Evaluation from Tokenization

Tokenization is the process of transforming a text-based sentence into a numerical representation. At its heart lies the conversion of characters, phonemes, n -grams, words and groups of words into unique identifiers. Section 2.2.1 offers a more refined explanation of this process. Tokenization algorithms create tokens according to the frequency with which they are used in a dataset, so additional information is obtained when the vocabulary is generated. This additional information can be useful when evaluating machine-generated translations, providing a more refined comparison between the hypothesis and the reference.

This hypothesis motivated Tokengram_F, the [Machine Translation](#) evaluation algorithm based on chrF++ presented in this Section and was developed as part of this research.

3.2.1 A Deeper Dive into chrF++

As mentioned in Section 2.5.2, chrF [273] is an algorithm that relies on character n -grams to evaluate the quality of generated translations, by comparing the overlap of these n -grams between the hypothesis and the reference. chrF++ [274] is an improvement over chrF which also adds word n -grams. This refinement increases the correlation of this score with human evaluations, as word n -grams capture sequences of words which maintains the semantic and syntactic integrity of sentences.

This sections focuses on the inner workings of the chrF++ algorithm as both an example of a tokenization-based evaluation score and as the inception of the Tokengram_F score. To make it clearer to understand chrF++, the evaluation of a reference/hypothesis sentence pair will be deconstructed using the original source code¹:

Reference: “the sovereign is the king of Northern England”

Hypothesis: “The sovereign is the ruler of Scotland”

¹<https://github.com/m-popovic/chrF/>

Parameters

chrF++ relies on three different parameters:

- *nworder* defines the maximum word *n*-gram order. The metric examines word *n*-grams from 1 up to the specified *nworder* value, which by default is 2.;
- *ncorder* is the maximum character *n*-gram order. As is the case with *nworder*, all character *n*-grams from 1 to *ncorder* are included, with the default value being 6;
- β balances the ratio between precision and recall, chrF++ being first and foremost an F-score, that is to say a harmonic mean between the precision and the recall. The default value of the parameter β is 2.0, that is to say, precision is twice as important as recall.

The process is identical for both words and characters. As the default value of *ncorder* is 6, the process for characters is omitted for brevity.

Dissecting into *n*-grams

Using the default parameters, chrF++ starts by splitting the words of both the hypothesis and the reference to obtain the following arrays:

```
ref = ["the", "sovereign", "is", "the", "king", "of", "Northern", "England"]
hyp = ["The", "sovereign", "is", "the", "ruler", "of", "Scotland"]
```

Since the *nworder* value is 2, two dictionaries are created for both the reference and the hypothesis, where the first counts the number of case-sensitive occurrences of each word:

```
ref = {"the": 2.0, "sovereign": 1.0, "is": 1.0, "king": 1.0, "of": 1.0, "Northern": 1.0, "England": 1.0}
hyp = {"The": 1.0, "sovereign": 1.0, "is": 1.0, "the": 1.0, "ruler": 1.0, "of": 1.0, "Scotland": 1.0}
```

The second dictionary counts the number of occurrences of groups of two consecutive words:

```
ref = {"the sovereign": 1.0, "sovereign is": 1.0, "is the": 1.0, "the king": 1.0, "king of": 1.0, "of Northern": 1.0, "Northern England": 1.0}
hyp = {"The sovereign": 1.0, "sovereign is": 1.0, "is the": 1.0, "the ruler": 1.0, "ruler of": 1.0, "of Scotland": 1.0}
```

Counting Matches

For all word n -gram dictionaries, the number of exact matches is computed. For $n = 1$, both the reference and hypothesis have in common the words “the”, “sovereign”, “is” and “of”, which obtains a score of 4.0. If the first “The” in the hypothesis was not capitalised, the score would be 5.0 since there would be another exact match.

For $n = 2$, the exact matches are “sovereign is” and “is the”, for a score of 2.0. Once again, if the first word of the ref was not capitalised, the score would be 3.0 since “the sovereign” would be an exact match.

Precision and Recall

The precision, recall and F-score for the first-order word n -grams, that is to say with $n = 1$, are respectively obtained as follows:

- $word\_prec\_n\_1 = \frac{4.0}{8.0} = 0.5$, since there are 4 exact matches on the 8 words contained in the hypothesis;
- $word\_rec\_n\_1 = \frac{4.0}{7.0} \approx 0.571$, as the reference contains 7 words;
- $word\_f\_score\_n\_1 = \frac{(1+\beta^2) \times word\_prec\_n\_1 \times word\_rec\_n\_1}{(\beta^2 \times word\_prec\_n\_1) + word\_rec\_n\_1} \approx 0.563$

The second-order calculations are:

- $word\_prec\_n\_2 = \frac{2.0}{7.0} \approx 0.286$, as there are 2 exact matches and the hypothesis contains 7 bi-grams;
- $word\_rec\_n\_2 = \frac{2.0}{6.0} \approx 0.333$ because there are 6 bi-grams in the reference;
- $word\_f\_score\_n\_2 = \frac{(1+\beta^2) \times word\_prec\_n\_2 \times word\_rec\_n\_2}{(\beta^2 \times word\_prec\_n\_2) + word\_rec\_n\_2} \approx 0.328$

The precisions and recalls for character n -grams from the first to the sixth are also computed even though they are omitted for brevity, since the formulas are identical. The final chrF++ score for the sentence is obtained using:

$$chrF++ = \frac{\sum_{i=1}^{ncorder} chr\_f\_score\_n\_i + \sum_{j=1}^{nworder} word\_f\_score\_n\_j}{ncorder + nworder} \approx 0.519$$

A Preventable Drawback

The chrF++ metric is partially based on exact word n -gram matches, which has the immediate consequence of considering two terms that are semantically close but spelled differently in the same manner as two completely unrelated terms. A misspelt word, a missing plural or another word from the same root are all examples of a word that is counted as not spelt correctly.

Dividing a sentence into a list of words is already a tokenization algorithm. Indeed, tokenization can be performed at different levels:

- Word n -grams: Sequences of words, as used in traditional [BLEU](#) scoring;
- Character n -grams: Sequences of characters, which capture finer-grained similarities and improve robustness to morphological changes;
- Token n -grams: Where tokens may not correspond exactly to words but can be subword units.

Instead of relying on word n -grams, one could explore alternative tokenization strategies to replace the word-based approach in chrF++, potentially leading to improved precision in evaluating [Machine Translation](#) outputs.

3.2.2 Tokengram_F

Modern tokenization algorithms rely on [Machine Learning](#) techniques, in the sense that an algorithm's performance on a given task improves as the amount of training data increases, to determine the most frequent character chains in a dataset in order to create a vocabulary.

The Tokengram_F metric [\[83\]](#) we developed is based precisely on the assumption that the use of a modern tokenization algorithm, to replace word n -grams with token n -grams, could overcome the fact that when the comparison between the reference sentence and the hypothesis is made at the level of whole words, the translation system is only given a favourable score if it has spelt the word perfectly.. Rather than comparing at the word level, comparisons are made at the subword level, which may correspond to common roots and similar meanings².

Tokengram_F relies on the same parameters as chrF++, that is to say β , *ncorder* and *nworder*, even though this last one refers to the maximum number of token n -grams instead of word n -grams. Two parameters linked to the use of a modern tokenization algorithm are added, namely, the size of the vocabulary and the choice of algorithm.

Vocabulary Size and Tokenization Algorithm

One of the two additional parameters that Tokengram_F requires is a parameter required by the two underlying modern tokenization algorithms. The vocabulary size, i.e. the number of different unique tokens into which words will be converted, is fixed and chosen before training. This size partly determines the size of the neural networks used to process the tokenized text, since the embeddings size depends directly on the number of elements in the vocabulary.

The ideal vocabulary size is large enough to reduce out-of-vocabulary words and allow words sharing a common root to share common tokens, but small enough not to make the neural network too computationally expensive.

²The original publication is reproduced in Appendix [B.1](#).

The second parameter concerns the tokenization algorithm used, which can be either [Byte Pair Encoding](#) or Unigram.

Unigram

Unigram tokenization [188] starts from a large initial vocabulary containing all possible subword candidates. The algorithm then iteratively prunes this vocabulary by removing lower-probability subwords, keeping only those that contribute most effectively to representing the text, gradually refining the vocabulary to an optimal set. This pruning process continues until a target vocabulary size is reached, resulting in a compact, efficient set of subwords that balance coverage and conciseness.

In order to detail the internal workings of the Unigram model training, i.e. the way in which it selects the tokens to be added to the vocabulary, a brief example is constructed from an imaginary dataset inspired by [149]. This dataset contains 5 different words and the frequency is as follows:

```
{"cat": 10, "dog": 5, "the": 12, "pun": 4, "cats": 5, "pug": 1}
```

The most frequent subwords are:

```
{"ca": 15, "at": 15, "th": 12, "he": 12, "do": 5, "og": 5, "cat": 5, "pu": 5}
```

As the sum of all frequencies is 74, the probability of the subword “at”, which is in both “cat” and “cats”, is $\frac{15}{74}$.

The probability of a segmentation $S = s_1, s_2, \dots, s_n$ is obtained using the formula:

$$P(S) = \prod_{i=1}^n P(s_i)$$

where $P(s_i)$ is obtained by:

$$P(s_i) = \frac{\text{count}(s_i)}{\sum \text{counts\_of\_all\_subwords}}$$

To tokenise the word “that”, tokenization “t” “h” “a” “t” has probability:

$$P(["t", "h", "a", "t"]) = P("t") \times P("h") \times P("a") \times P("t") = \frac{27}{74} \times \frac{12}{74} \times \frac{15}{74} \times \frac{27}{74} \approx 0.0043759$$

while tokenization “th” “at” has the probability:

$$P(["th", "at"]) = P("th") \times P("at") = \frac{12}{74} \times \frac{15}{74} \approx 0.0328707$$

The other possible tokenization would have the following probabilities:

- $P(["th", "a", "t"]) \approx 0.0119933$

- $P(["t", "h", "at"]) \approx 0.0149917$

Thus, the word “that” is tokenized using the token combination that has the highest likelihood, that is to say using the tokens “th” and “at”. With larger vocabularies, finding this highest likelihood is computationally expensive. The Viterbi algorithm finds the optimal unigram tokenization of a word by constructing a graph where branches represent valid subwords from the vocabulary, each assigned a probability. It iteratively determines the best segmentation ending at each position by considering all possible subwords and leveraging prior best scores. The final segmentation is obtained by backtracking from the endpoint.

In order to fit the requested vocabulary size, the subwords that are used infrequently to reconstruct words present in the datasets are discarded during training, the least used tokens being deleted first. The negative log-likelihood function penalises segmentations with excessive short tokens, leading to an effect similar to a length penalty.

BPE

[BPE](#) is analogous to a “reverse Unigram” in that rather than starting with long subwords and then subdividing them into shorter units, [BPE](#) begins from letters and tries to find longer units of text by merging the letters.

Using the same corpus as in the previous section, the most common pair of letters, in alphabetic order, is “a t”, with 15 occurrences. Thus, the token “at” is added to the vocabulary. The most frequent pair is now “c at” with 15 occurrences, thus adding the three-letter token “cat” to the vocabulary. The tokenizer merges pairs until the desired vocabulary size is reached.

3.2.3 Training

Unlike other metrics such as BLEU or chrF++, `Tokengram_F` needs to be trained before it can estimate the quality of a translation. For each language, a tokenizer must be trained, which requires a monolingual dataset. During prototyping, to compare the performance of `Tokengram_F` according to the different parameters of the tokenizer, namely the size of the vocabulary and the choice of the algorithm between Unigram and [BPE](#), several tokenizers were trained for each language.

Framework

`SentencePiece` [189] is a fast data-driven text tokenizer and detokenizer implementing the Unigram and [BPE](#) algorithms. The vocabulary size (number of individual tokens) needs to be provided before training. The minimum vocabulary size would consist of the number of special tokens and individual characters of the alphabet for each language. A large vocabulary size might lead to overfitting and a reduction in the effectiveness of the model, given that some parameters will be dedicated to rare words.

Dataset

The Tatoeba Translation Challenge [348] is an initiative that aims to evaluate the effectiveness of [Machine Translation](#) systems on a large, diverse, and high-quality parallel corpus. Although the main training data are based on OPUS [347], which provides open source sentence-aligned text corpora to support data-driven [Natural Language Processing](#), Tatoeba also provides monolingual datasets extracted from CirrusSearch Wikimedia dumps [97].

When the dataset is too small, it is possible that when the tokenizers are trained, too few unique subwords are extracted because the dataset itself contains too few different words. This scarcity limits the algorithm’s ability to find meaningful subwords that capture patterns in the data, which in turn restricts the pruning and selection processes needed to reach the target vocabulary. Consequently, the tokenizer may converge on a smaller vocabulary than intended. Hence, of the 279 different languages available, 240 had a sufficiently large corpus to train large tokenizers with the appropriate vocabulary size.

Finetuning

The optimal parameters, defined as the parameters producing the best correlations with human ranking, irrespective of the computational cost, were determined based on achieving the highest average correlation among segments or systems across three datasets: [WMT20](#) [27], [WMT21](#) [7] and [WMT22](#) [180].

Initially, character-gram length was assessed at values of 3, 6, and 9, while maintaining a vocabulary size of 50,000. Consistent with the findings of the original chrF++ work [274], a character n -gram length of 6 demonstrated the strongest correlation, and therefore it was chosen for subsequent evaluations.

Different vocabulary sizes were then compared (8,000, 16,000, 32,000 and 50,000). A vocabulary size of 8,000, which was the smallest size examined, exhibits generally weaker correlations compared to larger sizes, while the largest vocabulary produced the best results. Using even smaller vocabularies would likely lead to further performance degradation due to increased segmentation and fragmentation of words into excessively small subword units, which carry less contextual information and may weaken the model’s capacity to leverage relevant linguistic patterns, thus precluding the exploration of smaller sizes.

The choice of the tokenization algorithm only slightly affects the results. Subsequently, the β values of 2, 3, and 4 were examined specifically for a character n -gram length of 6.

3.2.4 Results

Despite the marginal disparity, when the results are not rounded, the parameters for Tokengram_F which obtain the highest correlation with human ranking are found to be a vocabulary size of 50,000, Unigram tokenization, a character n -gram length of 6, a token-gram of 2 and a β value of 3.0.

Table 3.1 displays a subset of the results obtained using a vocabulary size of 50,000 depending on the other parameters.

TABLE 3.1: Correlations between system and sentence translations with human ranking over all language pairs depending of the hyperparameters of Tokengram_F with a vocabulary size of 50,000

Vocabulary	50000									
Algorithm	BPE					Unigram				
Beta	4	3			2	4	3			2
<i>character-gram</i>	6	9	6	3	6	6	9	6	3	6
System WMT20	0.875	0.871	0.876	0.871	0.877	0.876	0.871	0.876	0.871	0.877
System WMT21	0.715	0.715	0.716	0.713	0.717	0.716	0.715	0.717	0.714	0.718
System WMT22	0.837	0.831	0.834	0.834	0.829	0.836	0.831	0.835	0.834	0.830
Segment WMT20	0.277	0.274	0.277	0.279	0.277	0.276	0.273	0.277	0.278	0.277
Segment WMT21	0.158	0.155	0.158	0.166	0.157	0.160	0.159	0.158	0.170	0.152
Segment WMT22	0.398	0.393	0.398	0.399	0.400	0.398	0.393	0.399	0.397	0.400
Average	0.543	0.540	0.544	0.543	0.543	0.544	0.540	0.544	0.543	0.542

Tokengram’s advantage over chrF++ lies mainly in the correlations between segments and not at the system level, as displayed in Tables 3.2, 3.3 and 3.4, which compare the results obtained respectively by chrF++ and Tokengram_F on the WMT20, WMT21 and WMT22 datasets. In the segment-level task of the WMT22 edition, Tokengram_F managed to obtain better overall correlations than any other metric that could provide results for all language pairs, including ones requiring extensive neural networks to operate. Complete results are available in Appendix A.1.

Although chrF++ mitigates some of the strictness of word n -gram matching by incorporating character n -grams, it still lacks the ability to fully capture morphological and semantic relationships between words. Character n -grams focus primarily on surface-level similarities, which helps in handling minor spelling variations (“color” and “colour”) but does not account for meaningful linguistic structures such as affixes and common word roots. In contrast, token n -grams, as implemented in Tokengram_F, provide a more linguistically informed representation by leveraging subword tokenization techniques.

This allows the metric to recognise morphological variations (“running” and “runner”) and semantic similarities (“government” and “governance”) more effectively than word n -grams.

TABLE 3.2: Tokengram_F results on the WMT20 dataset compared to chrF++

Language pair	Tokengram_F system r	chrF++ system r	Tokengram_F segment τ	chrF++ segment τ
en-cs	0.865	0.833	0.485	0.478
en-de	0.961	0.958	0.371	0.367
en-ru	0.981	0.952	0.162	0.156
en-ta	0.941	0.956	0.590	0.579
en-zh	0.851	0.983	0.403	0.388
en-ja	0.949	0.328	0.521	0.506
en-pl	0.958	0.315	0.256	0.255
en-iu	0.433	0.338	0.340	0.338
cs-en	0.872	0.844	0.095	0.09
de-en	0.997	0.998	0.440	0.435
pl-en	0.508	0.970	0.032	0.034
ta-en	0.957	0.522	0.184	0.186
km-en	0.984	0.965	0.281	0.275
ps-en	0.894	0.964	0.143	0.145
ja-en	0.972	0.763	0.251	0.245
ru-en	0.921	0.977	0.055	0.054
zh-en	0.960	0.841	0.130	0.130
iu-en	0.765	0.726	0.242	0.246
Mean	0.876	0.791	0.277	0.273

TABLE 3.3: Tokengram_F results on the WMT21 dataset compared to chrF (chrF was used in place of chrF++ as chrF++ where the results were not reported)

Language pair	Tokengram_F system r	chrF++ system r	Tokengram_F segment τ	chrF++ segment τ
en-cs	0.978	0.970	0.549	0.531
en-zh	0.625	0.549	0.121	0.092
en-ha	0.760	0.748	0.185	0.186
en-ja	0.967	0.966	0.384	0.371
en-ru	0.756	0.943	0.214	0.201
en-de	0.842	0.831	0.448	0.098
cs-en	0.562	0.562	-0.052	-0.053
zh-en	0.269	0.723	0.395	-0.035
ha-en	0.921	0.924	0.021	0.021
ja-en	0.823	0.831	0.006	0.005
ru-en	0.579	0.593	-0.123	-0.126
de-en	0.424	0.357	-0.151	-0.162
fr-de	0.655	0.646	0.049	0.054
de-fr	0.504	0.498	0.111	0.110
bn-hi	0.949	0.941	0.079	0.071
hi-bn	0.877	0.872	0.335	0.327
xh-zu	0.999	0.998	0.306	0.301
zu-xh	0.997	0.999	0.529	0.530
Mean	0.749	0.775	0.189	0.140

TABLE 3.4: Tokengram_F results on the WMT22 dataset compared to chrF++ (chrF was used in place of chrF++ as chrF++ where the results were not reported)

Language pair	Tokengram_F system r	chrF++ system r	Tokengram_F segment τ	chrF++ segment τ
en-cs	0.602	0.689	0.077	0.147
en-zh	0.248	0.210	-0.044	0.051
en-hr	0.899	0.920	0.274	0.185
en-ja	0.927	0.931	0.241	0.142
en-liv	0.989	0.988	0.370	0.101
en-ru	0.852	0.813	0.659	0.153
en-uk	0.869	0.895	0.178	0.177
en-de	0.799	0.811	1.000	0.085
liv-en	0.985	0.969	0.500	0.184
zh-en	0.787	0.881	0.415	0.071
sah-ruh	1.000	1.000	0.856	0.430
uk-cs	0.971	0.979	0.350	0.171
cs-uk	0.921	0.927	0.311	0.195
Mean	0.834	0.847	0.399	0.161

Since the scores are paired between chrF++ and Tokengram_F, and there is no reason to assume that they follow a normal distribution, the Wilcoxon signed-rank test was used to assess whether Tokengram_F outperforms chrF++, specifically by achieving higher

correlations with human rankings. Table 3.5 shows the p-values obtained by this test on all the language pairs across WMT20, WMT21 and WMT22. Although Tokengram_F is not significantly better than chrF++ on the system task, it performs significantly ($p < 0.05$) better than chrF++ on the segment task, with a mean correlation of 0.277 compared to 0.194 for chrF++, which is a small to moderate effect (Cohen’s d of 0.4).

TABLE 3.5: Wilcoxon signed-rank test results comparing Tokengram_F and chrF on WMT20, WMT21 and WMT22

Wilcoxon score	477	142
Expected Wilcoxon	612	612
p-value	0.357	<0.001
Cohen’s d	0.089	0.401

Compared to neural metric, Tokengram_F does not require a large dataset, as it only needs to train a tokenizer. With the dataset used, Tokengram_F can estimate the quality of machine-generated translations in 240 different languages, far more than the 100 languages supported by models of the COMET family, which won both the WMT22 and WMT23 metrics tasks.

3.2.5 Limitations

Tokengram_F suffers from the same shortcomings as other non-neural metrics. Although the use of a trained tokenizer limits these deficiencies, Tokengram_F measures surface-level similarity and cannot understand the context. Synonyms are discarded, since the only measure of quality relies on the overlap of tokens, and the algorithm is incapable of understanding nuance.

3.2.6 Future Work

The main innovation brought about by Tokengram_F is the use of a pre-trained tokenizer to improve the efficiency of non-neural metrics. Rather than comparing n -grams chosen arbitrarily, usually according to the number of characters or by dividing by word, this division by token makes it possible to retain some of the information that would normally be lost by these metrics.

The Tokengram_F metric replaces word-grams used in chrF++ with token-grams. However, future work could determine whether it would be preferable to add token-grams without removing the word-grams.

This approach could therefore be applied to metrics other than chrF++. For example, a derivative of BLEU could compare tokens sliced by a pre-trained tokenizer rather than comparing words. The process could be further refined by only keeping tokens longer than a certain number of characters, a number of characters that would depend on the language.

Tokengram’s tokenizers all come from the same dataset, the Tatoeba Translation Challenge. Larger datasets with more languages could increase the efficiency and versatility of Tokengram_F.

The only tokenization algorithms available in the current version of Tokengram_F are BPE and Unigram. Other algorithms such as WordPiece [80], from Google, exist and might offer better performance. Although WordPiece is not open source, independent implementations are available [150].

We have explored how a metric based on the overlap between subwords cut by pre-trained tokenizers improves the efficiency of a non-neural metric and makes it possible, by requiring a small monolingual dataset, to evaluate machine-generated translations of low-resource languages. At the opposite end of the spectrum, we now examine the use of very large models, trained on large datasets, to evaluate machine-generated translations.

3.3 Evaluation from Pre-trained Large Language Models

Large Language Models are transformer-based models trained on gigantic corpora, with numbers of parameters in the billions. Section 2.4.4 offers a deeper dive into LLMs and the wide array of functions they can fulfil. As mentioned in Section 2.5.2, fine-tuned LLMs can be used to evaluate machine-generated translations simply by prompting them, for example, to assign a score to the hypothesis. However, this section focuses on models that have not been fine-tuned for tasks such as following instructions or answering questions but have only been pre-trained on tasks like Causal Language Modeling. Embed_llama is a metric derived from the assumption that the embeddings contained in pre-trained LLMs can capture the semantic meaning of sentences.

3.3.1 Embeddings Distance

As discussed in Section 2.5.2, the distance between vectors obtained from trained embeddings can be used to estimate the similarity between two sentences. This method of computing the proximity between vectors is one of the main applications of the vector space model. The Gensim library [293] is a collection of trained models designed for purposes such as similarity retrieval, using algorithms such as the word2vec family [234]. Bag-of-Words models are discussed in more details in Section 2.2.2.

By computing the proximity between two sentences, or to be more specific the proximity between the vectors resulting from embeddings, a metric can estimate the quality of an automatically generated translation by comparing it with a reference translation.

3.3.2 Models

In order to reduce computational costs, to precisely examine the effect of the number of layers and to guarantee the reproducibility of the research, only open-source LLMs were

considered in the work leading to the present thesis. Thus, models such as GPT4 [253], were excluded, since they are continuously updated, preventing reproducibility. Also, none of the GPT-4, GPT-4o and o1 APIs permit the use of a specific number of layers.

LLaMA [353] is a **LLM** developed and trained by Meta, which has been released under an open-source licence for research purposes. Several models are available, depending on the number of parameters of the model, ranging from 7 billion to 65 billion. The smallest model has been pre-trained on 1 trillion text tokens, exclusively on languages relying on the Latin and Cyrillic alphabets.

GPT-J [365] is another open-source **LLM**, trained by Eleuther AI. Since it was only trained on The Pile [104], an English-only text dataset, the tokenizer produced too many out-of-vocabulary tokens on text from non-Latin alphabets during preliminary research, thus preventing its use. GPT-NeoX [33] is another Eleuther AI model, which was also only trained on this same dataset. Furthermore, since it contains 20 billion parameters, it could not fit unquantised on consumer-grade hardware available at the time of writing this document.

3.3.3 Distance Function

Figure 3.1, offers a classification of the different functions that can be used to determine the similarity between sentences [363].

In order to qualify as a similarity metric, a distance function has to satisfy several criteria [290]:

- The distance between a sentence and itself must be 0;
- The distance between a sentence A and a sentence B must be equal to the distance between sentence B and A;
- The distance must always be non-negative.

Several distance functions satisfy these criteria. The following list is not exhaustive; however, the functions included are frequently utilised in the literature and are briefly described to facilitate understanding. To ensure clarity, we will use the following terminology to explain the different distance functions: A and B are two embeddings of size n representing two different sentences whose similarity we want to evaluate, with A_i and B_i the elements of these vectors.

Manhattan Distance

The L_1 norm, also known as the Manhattan distance, measures the distance between two points by summing the absolute differences of their coordinates in each dimension. The formula is:

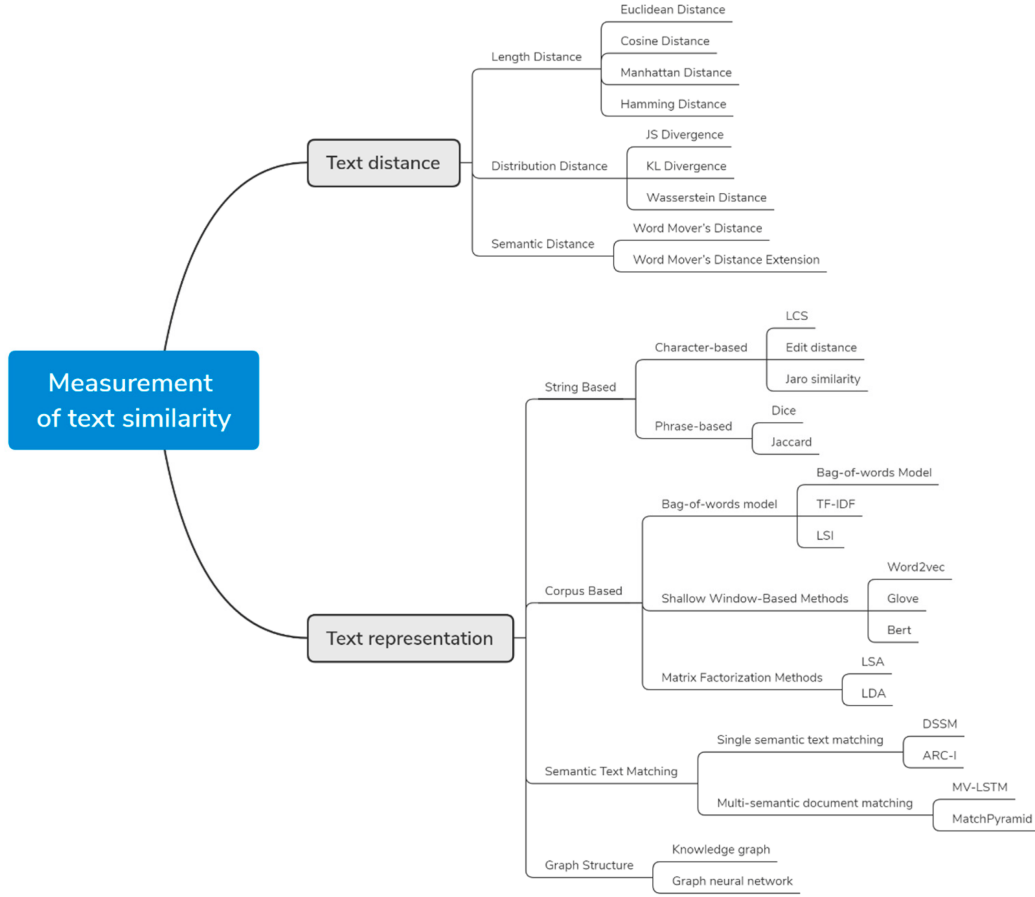


FIGURE 3.1: Classification of text similarity functions, obtained from [368]. Text distance is subdivided into length-based, distribution-based, and semantic-based approaches, including Euclidean distance, KL divergence, and Word Mover’s Distance. Text representation methods range from string-based techniques (e.g., edit distance and Jaccard similarity) to corpus-based models (e.g., TF-IDF, Word2Vec, [BERT](#)), as well as semantic text matching and graph-structured representations like knowledge graphs and graph neural networks

$$D(A, B) = \sum_{i=1}^n |A_i - B_i|$$

The Manhattan distance has shown excellent performance to measure the similarity between groupings of documents, generated using [Term Frequency-Inverse Document Frequency](#) and K-means [307] or K-nn [91].

Hamming Distance

The Hamming distance measures the number of positions at which two strings of equal length differ by comparing characters one-by-one. The Hamming distance exhibits excellent performances in estimating similarities between sentences [140] and its formula is:

$$D(A, B) = \sum_{i=1}^n \delta(A_i, B_i)$$

where $\delta(A_i, B_i)$ is defined as: $\delta(A_i, B_i) = \begin{cases} 0, & \text{if } A_i = B_i \\ 1, & \text{if } A_i \neq B_i \end{cases}$

Euclidean Distance

The Euclidean distance, also known as the norm L_2 , is the square root of the sum of the squared differences between the corresponding components of the vectors, which is also the straight-line distance between two points in a multidimensional space. The Euclidean distance is computed as:

$$D(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

The Euclidean distance seems to be an ineffective method for measuring the similarity between text documents [145].

Jaccard Distance

The Jaccard distance measures the number of identical elements between two sentences, by treating the similarity problem as an intersection of sets. The Jaccard distance formula is:

$$D(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Cosine Distance

The cosine distance measures the cosine of the angle between two vectors, where the smaller the angle, the closer the vectors are, which means a high similarity. The cosine distance formula is:

$$D(A, B) = 1 - \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

3.3.4 Embed_llama

Embed_llama [82] is a translation evaluation metric, which has a limited number of options, each offering a wide range of possible configurations. Since Embed_llama is based on a pre-trained model, the choice of model to use is the first parameter, and many Open Source LLMs are now available. Following this is the choice of how the distance between vectors is calculated. Finally, Embed_llama does not rely solely on embeddings, but can take advantage of the successive layers of pre-trained models. Thus, the last parameter determines the number of layers to be used to generate the vectors. A reproduction of the original publication is available in Appendix B.2.

Llama2

When the [WMT23](#) metrics evaluation task was announced, [LLaMA](#) Version 2 [354] had just been made available. This version, available in three different sizes, 7 billion, 13 billion, and 70 billion, was pre-trained on twice the number of tokens as the previous version, that is, 2 trillion. As all the [LLaMA2](#) benchmarks supplied by Meta showed superior results to [LLaMA1](#), this latest version was used in the present work, in its 7 billion version, as it is the only size that the available hardware could run.

Blocks in Llama2

As shown in Figure 3.2, each block, denoted as *LlamaDecoderLayer*, is structured with several components, including an attention layer, two normalisation layers, and a multilayer perceptron. The multi-layer perceptron, in turn, consists of three linear layers along with an associated activation function, while the attention layer also includes a rotary embedding layer.

The [LLaMA2](#) model, comprising 7 billion parameters, includes an embedding layer, 32 blocks, and a projection layer.

```
LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
```

FIGURE 3.2: Architecture of the [LLaMA2](#) model, which is an open-source decoder-only [LLM](#) trained autoregressively, as displayed in the Huggingface library

Cosine Similarity

In order to estimate the similarity between vectors, the cosine distance was chosen as the function. One of the main advantages of measuring the angle between the vectors rather than measuring the distance is that similar documents might be far apart but share the same orientation, capturing more semantic similarity. As the cosine distance only measures the similarity in direction, it is invariant to the differences in magnitude the vectors might have.

Furthermore, the cosine distance typically performs better in sparse vector spaces because it emphasises the alignment between vectors instead of their absolute values or coordinates, making it less affected by sparsity.

Since BERTScore is based on the [BERT](#) model, which includes a classification token [CLS] at the beginning of each sequence and is commonly used in classification tasks, one could expect this metric to rely on the embedding of that token. However, the authors opted for IDF-weighted pooling, ensuring that all tokens contribute to the final embedding used for distance computation. In contrast, as [LLaMA2](#) does not have an equivalent to the [CLS] token, Embed_llama employs a similar strategy as BERTScore, mean pooling, ensuring that all tokens contribute equally to the final sequence representation.

Number of Layers

While Word2vec exclusively relies on embeddings to generate the vector representation of words and sentences, Embed_llama relies on additional layers, which we refer to as “blocks”, to generate the embeddings to compare. Thus, rather than only relying on the vector space produced by the embeddings, it is possible to compare the vectors produced after a specific number of layers, depending on the language or context. Embed_llama differs from [BERT](#) in the sense that it is a decoder-only model with causal attention instead of an encoder-only bidirectional model. Moreover, [BERT](#) relies on the embeddings of a special [CLS] token while Embed_llama relies on average pooling.

3.3.5 Results

To enable efficient computation on consumer-grade hardware, the two sentences slated for comparison are padded to match the maximum token count of the longer sentence in the pair and not to an arbitrary number since an identical number of tokens is required to compute the distances between the vectors.

Since the embeddings have a dimension of 4,096, a sentence initially shaped as [length] is transformed into a vector of shape [length $\times$ 4,096] following its processing through the layer.

It was initially hypothesised that increasing the number of blocks would improve the contextual representation of sentence meaning. Figure [3.3](#) reveals that, in contrast to our initial hypotheses, the augmentation of block quantities does not significantly modify the correlation between the systems and the ground truth, whether by augmentation or reduction. Furthermore, it is noteworthy that this correlation exhibits variability across different datasets. Specifically, the Pearson correlation between the number of layers and algorithm accuracy is 0.34 for the [WMT20](#) dataset, but it decreases to -0.79 for the [WMT22](#) dataset.

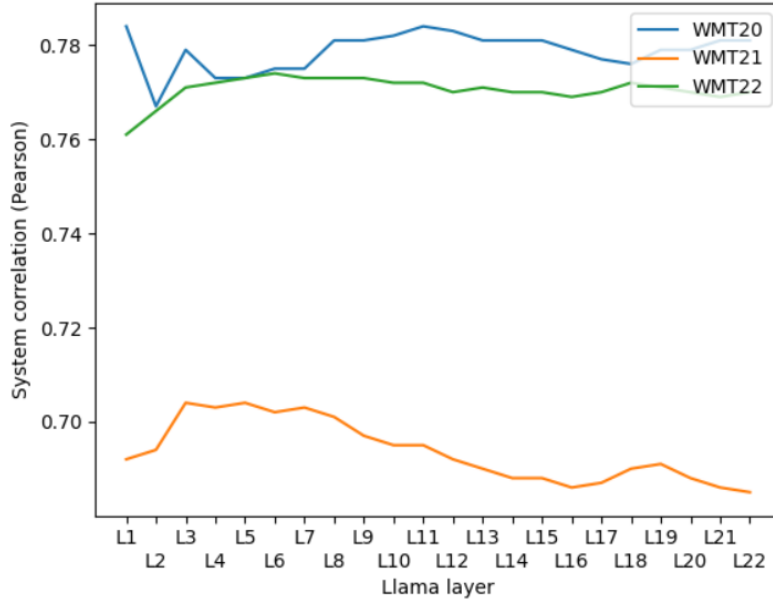


FIGURE 3.3: System correlations with human rankings between the vector space of the reference and the hypothesis, depending on the last **LLaMA** layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the **LLaMA** blocks

As shown in Figure 3.4, these observations also hold with respect to segment correlation. The quantity of blocks employed in the Embed_Llama does not consistently enhance the metric’s quality, whether for individual segments or entire systems.

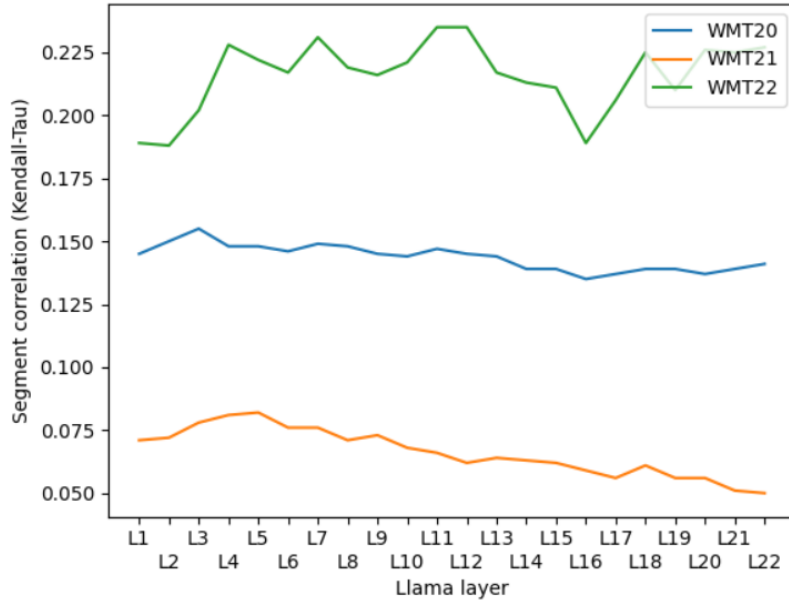


FIGURE 3.4: Segment correlations with human rankings between the vector space of the reference and the hypothesis, depending on the last **LLaMA** layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the **LLaMA** blocks

The highest levels of correlation between human rankings and Embed_Llama rankings were achieved by using a mere two blocks following the embedding layer, resulting in optimal overall performance across the **WMT20**, **WMT21**, and **WMT22** datasets. This not only expedited the computational process, but also decreased the GPU memory demands for metric computation. Full results are available in Appendix A.2.

Inter-languages variations

As depicted in Figure 3.5, the associations between metric quality and language pairs exhibit large variations. For instance, when considering the English-to-Hungarian language pair, the Pearson coefficient between the number of blocks used and the correlation with human rankings registers at 0.83, whereas it falls to -0.85 for the Czech-to-Ukrainian pair, that is to say that on average, each additional block reduces the correlation between the cosine distance and the human ranking in this latter language pair.

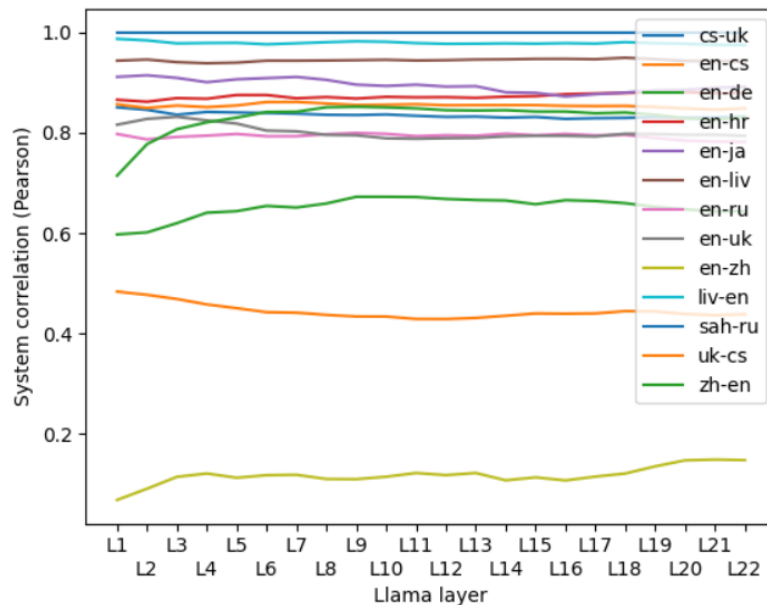


FIGURE 3.5: System correlations with human rankings for each language pair in the WMT2022 between the vector space of the reference and the hypothesis, depending on the last LLaMA layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the LLaMA blocks

Intra-Languages Variations

Considerable variability is observed even within the same language pair across various datasets. For instance, the English-to-Chinese language pair is contained within the WMT20, WMT21, and WMT22 datasets. However, as illustrated in Figure 3.6, no discernible correlations emerge between the number of utilized blocks and the quality of Embed_Llama scores. This is evident in the transformation of the Pearson coefficient, which shifts from -0.28 in the WMT21 to 0.73 in the WMT22 dataset.

This could be explained by the fact that there are domain differences among the datasets used in the WMT20, WMT21, and WMT22 tasks. While the primary focus of the WMT20 dataset is the news domain, WMT21 is an expansion with TED talks, which are characterised by a more informal and disfluent language style, covering a wide variety of topics. Finally, the WMT22 dataset is further diversified by the incorporation of chat translation tasks, reflecting the unique features and sensitivities associated with conversational data.

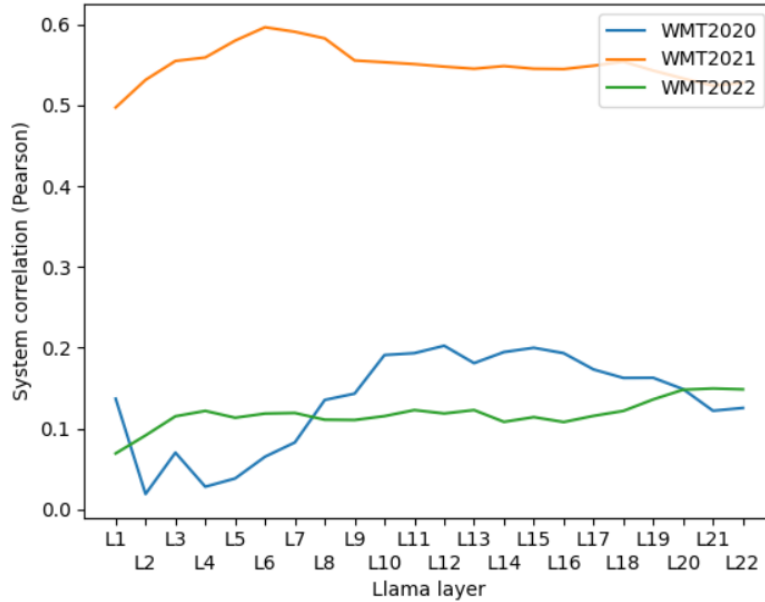


FIGURE 3.6: System correlations with human rankings of the English-to-Chinese language pair between the vector space of the reference and the hypothesis, depending on the last **LLaMA** layer used. L1 corresponds to the embedding layer, while L2 to L22 represent the **LLaMA** blocks

TABLE 3.6: Means and standard deviations of the system correlations with human rankings depending on the last **LLaMA** layer used, when **WMT20**, **WMT21** and **WMT22** are merged. L1 corresponds to the embedding layer, while L2 to L22 represent the **LLaMA** blocks

Language pair	Mean	Standard deviation
en-cs	-0.34	0.22
en-de	-0.02	0.4
en-ja	-0.89	0.04
en-ru	-0.02	0.54
en-zh	0.35	0.45
zh-en	-0.17	0.46

Merged Datasets Variations

Table 3.6 presents the average mean values and their corresponding standard deviations, showcasing the relationship between metric accuracy and the number of utilized blocks for language pairs common to all three datasets. It is notable that, apart for the English-to-Japanese pair, there exists a significant degree of variability in the performance of the same language pairs across different datasets. This variability is underscored by the substantial standard deviations in relation to the absolute values of the means. Given the limited usage of just three datasets, it is essential to acknowledge that the relatively small sample size may hinder the ability to draw conclusive inferences regarding inter-dataset variability.

3.3.6 Limitations

The massive inter-languages variations cause Embed_llama to be a fairly unpredictable quality estimation metric, since the performances on a specific language pair do not forecast performances on other languages. The main weakness of the approach is that the performances of a specific language pair vary from dataset to dataset. Thus, even if particular parameters offer excellent precision on a specified language pair on a single dataset, the performances might not be replicated on another dataset of the exact same language pair.

3.3.7 Llama3

As was expected in the original work, better open-source LLMs have since been released. LLaMA3 [86] is a significant improvement over LLaMA 2, being trained on 15 trillion tokens and released on 14th April 2024. To assess the potential performance gains from using this more recent model, the same benchmarks were used. The WMT20, WMT21 and WMT22 datasets are identical to those used to evaluate the original Embed_llama. Further optimisations allowed the exploration of 24 blocks in addition to the embedding layer instead of the previously computed 22.

The results obtained with this change of model show an overall improvement on the previous version. However, the correlations between the number of blocks used and the Embed_llama results are still rather unpredictable, with massive variations. So, although Embed_llama is not a state-of-the-art metric in its current version, new versions of LLaMA, or another open-source LLM, could make it more competitive.

Figure 3.7, which is an update to Figure 3.3, displays the system correlations in relation to the number of LLaMA3 blocks used. As observed using LLaMA2, the relations are non-linear.

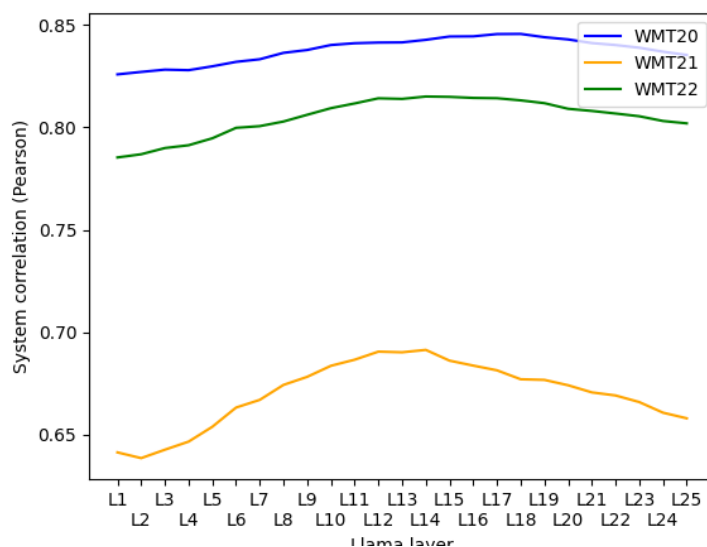


FIGURE 3.7: System correlations depending on the LLaMA layer, using LLaMA3

Similarly, Figure 3.8, the counterpart of 3.4 relying on LLaMA3 blocks, demonstrates that the segment correlations do not reliably increase with vectors generated by deeper layers.

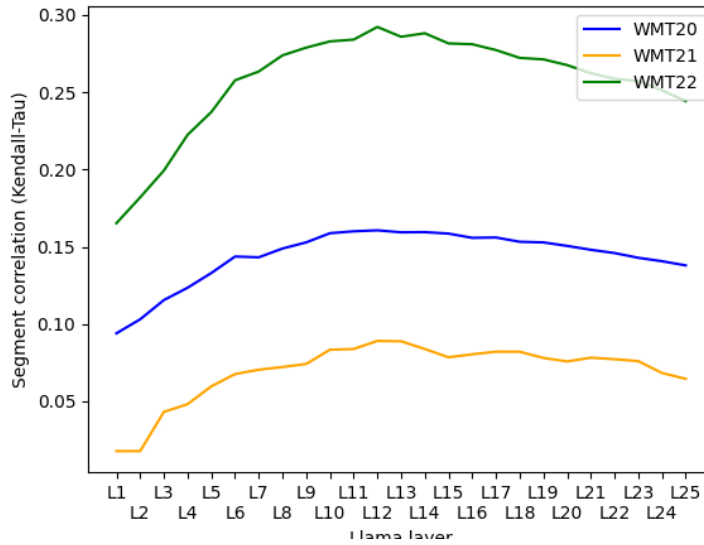


FIGURE 3.8: Segment correlations depending on the LLaMA layer, using LLaMA3

Inter-Languages Variations

Regarding the inter-languages correlations, earlier comments still apply. The variations are so large that earlier results concerning the English-to-Hungarian and Czech-to-Ukrainian do not replicate using LLaMA3. Indeed, while the former language pair had a Pearson correlation coefficient of 0.83 between the number of blocks and the quality of the metric, this coefficient falls to 0.10 using LLaMA3. The same also applies for the latter language pair, as the correlation coefficient soared from -0.85 to 0.67. Figure 3.9 is the equivalent of Figure 3.5 using LLaMA3.

Intra-Languages Variations

Figure 3.10, the analogue of 3.6, demonstrates the massive variations in accuracy of a single-language pair, English to Chinese, depending on the dataset. While the WMT21 and WMT22 exhibit a positive correlation between the number of blocks employed and the accuracy obtained, the opposite is true for WMT20.

Merged Datasets Variations

Table 3.7 references the means and standard deviations of the correlations between the number of blocks and the system evaluations of Embed_llama, using LLaMA3 blocks, for all the language pairs common in all three datasets.

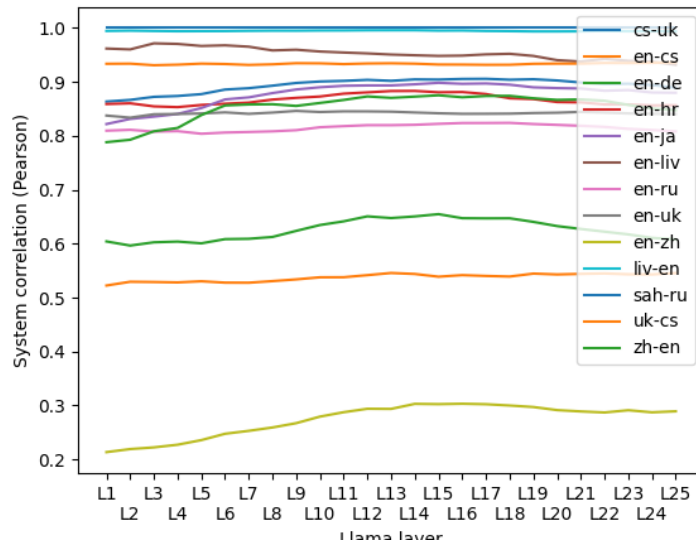


FIGURE 3.9: System correlations for each language pair in the WMT2022 dataset depending on the LLaMA layer, using LLaMA3

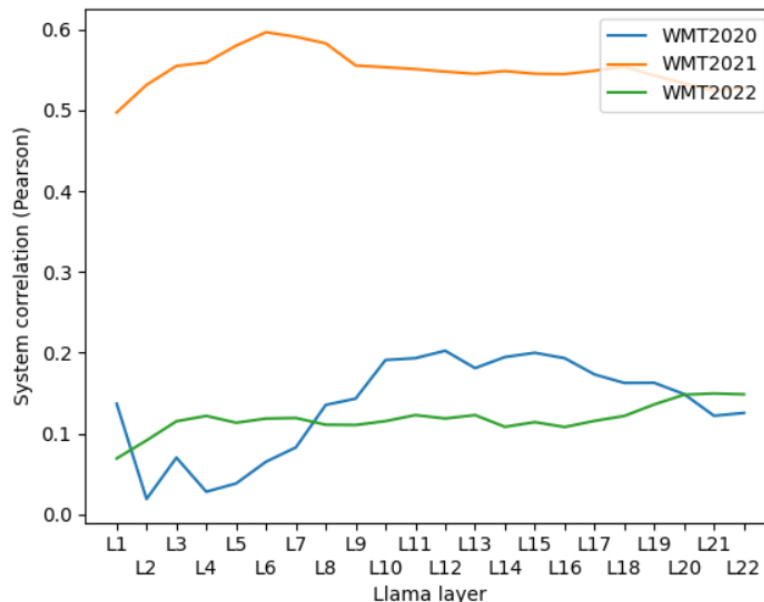


FIGURE 3.10: System correlations of the English-to-Chinese language pair depending on the LLaMA layer and the dataset, using LLaMA3

Contrary to results displayed in Table 3.6, the correlations across the merged datasets for these languages are all positive and except for the English-to-Chinese language pair, are all higher.

3.3.8 Future Work

In the current investigation, the exploration has been confined to the 7 billion parameters LLaMA2 model and the 8 billion parameters LLaMA3 model. It remains conceivable that employing a more extensive model with increased parameters may yield a more precise metric, albeit at the trade-off of heightened computational resource demands.

TABLE 3.7: Means and standard deviations of the system correlations depending on the LLaMA layer when WMT20, WMT21 and WMT22 are merged

Language pair	Mean	Standard deviation
en-cs	0.58	0.25
en-de	0.16	0.65
en-ja	0.64	0.06
en-ru	0.61	0.19
en-zh	0.22	0.78
zh-en	0.22	0.79

In addition, the exploration was limited to the initial 22 blocks for LLaMA2 and 24 blocks for LLaMA3 due to constraints on the memory. Subsequent endeavours may consider augmenting this number, as doing so could potentially result in further benefits. Quantising the model, a process thoroughly described in Section 2.8, would allow both an exploration of larger models and more blocks.

More recent open-source LLMs, such as Mistral 7B [158], trained by Mistral, or Qwen [21], trained by alibaba, were made publicly available shortly after the deadline for the WMT23 task. These newer models might increase the performances of the Embed_llama metric, as they might capture a better semantic representation of the sentences.

LLaMA3.1 [86], the latest LLaMA update at the time of this work, was released in July 2024. By offering a larger context window than earlier versions, it might offer better correlations at the document level than LLaMA3. However, its date of publication left insufficient time to thoroughly evaluate its performance prior to the submission timeframe.

In the scope of the current academic study, solely the cosine distance served as the chosen similarity measure for tensors. Future research endeavors may wish to investigate alternative distance metrics, such as the ones outlined in Section 3.3.3, for potential exploration. Indeed, while the cosine distance is the most common method of computing the similarity of vectors, results might be opaque and arbitrary [336] and previous work has demonstrated that the cosine distance was not always the best distance function [169] [431].

Future work should take into account the proximity of vector embeddings between the source and the hypothesis, rather than focusing solely on the similarity between the hypothesis and the reference.

3.4 Conclusion

The research conducted in this study led to the development of two new metrics for evaluating machine-generated translations. Measuring overlap between tokens generated by pre-trained tokenizers, as done by Tokengram_F, presents an effective and practical

alternative to traditional word overlap measurements. This approach enables the integration of information obtained through machine learning without introducing the added complexity and computational costs of neural networks.

The application of pre-trained LLMs trained on tasks unrelated to text translation for Machine Translation evaluation suggests potential for broader generalisation of these models. By comparing results obtained with Embed_llama based on the underlying model, notable improvements were observed between LLaMA2 and LLaMA3, indicating that this type of metric may achieve greater accuracy and popularity with future model iterations.

How can non-neural evaluation metrics and Large Language Models be leveraged to improve the assessment of machine translation quality while maintaining efficiency?

Non-neural evaluation metrics, such as Tokengram_F, can significantly enhance the efficiency of Machine Translation evaluation by reducing the computational cost compared to neural metrics, while still capturing key linguistic properties like morphological and lexical similarities through *token*-grams. On the other hand, LLM-based approaches like Embed_llama leverage pre-trained language model embeddings to capture semantic similarities between generated and reference translations in a vector space, without requiring end-to-end fine-tuning. However, while Embed_llama offers an alternative by leveraging pre-trained embeddings, its performance still falls short of state-of-the-art neural metrics such as COMET, particularly in terms of correlation with human judgments across diverse language pairs.

Chapter 4

Unsupervised Machine Translation

To translate is to produce
analogous effects using different
means

Paul Valéry

In Chapter 3, we introduced two innovative evaluation metrics for [Machine Translation](#), both of which have been published. The first metric, `Tokengram_F`, based on `chrF++`, demonstrates that token n -grams yield better performance compared to word n -grams, as tokens generated by trained tokenizers contain additional information, such as the roots of the words. Indeed, contrary to character n -grams which can also capture the roots, token n -grams do not “over-fragment” words. `Embed_Llama`, the second metric, shows that sentence similarity can be effectively measured by computing cosine similarity with a pre-trained [Large Language Model](#), even without fine-tuning for this specific task. Building upon this research on the evaluation of machine-generated translations, in this chapter, we introduce `CycleGN`, a framework for unsupervised [Neural Machine Translation](#) that we published. `CycleGN` is based on a cyclic loss approach enabled through pre-training, allowing effective training without parallel data. We aim to answer the Research Questions 2. How can techniques from image style transfer, such as `CycleGAN`, be adapted for unsupervised [Machine Translation](#) in the absence of parallel datasets?

4.1 Introduction

We introduce `CycleGN`, a novel framework for unsupervised [NMT](#). The primary goal of this chapter is to demonstrate the feasibility of training [Machine Translation](#) models using non-parallel datasets through a cyclic approach inspired by the `CycleGAN` architecture. This approach operates under the assumption that the most accurate back-translation of a sentence’s translation should ideally be the original sentence itself.

To train the NMT model, we explore the use of a Cycle-Consistency Loss function, comparing the original sentence to the back-translation of its translation.

We introduce two distinct categories of non-parallel datasets: permuted datasets, in which every sentence has a corresponding translation, although its position within the dataset is unknown, and non-intersecting datasets, in which no sentence has an exact translation within the dataset.

We then explore the early work that laid the foundations for our architecture and the reasons why prototypes could not achieve unsupervised Machine Translation.

We provide a detailed explanation of the architecture of CycleGN, including the vocabulary organisation, the reason behind the absence of a discriminator, and the use of a pre-training method.

We then explore the challenges of this novel architecture, such as its instability during training and the semantic confusions it produces.

Furthermore, we present the findings of CycleGN, namely its result on unsupervised Machine Translation along with comparisons with existing frameworks.

Finally, we conclude the chapter discussing the current limitations of CycleGN and the future work required to improve CycleGN.

4.1.1 A Cyclic Approach for Translation

The aim of this chapter is to show that, quite surprisingly, it is possible to train Machine Translation models using non-parallel datasets. This is done using a cycle-consistent approach, inspired in particular by the original CycleGAN architecture presented in Section 2.6.4. Using this technique—traditionally reserved for style transfer—for text translation could pave the way for a unification of domains, as well as for a massive increase in the size of datasets.

The assumption underlying this whole chapter is that for each sentence, the best translation of its translation is the original sentence itself. As a result, using a Cycle-Consistency Loss between the original sentence and the back-translation of its translation should allow training of a Neural Machine Translation model.

Given two human languages, X and Y , each forming a distribution, the task is to obtain two mappings $X \mapsto Y$ and $Y \mapsto X$. The role of the $\mathcal{G}$ generator is to perform this first mapping, while $\mathcal{F}$ performs the second. It follows that for any $x \in X$, using the mapping $X \mapsto Y$, we would obtain $\hat{y} \in Y$, the translation of x into the language Y , and vice versa to translate $y \in Y$. Applying a Cycle-Consistency Loss between $\hat{x}$ and $\hat{\hat{x}}$, the backtranslation of $\hat{y}$, enables the training of the generators.

At first, since the two generators are initialised with random parameters, the generated intermediate translations will also be random tensors, and when recovering from these intermediate translations, the models will learn how to produce correct sentences from

a random vector, as is the case in a normal GAN network. With time, the networks will generate credible “fakes” and the reconstructions end up sharing the meaning and words of the original sentences. As the best possible way to translate a sentence back and forth and obtain a good reconstruction is for the translation to have the same meaning as the original sentence, the networks should converge toward such a state.

4.1.2 Permuted and Non-Intersecting Dataset

The definition of a parallel dataset is provided in Section 2.5.1. A non-parallel dataset on the other hand, does not consist of pairs of text segments, such that the source and target sentences do not share any explicit correspondence. This dataset can be created by combining any two monolingual datasets and adjusting the number of samples so that both languages have the same number of sentences. A subcategory of non-parallel datasets that warrants mention is comparable corpora [103]. These are collections of texts that share similarities in content, topic, or genre but are not direct translations of one another. Rather than being structured at the sentence level, comparable corpora are typically organised at the document level. They have been used, for example, to train Machine Translation models [124] and to align sentences in two different languages [226].

In the context of this research, two additional subcategories of non-parallel datasets that have their own distinct characteristics are introduced.

Let n be a positive integer representing a sentence in the source language and let n' be its corresponding translation in the target language. Table 4.1 presents the alignments of sentences in a parallel dataset containing 4 sentences.

TABLE 4.1: Representation of a parallel dataset

Source sentence	Target sentence
1	1'
2	2'
3	3'
4	4'

TABLE 4.2: Representation of a permuted dataset

Source sentence	Target sentence
1	3'
2	1'
3	4'
4	2'

A “permuted” dataset is defined as a parallel dataset wherein the sentences of one language have been systematically rearranged. Consequently, this results in a non-parallel corpus, where it is guaranteed that each sentence has a corresponding translation located at an unspecified index within the dataset. We postulate that when employing sufficiently large pairs of monolingual datasets, which are not derived from permuted parallel corpora, it is likely that many sentences in one of the monolingual pairs will possess an accurate translation *somewhere* within the other of the monolingual pair.

In order to obtain a permuted dataset, it is necessary to start from a parallel dataset and reorder the sentences of one of the two languages. In this way, since the dataset was parallel and no sentences were removed, it is guaranteed that each sentence has its translation at an unknown index. This also makes it trivial to truncate the dataset if it

is too long, simply clipping it before permuting one of the languages. Table 4.2 shows the organisation of the sentences in a permuted dataset.

Unlike a permuted dataset, in a non-intersecting dataset, no sentence has its translation. A “non-intersecting” dataset is a non-parallel dataset for which it is guaranteed that no sentence has an exact translation. This type of dataset can be synthesised from a parallel corpus by selecting a subset of sentences from one language and a different non-overlapping subset from the other language. Consequently, contingent upon the precondition that the original corpus is devoid of repetitive sentences, the resultant non-intersecting dataset will similarly be free from such duplicates.

A non-intersecting dataset is derived from a curated parallel dataset that is completely devoid of duplicate entries. Two unique sets of integers are produced, each acting as an index list of phrases to retain from each respective language. This process can be implemented, for example, by selecting sentences with an even index for one language and those with an odd index for the other. Alternatively, it may involve retaining only the first half of the sentences from the first language and the second half from the second language. Table 4.3 shows the representation of a non-intersecting dataset.

TABLE 4.3: Representation of a non-intersecting dataset

Source sentence	Target sentence
1	8'
2	5'
3	7'
4	6'

A non-intersecting dataset can be considered the “worst-case scenario” of a pair of monolingual datasets, in which no sentence has its translation, while a permuted dataset is the “realistic best-case scenario” for a pair of monolingual datasets, as all sentences have their translations “somewhere”, at an unknown index, as it could be argued that the true “best-case scenario” for a pair of monolingual datasets is that this pair is, in fact, a parallel dataset.

4.1.3 Dataset

Since its third instance in 2006, the [Workshop on statistical Machine Translation \(WMT\)](#) has released annual translation tasks to compare [Machine Translation](#) algorithms. The dataset used in the preliminary work described here is the English-German language pair, from the [WMT23](#) challenge [181], because it is the language pair with the highest number of parallel sentences in the [WMT](#) datasets.

The data released for the [WMT23](#) General [Machine Translation](#) task can be freely used for research purposes. As the CycleGAN framework is computationally expensive, only 10% of the English-German dataset was used, comprising 27,801,496 sentences.

4.1.4 Tokenizer Training

One of the determining parameters of a [Natural Language Processing](#) model is the maximum size of sentences that the model will accept, that is, the number of tokens that can be taken into account per sentence. Although it is possible to count the number of characters rather than the number of tokens, since the sentences will be tokenised and it is only in tokenized form that they will be presented to the model, the size or length of the sentences here means the number of tokens to which they are converted.

This parameter, here referred to as *max_length*, is a trade-off between the computational cost of the network and the number of incomplete sentences that can be accepted. This parameter determines the size of the “position embeddings”, *position_embed*, which is an essential component of the Transformer architecture. In distinction to [RNNs](#) and [LSTMs](#), Transformers process input data simultaneously. Without these position embeddings, a Transformer interprets sentences as collections of words, disregarding the order in which the words appear, and of course, order is crucial to comprehend a language so it is important not to lose it.

For the dataset exploration described here, two Unigram tokenizers were trained using the SentencePiece library¹ provided by Google, on the first 100,000 sentences of each language, with a vocabulary of 16,000 words.

4.1.5 Maximum Length of Sentences

When the number of tokens making up a sentence is less than *max_length*, the `<pad>` token will be used to pad the sentence. Thus, with a *max_length* of 16, the sentence

From ancient grudge break to new mutiny

is first tokenized to:

`_From _ancient _g ru d ge _break _to _new _mu tin y </s>`

Since this sentence only contains 13 tokens, it is then padded to:

`_From _ancient _g ru d ge _break _to _new _mu tin y </s> <pad> <pad>
<pad>`

However, with a *max_length* that is too short, the sentence would have been truncated and information would have been lost. In fact, with a *max_length* equal to 8, we would have obtained:

`_From _ancient _g ru d ge _break </s>`

¹<https://github.com/google/sentencepiece>

In order to choose a suitable *max_length* that is not too costly while still allowing very few sentences to be truncated, we also need to consider the dataset. Depending on the nature of the dataset, i.e. the linguistic domain of the sentences that make it up, but also the language, the size may vary. In order to adapt this parameter to the dataset used here, the *max_length* will vary between 2 and 1024, and by counting the presence or absence of a <pad> token for each sentence, we can deduce whether or not it has been truncated. The following snippet of code calculates the percentage of untruncated sentences depending on the *max_length* parameter on the first 100,000 sentences of the English dataset:

```
BATCH_SIZE = 100
BATCH_NUMBER = 100_000/BATCH_SIZE
for max_length in [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]:
    number_of_incomplete_sentences = 0
    for i, (input_ids, _) in enumerate(train_dataloader):
        if i == BATCH_NUMBER:
            break
        input_ids = input_ids[:, :max_length]
        number_of_incomplete_sentences += int((input_ids == config.
pad_token_id).any(dim=1).sum())
    print(f"{max_length=}: {(number_of_incomplete_sentences/(
BATCH_SIZE*BATCH_NUMBER))*100}%")
```

CODE LISTING 4.1: Python code measuring the number of truncated sentences in a dataset depending on the *max_length* of the model

As expected, a larger *max_length* results in a lower number of truncated sentences, as shown in Table 4.4. The difference in the percentage of non-truncated sentences between the two languages is attributed to the fact that, in this sample, the average English sentence contains only 123 characters, compared to 143 characters for the German sentence. As shown in 4.1 and 4.2, the token distributions are skewed, and the median reveals additional information for this distribution. The English sentences have a median of 109 characters, while the German ones have a median of 126. Since this is a parallel sample, each pair of sentences tends to convey the same amount of information, assuming the translations are high-fidelity. As noted in [208], “German writers tend to produce longer sentences”. The observed length difference likely reflects linguistic differences rather than content disparities.

A *max_length* value of 128 strikes a practical balance between retaining the vast majority of sentences and managing computational efficiency. A value of 64 would exclude too much of the dataset to ensure adequate coverage. On the other hand, higher *max_length* values, such as 256, 512 or 1024, would capture nearly all sentences but come with significantly higher computational costs for little added benefit. Table 4.5 presents the maximum batch size supported by the available hardware, which has 24 GiB (GiB, or gibibyte, is a binary multiple of bytes, where 1 GiB equals 2^{30} bytes [1]) of memory, as

TABLE 4.4: Impact of the maximum sentence length on the percentage of untruncated sentences

Max_length	English untruncated sentences (%)	German untruncated sentences (%)
2	0.000	0.000
4	0.232	0.270
8	7.085	5.522
16	35.953	31.911
32	77.573	73.034
64	97.745	96.769
128	99.899	99.875
256	99.992	99.991
512	99.997	99.994
1024	100.000	100.000

a function of the *max_length* value when training two default Transformer models from the original implementation [359] simultaneously.

By choosing 128, 99.899% of the English sentences and 99.875% of the German sentences remain untruncated while still allowing for a batch size which allows for training a single epoch on the chosen dataset in less than a week, and thus this value was chosen for this work.

TABLE 4.5: Impact of the maximum sentence length on the maximum batch size

Max_length	Maximum batch size
32	160
64	76
128	32
256	16
512	8
1024	2

4.1.6 Differences Between the English and German Corpora

The differences between the two corpora in terms of the number of tokens per sentence are minimal, as can also be observed from the distribution of the number of tokens per sentence in each of these two languages. Figures 4.1 and 4.2, respectively, display the histograms of the English and German sentences. The mean number of tokens is quite similar (22.37 and 24.39), as is the standard deviation (16.10 and 17.58).

Understanding token distributions informs model design decisions such as embedding sizes, which in turn have an effect on training time and memory requirements. However, the model is not determined by a single parameter as many options can influence the architecture of a Transformer model.

By specifying the structure of a model in [Machine Learning](#), i.e., the number of layers, their organisation, and certain key parameters, we determine not only the model’s ability to learn, but also its computational cost. Furthermore, depending on the task at hand, some architectures are more efficient than others. As mentioned in Section 2.3, Transformer models are particularly well suited for [Natural Language Processing](#).

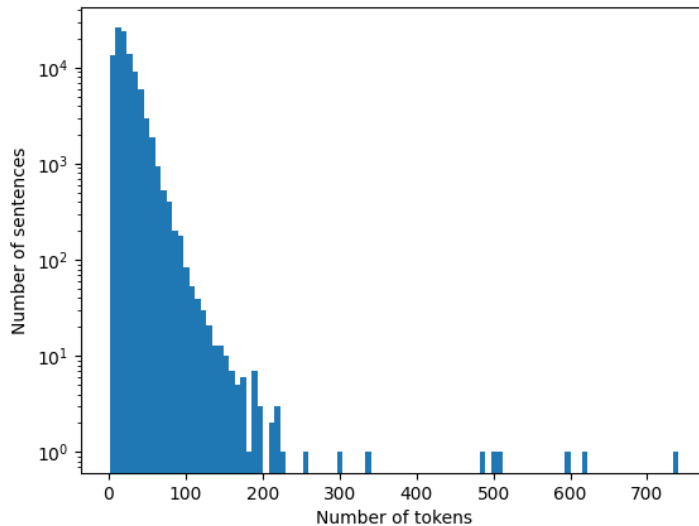


FIGURE 4.1: Distribution of the number of tokens in the English dataset

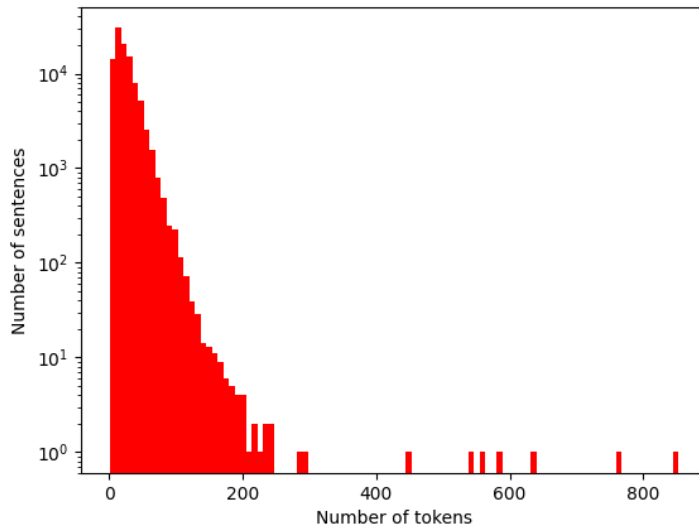


FIGURE 4.2: Distribution of the number of tokens in the German dataset

The HuggingFace Transformer library [391] includes implementations of numerous architectures in Python and was therefore chosen for prototyping. Based on PyTorch [265], it enables low-level modifications to be made so that the behaviour and operation of models can be easily changed.

4.1.7 Generator

MarianMT [166] was found by the author to be an excellent Transformer implementation that can be trained on consumer-grade hardware, and it is a highly flexible HuggingFace implementation that makes it easy to modify.

The work of the University of Helsinki [349], on trained Marian models in the HuggingFace library, which are licenced under the Apache licence, provided significant guidance on the selection of model parameters. These models were used as guidelines for the parameters of the models, in particular the size of the vocabulary, the number of layers and their dimensions, and the choice of tokenizer.

The model hyperparameters were also adapted to the hardware used, namely a NVidia 4090, and in particular to its amount of video memory, i.e., 24GiB. Since the CycleGAN architecture requires that two models be trained at the same time, the amount of memory available restricted the number of parameters for each generator. Thus, the context window, i.e., the number of tokens to which the model has access at a given time, was set to 128 tokens to satisfy the context window constraint, as explained in Section 4.1.5. Sentences that are any longer than this were systematically truncated to the first 128 tokens, and translations of these sentences cannot exceed this size either.

The internal dimension of the generator model was set at 1024. It is at the same time the dimension of the latent vector located between the encoder and the decoder, the dimensions of the four linear layers located in each attention heads and the final projection layer. There were six encoder layers, each of which has eight attention heads, and an internal projection layer of dimension 2048. The composition of the encoder layer is shown in Figure 4.3.

```
MarianEncoderLayer(
  (self_attn): MarianAttention(
    (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
  )
  (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  (activation_fn): ReLU()
  (fc1): Linear(in_features=1024, out_features=2048, bias=True)
  (fc2): Linear(in_features=2048, out_features=1024, bias=True)
  (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
)
```

FIGURE 4.3: Architecture of a Marian Encoder block as displayed in the HuggingFace library

The decoder has the same number of layers and the same dimensions. Each layer is organised as shown in Figure 4.4.

The dimension of the position embeddings is equal to the maximum length accepted by the model, which is equal to the *max_length* value of the tokenizer. The vocabulary size is set at 32,000, which is less than the 58,101 used in the Helsinki models, but corresponds to recent findings [115]. Table 4.6 shows how a larger vocabulary of 58,101 tokens does not reduce the number of out-of-vocabulary words in the English dataset, it reduces the number of occurrences in the German dataset. This absence of difference observed in the case of German can be explained by the presence of words in the test set using characters which are not present in the training set. This means that the tokenizer does not benefit from a larger vocabulary with the training set used.

```

MarianDecoderLayer(
  (self_attn): MarianAttention(
    (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
  )
  (activation_fn): ReLU()
  (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  (encoder_attn): MarianAttention(
    (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
    (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
  )
  (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  (fc1): Linear(in_features=1024, out_features=2048, bias=True)
  (fc2): Linear(in_features=2048, out_features=1024, bias=True)
  (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
)

```

FIGURE 4.4: Architecture of a Marian Decoder block as displayed in the HuggingFace library

TABLE 4.6: Impact of the size of the vocabulary on the number of unknown tokens

Language	<unk> tokens, vocabulary size 32,000 (%)	<unk> tokens, vocabulary size 58,101
English	1.44	1.44
German	2.91	1.98

Table 4.7 shows examples of German sentences in which no out-of-vocabulary errors arise from a vocabulary size of 58,101 but exist with a smaller vocabulary of 32,000. For all the cases observed by the author, the presence of a token <unk> was due to a character that does not exist in the German language. Although the one percentage point reduction in the number of sentences in German that contain at least one token <unk> is considerable, the CycleGAN architecture requires generators to have the same number of parameters and the same architecture to balance their training, knowing that the framework is inherently fragile, as highlighted in Section 2.6.4. Therefore, since a vocabulary of 58,101 would only reduce out-of-vocabulary errors for one of the two languages, a more contained size of 32,000 was chosen.

Moreover, using a smaller vocabulary allows for a bigger batch size on the GPU provided for this work. Although the maximum batch size using a vocabulary of 58,101 is 12, the smaller vocabulary of 32,000 allows for a batch size of 32, which makes training faster. Finally, the projection head is a linear layer of dimensions $1024 \times 32,000$.

TABLE 4.7: Examples of German sentences where a larger vocabulary size prevents out-of-vocabulary errors

Vocabulary size 32,000	Vocabulary size 58,101
“Hotel CaSerena in Sirmione”	“Hotel Ca‘Serena in Sirmione”
“IS THE htel SITUATED?”	“IS THE hôtel SITUATED?”
“San Don di Piave-Jesolo”	“San Donà di Piave-Jesolo”

These parameters are directly inspired by the English to German model that the University of Helsinki uploaded for this language pair. Table 4.8 references the changes that were made in order to reduce the computational cost of the architecture. Indeed, while

the MarianMT models trained by the University of Helsinki are independent and can be trained separately, the architecture of the current work requires simultaneous training of two models. This constraint reduces the number of parameters per model to ensure they fit within the GPU memory.

TABLE 4.8: Non-default parameters in the configuration of Marian Transformer models

Parameter	Helsinki	Current work
Vocabulary size	58,101	32,000
Encoder layers	12	6
Decoder layers	12	6
Encoder attention heads	16	8
Decoder attention heads	16	8
Encoder feed-forward	4096	2048
Decoder feed-forward	4096	2048
Position embeddings	1024	128
Activation function	GELU	ReLU

The `count_parameters` [23] function is used to determine the number of trainable parameters and estimates this number to be 158,769,152 using this configuration.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.
requires_grad)
```

CODE LISTING 4.2: PyTorch function that returns the number of trainable parameters in a model

4.1.8 Discriminator

Although the numerous works published on [GANs](#) focus mainly on images, there is also a large academic background on the use of [GANs](#) for text generation. Different architectures such as [Deep Neural Network \(DNN\)s](#) [362], [CNNs](#) [174] and [Recurrent CNNs](#) [192] have been successfully used for Text Classification. Both a [CNN](#) classifier [425] and a transformer-based classifier were used as discriminators. Nevertheless, before defining these two architectures, we need to answer a question that is common to all Discriminator implementations: output dimension.

Since the role of the Discriminator in a [GAN](#) is to separate the dataset data from the generated data, the output dimension can be configured in two different ways: a single output, which for example indicates *real* when it is 0 and *fake* when it is 1, or two outputs, each corresponding to the probabilities of the input being real or fake, respectively.

The dual output implementation is similar to a binary classification model. Using a softmax activation function in the last layer, the model produces a two-dimensional vector for each sample, where each element represents the probability of its class and the total probability is equal to 1. Multiple outputs are also necessary in the specific case of Conditional [GANs](#) [236] [249], although using the number of classes as the dimension of

the output, where the objective of the discriminator is not simply to determine whether the sample is *real* or *fake*, but to determine the class of the generated sample.

Using a single output is the traditional approach, used in particular by the creators of the GAN architecture [112]. It is a simpler implementation, the final activation function of which is a sigmoid that compresses the output between 0 and 1. For prototyping purposes and to simplify the model as much as possible, this implementation has been chosen.

For each of the prototypes, two different discriminator architectures were tested. The first largely shares the architecture of the generators, whereas the second, which is less expensive, has already been tried as a classification architecture in previous work.

Transformer Discriminator

The transformer discriminator is an encoder-only transformer model, followed by a linear layer. As described in Section 2.4.3, BERT constitutes an encoder-only architecture that demonstrates exceptional proficiency in tasks related to text classification.

The main advantage of using a transformer model for the discriminator is that the same configuration can be used for both the Discriminator and the Generator. This allows the number of parameters to be approximately balanced between the two models. In Progressive GANs [170], the Discriminator and Generator are described as “mirror images” and almost contain exactly the same number of parameters. BigGAN-deep [39] relies on a smaller number of parameters for the Discriminator than for the Generator: 34.6M and 50.4M parameters, respectively, that is, a ratio of approximately $\frac{2}{3}$.

Using the `count_parameters` function defined in Section 4.1.3, the number of trainable parameters in the current Transformer discriminator is 100,109,313, which is also a ratio of approximately $\frac{2}{3}$ with the generator, similar to the previous BigGAN-deep work.

Convolutional Discriminator

Text generation models such as SeqGAN [414], LeakGAN [119], CatGAN [215] and TextGAN [427] all rely on the CNN architecture identified by [174] and [65] for their discriminators. This architecture has been proven to be effective as a text classifier and also as a Discriminator for textual GANs, so it was chosen for this work, as implemented by TextGAN-PyTorch².

The discriminator employs multiple convolutional layers that incorporate multiple filters of varying sizes to detect relationships across different word lengths. Additionally, it uses a maximum pooling layer applied across the entire input sentence for each feature map to extract important features. Figure 4.5 displays the architecture of the Discriminator, where the filters were obtained from SeqGAN. According to the `count_parameters`

²<https://github.com/williamSYSU/TextGAN-PyTorch>

function defined in Section 4.1.3, this discriminator possesses 17,557,282 trainable parameters, which only represents about 11% of the size of the generator.

```

CNNDiscriminator(
  (embeddings): Embedding(16000, 512, padding_idx=1)
  (convs): ModuleList(
    (0): Conv2d(1, 100, kernel_size=(1, 512), stride=(1, 1))
    (1): Conv2d(1, 200, kernel_size=(2, 512), stride=(1, 1))
    (2): Conv2d(1, 200, kernel_size=(3, 512), stride=(1, 1))
    (3): Conv2d(1, 200, kernel_size=(4, 512), stride=(1, 1))
    (4): Conv2d(1, 200, kernel_size=(5, 512), stride=(1, 1))
    (5): Conv2d(1, 100, kernel_size=(6, 512), stride=(1, 1))
    (6): Conv2d(1, 100, kernel_size=(7, 512), stride=(1, 1))
    (7): Conv2d(1, 100, kernel_size=(8, 512), stride=(1, 1))
    (8): Conv2d(1, 100, kernel_size=(9, 512), stride=(1, 1))
    (9): Conv2d(1, 100, kernel_size=(10, 512), stride=(1, 1))
    (10): Conv2d(1, 160, kernel_size=(15, 512), stride=(1, 1))
    (11): Conv2d(1, 160, kernel_size=(20, 512), stride=(1, 1))
  )
  (highway): Linear(in_features=1720, out_features=1720, bias=True)
  (feature2out): Linear(in_features=1720, out_features=1, bias=True)
  (dropout): Dropout(p=0.25, inplace=False)
)

```

FIGURE 4.5: Architecture of the CNN Discriminator

4.1.9 Data types

The dataset contains sentences encoded in UTF-8 separated by the Unix newline character `\n`. The data loader loads *batch_size* sentences at a time that are then transformed by the tokenization process. The tokenizer converts each sentence into a vector of strings corresponding to the learnt tokens and then converts this vector of strings into a vector of integers, where each integer is in the range $[0, vocab\_size - 1]$ and corresponds to the unique ID of the token. The vector is then truncated or padded to reach a length of *position_embed* tokens. The vectors are then merged into *input_ids*, a 2D vector of dimensions $(batch\_size, position\_embed)$.

The Embedding layer transforms *input_ids* into *embeddings*, a 3D vector of floating point values of shape $(batch\_size, position\_embed, d\_model)$. This shape is kept by the encoder to create *encoded* from *embeddings*, and the decoder to generate *decoded* from *encoded*. The linear projection layer then transforms this vector into *logits*, a 3D vector of floating point values of shape $(batch\_size, position\_embed, vocab\_size)$. Finally, by taking the indices of the maximum value of all elements in this vector along the *vocab_size* dimension, *output_ids*, a 2D vector of integers of shape $(batch\_size, position\_embed)$ where integers are in the range $[0, vocab\_size - 1]$ is created, which can be transformed back into *batch_size* sentences by the tokenizer.

For the sake of clarity, the list below shows the dimensions and numbers set at each stage of generation by the Transformer architecture.

- $input\_ids \in \mathbb{N}^{batch\_size, position\_embed} | \forall x \in input\_ids, x \in [0, vocab\_size[;$
- $embeddings \in \mathbb{R}^{batch\_size, position\_embed, d\_model},$

- $encoded \in \mathbb{R}^{batch\_size, position\_embed, d\_model}$,
- $decoded \in \mathbb{R}^{batch\_size, position\_embed, d\_model}$,
- $logits \in \mathbb{R}^{batch\_size, position\_embed, vocab\_size}$,
- $output\_ids \in \mathbb{N}^{batch\_size, position\_embed} | \forall x \in input\_ids, x \in [0, vocab\_size[$.

While floating-point numbers, as defined by the IEEE 754 standard [250], are often used to approximate real numbers, they do not fully represent $\mathbb{R}$, the set of all real numbers. This is due to their limited precision and the presence of special values such as positive and negative infinity, which represent unbounded values rather than actual real numbers. Additionally, IEEE 754 defines NaN (Not a Number) to indicate undefined or invalid operations, which does not belong to $\mathbb{R}$.

4.1.10 Reproducible Build

A reproducible model in machine learning is defined as a model that can consistently achieve identical results when re-run multiple times with the same data, hardware, and configuration. According to the U.S. National Science Foundation (NSF) Subcommittee on Replicability in Science, “reproducibility refers to the ability of a researcher to duplicate the results of a previous study using the same materials as were used by the original investigator” [435].

This ensures that the results can be independently verified by others in order to guarantee the reliability of the work. Moreover, reproducible builds allow benchmarks to be more consistent in order to compare models.

As the dataset used for this work, described in Section 4.1.3, is public and kept identical throughout the entire prototyping, the data consistency aspect of reproducibility is ensured. The other main aspects are model configurations and the uniformity of the environment.

Randomness

Although the configurations of the models are described in Section 4.1.3, a crucial element for reproducibility is missing: the seeds. In [Machine Learning](#), models are usually initialised with random weights, that are obtained through a [Random Number Generator \(RNG\)](#). It is crucial to note that some “random” sequences are more random than others, depending on the [RNG](#) used. Cryptographic random numbers should be unpredictable, even when the algorithm is known and previous generated numbers are available. These generators typically rely on physical devices such as mouse movements, storage disk timings, and even electrical noise measured by the CPU itself³ [123] to collect entropy⁴.

³<https://github.com/torvalds/linux/blob/master/drivers/char/random.c>

⁴https://wiki.archlinux.org/title/Random_number_generation

Pseudorandom Number Generator (PRNG)s are algorithms that simulate randomness and can generate sequences that cannot be reasonably predicted better than by chance. Although these random numbers are not suitable for cryptography-related tasks, they are crucial in cases such as simulation, shuffling, sampling, or exponential backoff [72]. They are deterministic algorithms that are initialised with a seed, which is a crucial step since the same seed always produces the same sequences. In order to obtain a reproducible model, the seed is set in the configuration. The number “4512540”, used as a seed in this work, was arbitrarily chosen.

The PyTorch documentation⁵ includes a section dedicated to reproducibility and includes multiple seed sources that should be set:

- The Python language includes a module to generate random sequences⁶ and as such, the seed has to be determined;
- The NumPy is also used in many libraries and the global seed has to be set;
- The PyTorch seed is directly responsible for the initialisation of models.

```
import random, numpy as np, torch
random.seed(4512540)
np.random.seed(4512540)
torch.manual_seed(4512540)
```

CODE LISTING 4.3: Python code used to set all seeds for random number generation

Dependencies and Versioning

As stated in the documentation, “Completely reproducible results are not guaranteed across PyTorch releases, individual commits, or different platforms”. Dependency pinning is the practice of specifying the exact versions of libraries on which software builds depend. This is in contrast to allowing the newest versions to be used automatically. Pinning the dependencies is crucial in ensuring reproducible builds, and it helps in obtaining consistent behaviours.

Insofar as Python relies on the *getrandom()* syscall, which relies on the special file */dev/urandom*, to obtain random numbers on GNU/Linux systems, the kernel version can also influence the **PRNG** algorithm and must also be pinned. PyTorch also relies on */dev/urandom* to obtain random numbers⁷. Similarly, the CUDA library is frequently updated with no guarantee of backward compatibility, and its version must also be pinned. Finally, the Python language itself is sometimes updated in a way that is not backward compatible, and the exact version used must also be pinned.

⁵<https://pytorch.org/docs/stable/notes/randomness.html>

⁶<https://docs.python.org/3/library/random.html>

⁷<https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/CPUGeneratorImpl.cpp>

In Python, the “requirements.txt” file can be generated using “pip freeze” and used to install the specified version of each library, ensuring consistency between multiple deployments and installations.

Docker [232] is a containerisation tool, built on cgroups and namespaces, that is frequently used to help the reproducibility of the work. It allows the maintainer to specify the exact configuration required, which is then used to build a container, a mechanism analogous to a virtual machine, with all the specified components. However, this does not take into account hardware modifications, or potential changes to the kernel, because unlike virtual machines, containers are based on the host kernel. As this would appear to be the tool most frequently used for reproducibility in [Machine Learning](#), we published a Dockerfile.

4.2 Early Work

Since, in a permuted dataset, every sentence is guaranteed to have a corresponding translation, whereas in a non-intersecting dataset, no sentence has a direct translation, one of the key assumptions in this work was that training on a permuted dataset should be “easier”.

One way to define “easier” is that, all other factors being equal, a model trained on a permuted dataset should produce higher-quality translations compared to a model trained on a non-intersecting dataset.

Another advantage of using permuted datasets is that it makes comparisons with parallel models more fair, as it is possible to compare two models trained on the exact same sentences, all other things being equal.

Thus, all of the initial work was carried out on permuted datasets with a view to rapid prototyping.

4.2.1 Baseline Results

The [COMET](#) score, discussed in Section 2.5.2, is the state-of-the-art evaluation technique for [Neural Machine Translation](#). Section 4.4.7 provides a detailed explanation of why this score was chosen over the other metrics, particularly in relation to metrics based on the overlap of n -grams, such as [BLEU](#), chrF++ or Tokengram_F.

Before training an unsupervised model on non-parallel data, it is advisable to obtain results that would have been obtained if the data had been parallel and the training supervised. The test set used to obtain the [COMET](#) score comes from the [WMT23](#) source dataset. These are the same sentences on which the architecture-matched models, pre-trained models, and prototypes are tested.

Architecture-Matched

For comparison purposes, two architecture-matched baseline models were trained. These models use the exact configuration of the Generator and are trained on the same unpermuted dataset. In this way, the sentences on which the models have been trained will be the same.

It is anticipated that, for equal training, using a parallel dataset will enable a model to achieve better performance, since the amount of information contained in a pair of sentences is greater than the amount of information contained in two separate unrelated sentences. The objective of this work is to get as close to the results obtained by the baseline models as possible, which are displayed in Table 4.9. As shown in the results of the WMT23 [181] in Tables 12 to 26, for each pair of languages, the score depends on the direction of the translation and therefore some variation is expected depending on the direction.

TABLE 4.9: COMET score of the baseline models

Language Pair	Score (COMET)
English $\rightarrow$ German	0.780
German $\rightarrow$ English	0.775

Pre-Trained Models

As part of the OPUS-MT project [349], the University of Helsinki has released many Neural Machine Translation Marian models for different pair of languages [251], including English to German and German to English. Although these models contain more parameters than the prototype configuration used in the author’s research work and were not trained on the same dataset, the results are still included as a comparison. Because these models are larger and trained on more data, it was expected that they would perform better than the models and prototypes matched to architecture, which is confirmed by the results obtained and available in Table 4.10.

TABLE 4.10: COMET score of the Helsinki models

Language Pair	Score (COMET)
English $\rightarrow$ German	0.789
German $\rightarrow$ English	0.791

4.2.2 Ideal implementation

The assumption here is that the original CycleGAN architecture can be perfectly transposed to text using transformer networks.

Generators and discriminators were initialised with random parameters. x and y were drawn from the X and Y distributions. $\hat{x}$ and $\hat{y}$ were obtained from $\hat{x} = F_{Y \rightarrow X}(y)$ and $\hat{y} = G_{X \rightarrow Y}(x)$. The recovered IDs were then calculated using the formulas $\hat{\hat{x}} = F_{Y \rightarrow X}(\hat{y})$ and $\hat{\hat{y}} = G_{X \rightarrow Y}(\hat{x})$. Finally, $id\_x = F_{Y \rightarrow X}(x)$ and $id\_y = G_{X \rightarrow Y}(y)$ are determined.

Generator training

The generators $\mathcal{G}$ and $\mathcal{F}$ are trained using different losses:

- The [Cycle-Consistency Loss](#) is computed between x and $\hat{x}$, and between y and $\hat{y}$;
- The [Identity Loss](#) is computed between x and id_x , and between y and id_y ;
- The adversarial loss is computed from $\hat{x}$ and $\hat{y}$ using the Discriminators D and E .

For clarity purposes, only one half of the process, that concerning x , is shown in Figure 4.6.

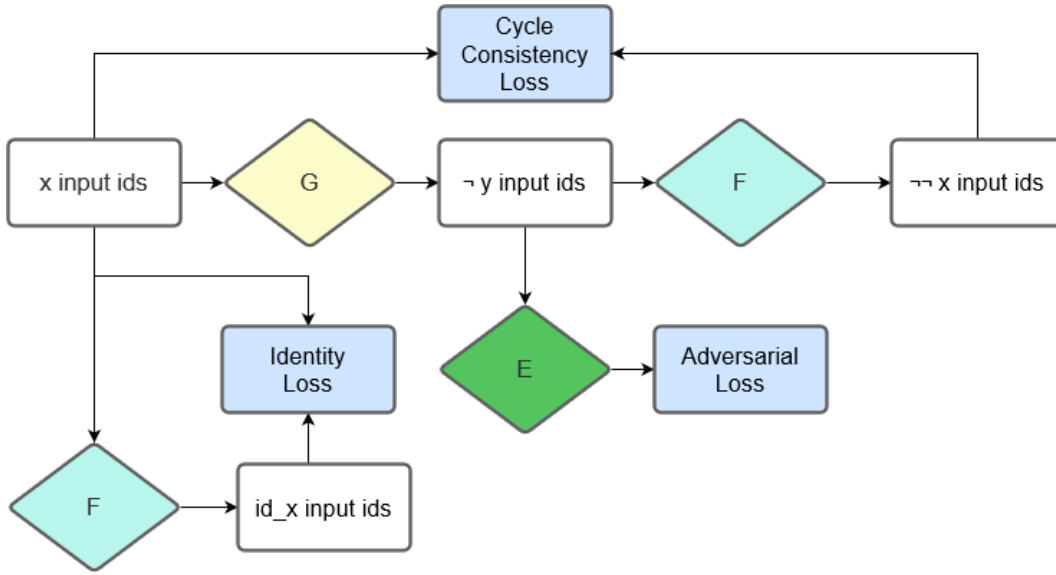


FIGURE 4.6: Generator training process, where $\neg y$ input ids refers to $\hat{y}$ and $\neg\neg x$ input ids refers to $\hat{x}$

Discriminator Training

The losses of the Discriminators D and E are computed as follows:

- D and E have to respectively predict that $\hat{x}$ and $\hat{y}$ are not original samples from the dataset;
- D and E have to respectively predict that x and y are original samples from the dataset.

The training process of the Discriminator E is shown in Figure 4.7.

The Necessity of Labels

Labels, that is, tokens of the target sentence, are not needed in inference, as requiring the translation of a sentence would defeat the purpose of [Neural Machine Translation](#),

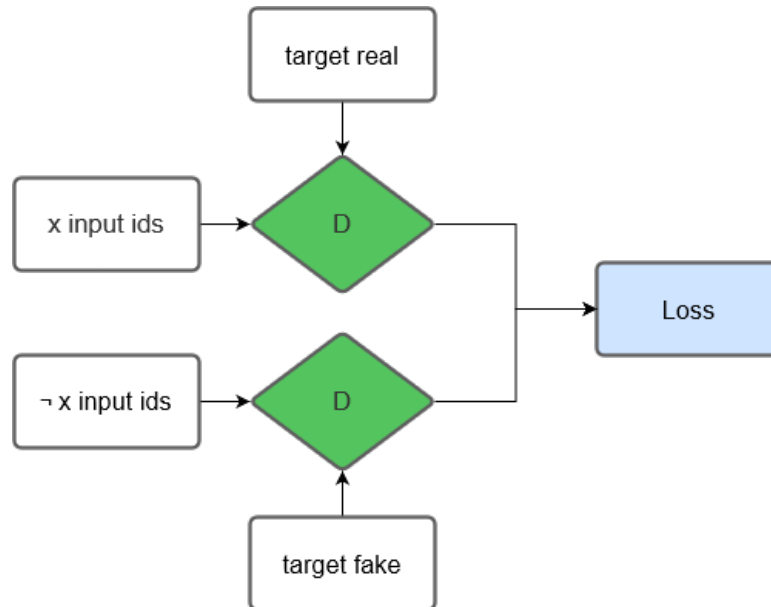


FIGURE 4.7: Discriminator training process

but they are needed during training. [seq2seq](#) is traditionally a supervised learning task. Consequently, the architectures developed for [Neural Machine Translation](#) expect to have the labels. Labels provide the specific results that the model must learn to produce, as in their absence, the model would have no precise target for the correct results.

The attention mechanisms in the [seq2seq](#) models are crucial to understand the connections between sentences of the source and target languages. The presence of labels allows the model to grasp these relationships. By aligning the inputs and their corresponding outputs, the model is trained to selectively focus on the relevant tokens of the input while generating each token of the generated translation. A consequence of teacher forcing, discussed in Section 2.3.2, is that this training technique can only be used when the output is known, that is, in supervised learning, as is usually the case when training for a translation task using a parallel dataset.

Thus, the present implementation is not consistent with [seq2seq](#) requirements for the Transformer architecture, as the generators require labels during training.

4.2.3 Naive Implementation

In the training of a Transformer, the necessity of labels poses a challenge when dealing with a permuted dataset as it lacks the exact labels corresponding to the inputs. However, one can still employ the use of approximate labels in such scenarios. In this context, an approximate label refers to a label that may not directly match the input but is still relevant and belongs to the appropriate domain.

Generator Training

Intermediate translations were first obtained by generating $\hat{x}$ using x as the label and $\hat{y}$ using y as the label.

Generating the end translation to complete the cycle is trivial, as the labels are already known. In fact, $\mathcal{F}$ produced $\hat{x}$ from $\hat{y}$ with x used as the label, because the objective is to revert to the original sentence. $\hat{y}$ was also generated by $\mathcal{G}$ from $\hat{x}$ using y as the label. Figure 4.8 displays only one-half of the training process.

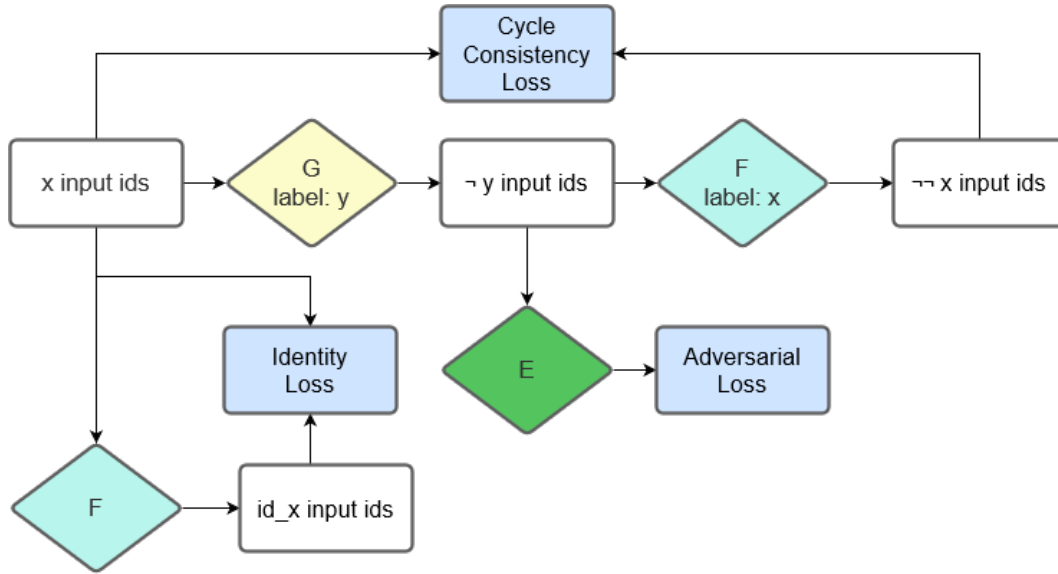


FIGURE 4.8: Generator training process

Discriminator Training

The training process for the discriminator does not differ from the ideal implementation discussed in Section 4.2.2.

Incorrect Mappings

With the present implementation, the learning stops very quickly, long before one manages to produce coherent translations that have a link with the source sentences. The fundamental aim of a [Neural Machine Translation](#) model is to learn the mappings from the source language domain to the target language domain. Providing incorrect labels leads the model to learn wrong associations, which degrades its ability to perform accurate translations.

4.2.4 Encoded Space

In a transformer encoder-decoder architecture for [seq2seq](#), only the decoder requires labels during training. The encoder first contains an embedding layer and then a predefined number of sequential identical blocks. Rather than working directly in the X and Y domains, i.e. in token representations of natural languages of the dataset domain and the domain obtained at the decoder output, this implementation aims to work in the encoder output domain.

Adding depth to an encoder means adding new blocks. Rather than adding new blocks to an encoder, we can instead chain together another encoder with the same configuration, which would be deprived of its initial embedding layer. Figure 4.9 displays the architecture of an encoder.

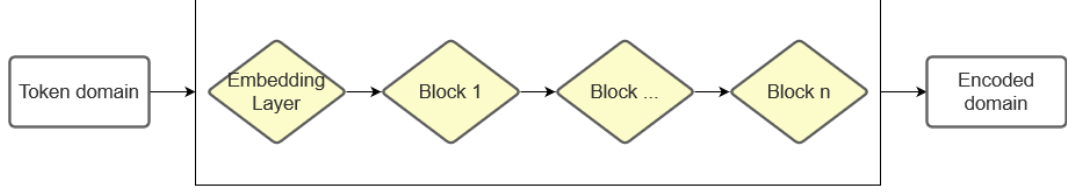


FIGURE 4.9: Single encoder architecture

Rather than using a single encoder, the implementation is based on using two in a row for each generator. The output of the first encoder would be the encoded space representation of the token source distribution. Figure 4.10 displays the representation of two chained encoders.

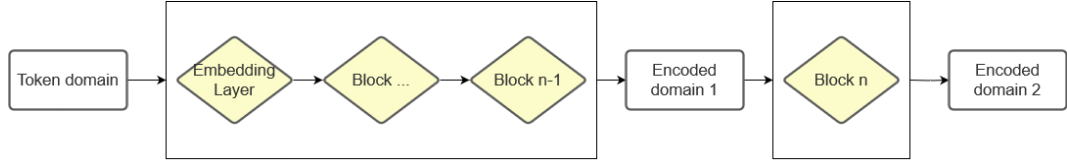


FIGURE 4.10: Dual encoder architecture

Let G_{enc} be the first encoder of $\mathcal{G}$, which maps X to zX , its encoded representation, and F_{enc} , the first encoder of $\mathcal{F}$, which maps Y to zY . The role of the second encoder is to learn the mapping between the two encoded distributions. G_{gen} , the second encoder of $\mathcal{G}$, learns the mapping between zX and zY while F_{gen} , the second encoder of $\mathcal{F}$, obtains the mapping between zY and zX .

The role of the decoders is to switch from the encoded domains to the natural language domains. G_{dec} learns the mapping between zY and Y while F_{dec} learns the mapping between zX and X .

In mathematical terms:

- $x \in X$;
- $y \in Y$;
- $G_{enc_{X \mapsto zX}}(x) = zx$, with $zx \in zX$;
- $F_{enc_{Y \mapsto zY}}(y) = zy$, with $zy \in zY$;
- $G_{gen_{zX \mapsto zY}}(zx) = \hat{zy}$, with $\hat{zy} \in zY$;
- $F_{gen_{zY \mapsto zX}}(zy) = \hat{zx}$, with $\hat{zx} \in zX$;
- $G_{gen_{zX \mapsto zY}}(\hat{zx}) = \hat{\hat{zy}}$, with $\hat{\hat{zy}} \in zY$;
- $F_{gen_{zY \mapsto zX}}(\hat{zy}) = \hat{\hat{zx}}$, with $\hat{\hat{zx}} \in zX$;

- $Gdec_{zY \rightarrow Y}(z\hat{y}) = \hat{y}$, with $\hat{y} \in Y$;
- $Fdec_{zX \rightarrow X}(z\hat{x}) = \hat{x}$, with $\hat{x} \in X$;

Generator Training

The training procedure for the Generators encompasses several extra steps. For the sake of simplification during the initial prototyping phase, the **Identity Loss** was omitted. However, this architecture allows for the inclusion of another **Cycle-Consistency Loss** at a different stage in the process. The initial application of **Cycle-Consistency Loss** still occurs between the input x and its corresponding output $\hat{x}$, while the subsequent application takes place between zx and $z\hat{x}$.

The addition of this second **Cycle-Consistency Loss** enables the second encoders $Ggen$ and $Fgen$ to learn the mapping between zX and zY , and between zY and zX respectively. Figure 4.11 displays the generator training process $\mathcal{G}$.

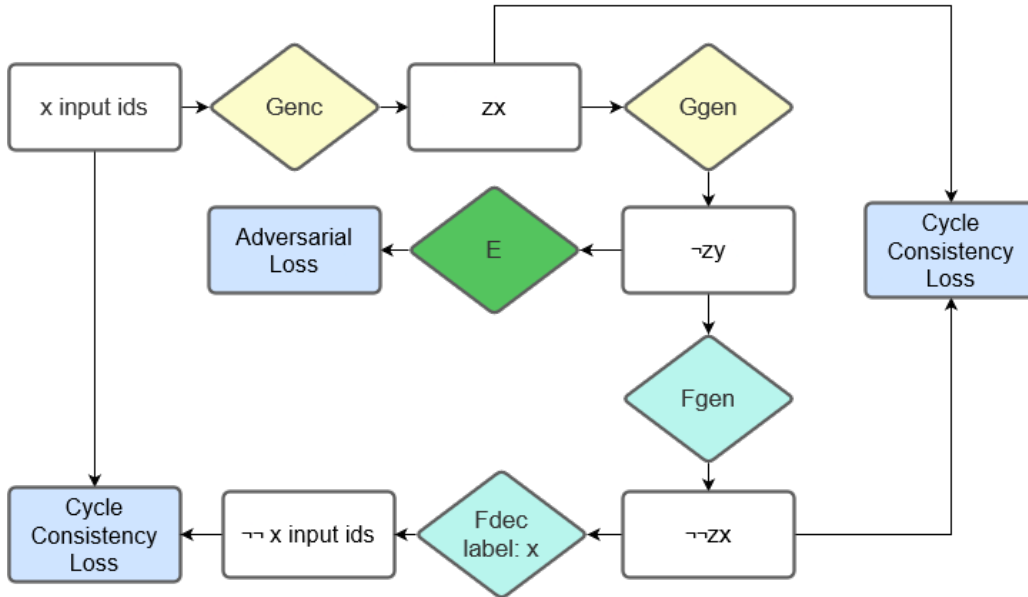


FIGURE 4.11: Generator training process. The encoder of $\mathcal{G}$ encodes x into zx . zx is then used as the input of the generator of $\mathcal{G}$ to obtain zy . The generator of $\mathcal{F}$ transforms zy into $z\hat{x}$ which is decoded by $\mathcal{F}$ into $\hat{x}$. The similarities between x and $\hat{x}$ on one side, and zx and $z\hat{x}$ on the other, are used as loss functions. Another loss function is obtained through the discriminator $\mathcal{E}$. This process is repeated with y as inputs but was not represented for the sake of clarity

Discriminator Training

In the basic setup, the purpose of the Discriminators was to assess if the outputs from each Generator were part of the appropriate natural language. However, in this implementation, where the translation occurs in the encoded space between zX and zY , the Discriminators must perform their verification at this encoded level. Therefore:

- D and E have to respectively predict that $z\hat{x}$ and $z\hat{y}$ are not generated by the first encoders of the Generators but have been generated by the second;

- D and E have to respectively predict that zx and zy are generated by the first encoders and not the second.

This process is shown in Figure 4.12.

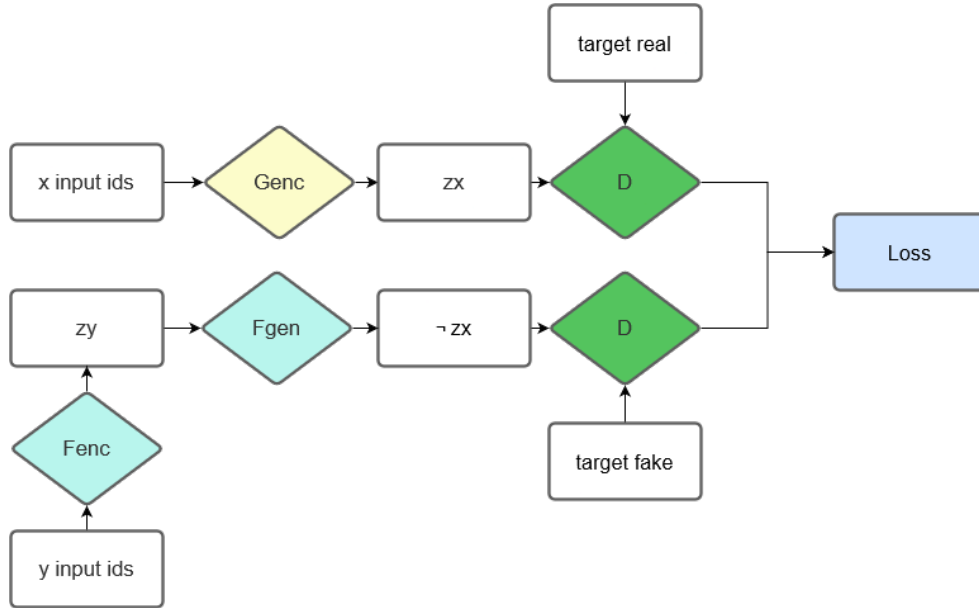


FIGURE 4.12: Discriminator training process. $\mathcal{D}$ takes as input zx , that is to say the output of the encoder of $\mathcal{G}$ given x and learns to classify it as a “real” sample. On the other hand, it learns to classify as a “fake” sample $\hat{zx}$, which is obtained from the generator of $\mathcal{F}$ given zy , itself obtained through the encoder of $\mathcal{F}$ given y . This training process is repeated for the discriminator $\mathcal{E}$ but is not shown for the sake of clarity.

A Varying Truth

In standard GAN architectures, the role of discriminators is to distinguish between what comes from the dataset, the “ground truth”, and what is synthetically generated by the generators, the “fakes”. As the dataset is static and does not change during training, the distribution remains identical from start to finish.

However, in this implementation, the ground truth comes from the first G_{enc} and F_{enc} encoders. As these generators learn throughout the training, the encoded version of the same input varies according to the training. As a result, the distribution of ground truth will shift during training, which prevents discriminators from learning to distinguish between ground truth and fakes.

As the discriminators are unable to compare the two changing distributions, their feedback is useless or even counterproductive, and the training of the framework ceases after a few batches.

4.2.5 Pre-Trained Compressed Space

For the Discriminators to be able to give correct feedback to the Generators, the ground truth must remain stable from start to finish, which means that the same input will

always be encoded from start to finish by the same encoded representation. To achieve this, the Generators' first encoders, G_{enc} and F_{enc} , are extracted and pre-trained on another task, and their weights are frozen while the framework is being trained.

Pre-Training

The aim of pre-training is both to have a stable ground truth distribution and to extract information from the input. In fact, while it would be sufficient to use random frozen parameters without further training to obtain a stable encoded representation of the input, using matrices with random parameters would lead to a massive loss of information, making it difficult to obtain a mapping between X and Y .

By pre-training the two encoders G_{enc} and F_{enc} , the information can be retained and freezing the weights will enable a stable ground truth to be obtained. To achieve this results, G_{enc} and F_{enc} were pre-trained as part of two VAEs, a category of AE discussed in Section 2.6.2. AE are a classic self-supervised learning architecture in which compression occurs between the encoder and decoder, forcing the whole model to learn how to reconstruct the input. The advantage of using a VAE rather than an AE is that it is a generative architecture. Thus, if the Discriminators ever need additional training data, rather than obtaining a larger dataset, it will be possible to generate synthetic examples of ground truth.

Since we do not have a sentence-aligned dataset, we do not have any labels for the inputs. Using unrelated labels as input, as would be the case when training directly from an unparallelled dataset, does not teach anything to the network. As there is no link between the input and its label, the parameters can only converge towards pseudo-randomness.

The present implementation required the pre-training of two VAE transformer models, $X2X$ and $Y2Y$, that each contained an encoder, a compressor and a decoder such that $X2X$'s encoder first mapped x to zx like a standard transformer, the compressor mapped zx to a lower dimension representation $z zx$ and the decoder then reconstructed $z zx$ to obtain x . Similarly, the encoder of $Y2Y$ mapped y to zy , its compressor zy to $z zy$ and its decoder $z zy$ to y . Once training was complete, the weights of the encoder of $X2X$ were transferred to G_{enc} and the weights of the encoder of $Y2Y$ were transferred to F_{enc} .

The training process of $X2X$ is schematised in Figure 4.13.

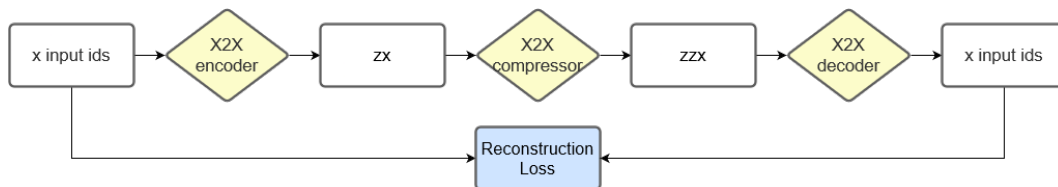


FIGURE 4.13: Pre-training process

Apart from this additional pre-training step for G_{enc} and F_{enc} , the rest of the training for the Generators and the Discriminators remained identical to the encoded space implementation.

Indistinguishability Between Ground Truth and Fake

Despite pre-training, which enables the creation of a stable ground truth distribution and allows for the generation of synthetic examples using $X2X$ and $Y2Y$ VAEs, the Discriminators consistently fail to reliably distinguish between real ground truth data and synthetic outputs. This occurs even though the synthetic outputs remain far from accurately representing a natural language in their encoded form.

As Discriminators are no better than a coin toss at determining whether the data presented is synthetic or not, feedback does not allow Generators to learn the mappings between X and Y .

4.2.6 GPT2 with Language Modeling Head

Although the subsequent GPT3 and GPT4 models have not been open sourced by OpenAI, the source code of GPT2 [287] is public and there is an HuggingFace implementation. Furthermore, HuggingFace implemented a variant with a Language Modeling head, giving sequence generation capabilities to this decoder-only model.

In a traditional encoder-decoder architecture, the encoder is responsible for transforming the input language text into a vector representation that enables the decoder to transcribe it into the target language. The absence of an encoder appears to make it impossible to map one language to another, from a source token distribution to a target distribution. Consequently, even with a parallel dataset, this model does not show the ability to learn text translation.

4.2.7 BERT Generation

Although, as mentioned in Section 2.4.3, BERT is an encoder-only architecture, it has also been adapted as an encoder-decoder seq2seq model [303] and has demonstrated excellent capabilities in Machine Translation tasks. A notable example of such an adaptation is BART (Bidirectional and Auto-Regressive Transformer) [194], which extends BERT by incorporating a seq2seq structure with a denoising autoencoder pretraining objective. BART is particularly effective in translation tasks because it learns to reconstruct corrupted text, making it suited for noisy or low-resource translation settings. While the GPT model with a language modeling head failed to acquire the ability to translate text even when trained on parallel datasets, the BERT model achieves translation capability, even when weights are randomly initialised rather than loaded from a pre-trained checkpoint.

However, when it comes to using non-parallel datasets, the present architecture collapses, with the total loss continually increasing to a maximum, regardless of the coefficients applied to the different loss functions. This maximum is reached when the output mirrors the input, and the model does not apply transformation to the text it receives.

4.2.8 BERT and GPT2

HuggingFace includes an `EncoderDecoderModel` class that allows for the combination of radically different model architectures. In this case, it is used to create a model that employs [BERT](#) as the encoder and [GPT2](#) as the decoder, trying to leverage the advantages of both architectures.

Tokenization

Given that the [BERT](#) and [GPT2](#) models employ different tokenization algorithms, the tokenization process is more complex than when using a single architecture. Specifically, the model must receive tokens segmented with a tokenizer suited to [BERT](#) and output tokens compatible with the [GPT2](#) tokenizer. Consequently, four tokenizers are trained:

- A [BERT](#) tokenizer to create the input tokens X_{bert} for $\mathcal{G}$, trained on an English dataset;
- another [BERT](#) tokenizer to create the input tokens Y_{bert} for $\mathcal{F}$, which is trained on a German dataset;
- a [GPT2](#) tokenizer to create the labels Y_{gpt} for $\mathcal{G}$, and is trained on a German dataset;
- finally, a [GPT2](#) tokenizer used to generate the labels X_{gpt} for $\mathcal{F}$, trained on an English dataset.

In this scenario, the mappings differ, as each language has two distinct sets of tokens rather than one. Thus, the mappings are now as follows:

- $G_{X_{bert} \mapsto Y_{gpt}}$;
- $F_{Y_{bert} \mapsto X_{gpt}}$.

Identifier Overlap

Since each language has two tokenizers, the number of tokens for the same sentence varies depending on whether it is in the input or output, and for tokens common to both the [BERT](#) and [GPT2](#) tokenizers, their identifiers will also differ. Thus, the identifier n does not correspond to the same subword in English depending on whether it is in the input of $\mathcal{G}$ or the output of $\mathcal{F}$.

The added complexity in the mappings seems to make learning of the present architecture more challenging than necessary for an already computationally expensive model. This doubling of the number of token distributions is the primary reason for the inability of this hybrid model to learn effectively.

4.3 CycleGN

The present implementation finally makes it possible to obtain a functional [Neural Machine Translation](#) model that can be adapted to numerous language pairs. The CycleGN [84] framework differs from the previous ones in two manners: the use of a different form of pre-training, and the abandonment of discriminators. However, there are other additional modifications, such as the reorganisation of the vocabulary. A paper describing CycleGN has been peer-reviewed and published, and a reproduction of the publication is available in [Appendix B.3](#).

4.3.1 Vocabulary Organisation

Encoder-decoder models can either employ a unified tokenizer or two distinct tokenizers. In the case of a single tokenizer, it is trained using sentences from both the source and target distributions, avoiding duplicates. This approach facilitates the sharing of encoder and decoder embedding layers, thus reducing computational demands and improving the accuracy of the model [277].

Let X_a , X_b , and X_c be the tokens of a source language, Y_a , Y_b , and Y_c the tokens of a target language, and Z_a , Z_b and Z_c the tokens that are in common between the two languages. For the sake of clarity, special tokens such as $\langle eos \rangle$ and $\langle unk \rangle$ are ignored. Table 4.11 shows the vocabulary of a single tokenizer trained on both the source and target languages.

TABLE 4.11: Vocabulary organisation with a single tokenizer

Token	Token ID
X_a	1
Y_a	2
Z_a	3
X_b	4
Y_b	5
Z_b	6
X_c	7
Y_c	8
Z_c	9

In contrast, the alternative approach involves training one tokenizer on the source distribution and another one on the target distribution. Although this method restricts the possibility of tying embeddings, it can potentially double the vocabulary size. The overall vocabulary size of the model in this scenario is the cumulative total of the two individual vocabularies, barring shared tokens such as punctuation symbols. Table 4.12 shows the vocabulary organisation of such an approach.

Although contemporary Transformer models like [BERT](#) and [GPT](#) typically utilise a single tokenizer, the present work introduces a novel vocabulary methodology that amalgamates the aforementioned approaches. This method involves training two tokenizers, each for a respective language and with half the vocabulary size. Subsequently, the

TABLE 4.12: Vocabulary organisation with two tokenizers

Source token	Source token ID	Target token	Target token ID
X_a	1	Y_a	1
X_b	2	Y_b	2
X_c	3	Y_c	3
Z_a	4	Z_a	4
Z_b	5	Z_b	5
Z_c	6	Z_c	6

identifiers of one tokenizer are adjusted to prevent overlap, yielding a result analogous to that of a single tokenizer that includes duplicates across languages. It is important to note that special tokens such as $\langle eos \rangle$ (End of Sequence) and $\langle pad \rangle$ (Padding) are shared and not duplicated. This strategy is designed to simplify model analysis during development, albeit at the expense of a reduced vocabulary. Table 4.13 shows the organisation of the vocabulary of our tokenizer, which contains duplicates.

TABLE 4.13: Vocabulary organisation with our tokenizer

Token	Token ID
X_a	1
X_b	2
X_c	3
Z_a	4
Z_b	5
Z_c	6
Y_a	7
Y_b	8
Y_c	9
Z_a	10
Z_b	11
Z_c	12

4.3.2 Dropping the Discriminator

The CycleGAN methodology, as indicated by its nomenclature, is predicated on the GAN framework. This paradigm involves the training of a Generator model in conjunction with another model, termed the Discriminator. The Discriminator is specifically trained to distinguish between authentic samples drawn from the dataset and synthetic samples produced by the Generator.

However, as mentioned in Section 4.2.2, gradient computation cannot occur during the generation of $\hat{x}$ and $\hat{y}$, as they are produced in inference mode. In Transformer-based models, when no labels are available, the generation process becomes non-differentiable due to the token selection step: at each step, the model selects the most likely token, a discrete operation that prevents gradient backpropagation. As a consequence, discriminators cannot be used in the present work, since adversarial training relies on gradient-based optimisation. This is why CycleGN is not an “Adversarial” approach, hence its name.

4.3.3 Absence of Intermediate Evaluation

The loss of the Discriminators creates a major problem: the lack of an intermediate evaluation. This intermediate evaluation refers to the discriminators' process of confirming that $\hat{x} \in X$ and $\hat{y} \in Y$. The model consistently defaults to a trivial solution in which it merely reproduces its input, effectively satisfying the loss function criteria without achieving any meaningful learning or transformation of the data.

Indeed, as there is no Discriminator to ensure that $\hat{x}$ belongs to $\mathcal{X}$ and $\hat{y}$ belongs to $\mathcal{Y}$, $\mathcal{G}$ and $\mathcal{F}$ will converge towards $x = \hat{y} = \hat{x}$ and $y = \hat{x} = \hat{y}$, as this approach achieves an optimal outcome on the [Cycle-Consistency Loss](#) function, registering a value of zero, as schematised in Figure 4.14.

That is, if both $\mathcal{G}$ and $\mathcal{F}$ do not apply any change to their input, they can still achieve $\mathcal{G}(\mathcal{F}(y)) = y$ and $\mathcal{F}(\mathcal{G}(x)) = x$ without ever learning to translate.

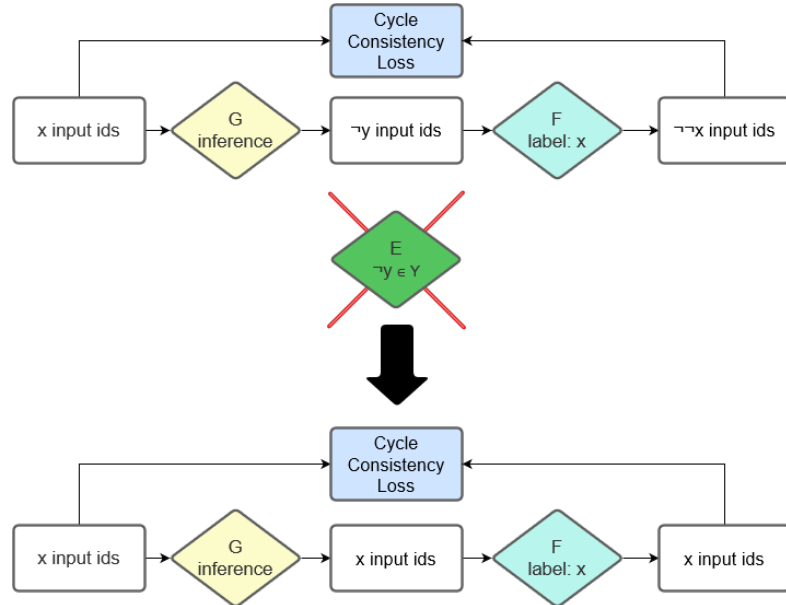


FIGURE 4.14: In the absence of a Discriminator $y \in \mathcal{Y}$, the CycleGN architecture will converge towards a state where no translation happens and still perfectly satisfy the [Cycle-Consistency Loss](#) function

4.3.4 Escaping the Global Minimum

As discussed in Section 4.3.3, the absence of discriminators prevents the framework from ensuring that the intermediate translations belong to the correct language, and even encourages the networks to not apply any modification to the input. In fact, the global minima for the [Cycle-Consistency Loss](#) is obtained when the input is compared to itself.

The first step of the training process is to shift the randomly initialised weights of the networks away from the global minimum of not performing any modification to the input. This can be achieved by pre-training the network on an unrelated task, which still allows it to learn the relationships between tokens in both the source and the target languages.

Although many pre-training strategies can be used to pre-train a [seq2seq](#) transformer, [Masked Language Modeling](#), discussed in Section 2.4.2, is trivial to implement and does not require any additional dataset. [Next Sentence Prediction \(NSP\)](#) was also considered, but as discussed in Section 2.4.3, [RoBERTa](#) only relied on [Masked Language Modeling](#) and still outperformed [BERT](#).

4.3.5 Pre-Training

[Masked Language Modeling \(MLM\)](#) is one of the two pre-training strategies implemented in BERT, as discussed in Section 2.4.3, wherein a specified proportion of tokens within the input text are substituted with a unique `<mask>` token. The objective of the neural network under this paradigm is to accurately reconstruct the original sentence. This process enables the model to discern intricate relationships between words and develop a deep representation of the language.

By randomly forgetting certain tokens, the model learns to make links between the words in a sentence, and the multihead mechanism can improve its ability to detect the relationships between the different tokens in order to reconstruct sentences more efficiently, which is later useful during real training. This pre-training has revealed excellent performances in diverse [Natural Language Processing](#) application such as sentiment analysis [9], text classification [342], [Named Entity Recognition \(NER\)](#) [333] [49] [8] and paraphrase detection [172].

As [Masked Language Modeling](#) does not require any labelling, it is perfectly adapted to the CycleGN approach. A single model $\mathcal{H}$ is trained on the entire dataset for a single epoch to reconstruct both languages, with 15% of the input tokens masked. When CycleGN is trained, rather than randomly initialising $\mathcal{G}$ and $\mathcal{F}$, the parameters from $\mathcal{H}$ are copied directly to $\mathcal{G}$ and $\mathcal{F}$. In fact, since $\mathcal{H}$ learns to reconstruct both the languages $\mathcal{X}$ and $\mathcal{Y}$, it can be used to initialise both networks. The architecture of $\mathcal{H}$ is identical to the architecture of the generators, which is described in Section 4.1.3. Figure 4.15 shows the training process of $\mathcal{H}$.

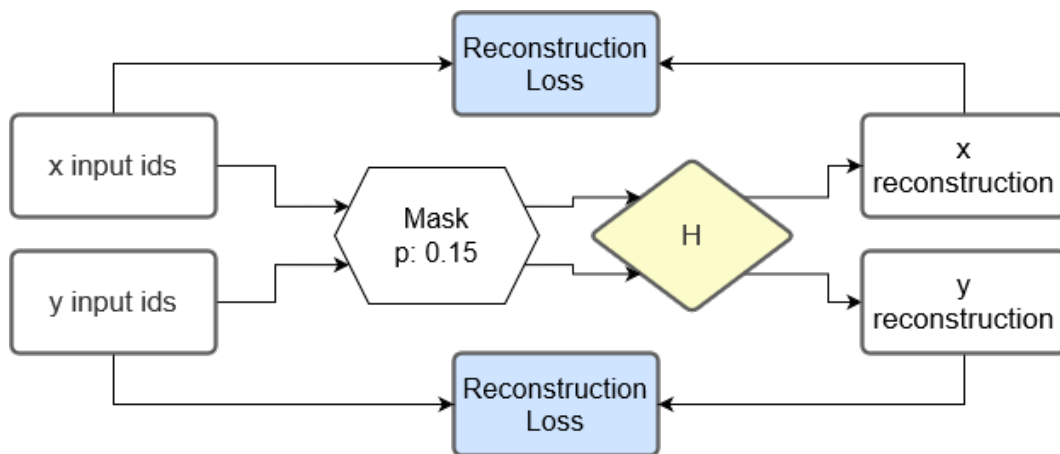


FIGURE 4.15: Masked Language Modeling training process

4.3.6 Training

The two models $\mathcal{G}$ and $\mathcal{F}$ are initialised with the weights of the model $\mathcal{H}$. As discussed in Section 4.3.1, a unique vocabulary is used for both models, thus each embedding layer contains parameters for tokens of the input and target languages. Pre-training the model $\mathcal{H}$ to reconstruct both languages enables the Embedding layers to grasp the connections among tokens from both languages, which is beneficial for the subsequent training phase. In fact, while the model $\mathcal{G}$ interacts mainly with the tokens from the X distribution due to its task of achieving a mapping $X \mapsto Y$, it also interacts with the tokens from the Y distribution when it needs to compute $\text{IdentityLoss}(G_{X \mapsto Y}(y), y)$. As a matter of fact, when two models $\mathcal{H}$ and $\mathcal{I}$ are initialised, the first one being trained on reconstructing the first language and initialising $\mathcal{G}$ and the second one being trained on reconstructing the second language and initialising $\mathcal{F}$, the CycleGN framework did not escape the global minimum.

Each batch contains sentences in both languages that are then tokenized. For each batch, four different loss functions are computed:

- $\text{Cycle} - \text{ConsistencyLoss}(x, \hat{x}) | \hat{x} = F_{Y \mapsto X}(G_{X \mapsto Y}(x))$;
- $\text{Cycle} - \text{ConsistencyLoss}(y, \hat{y}) | \hat{y} = G_{X \mapsto Y}(F_{Y \mapsto X}(y))$;
- $\text{IdentityLoss}(x, id_x) | id_x = F_{Y \mapsto X}(x)$;
- $\text{IdentityLoss}(y, id_y) | id_y = G_{X \mapsto Y}(y)$.

The two **Identity Losses** are multiplied by a factor of 0.1, similarly to the original CycleGAN work, as they act as a regularisation term. Scaling down these terms shifts the focus to learning effective domain transformations, which is the primary objective. The training process is shown in Figure 4.16.

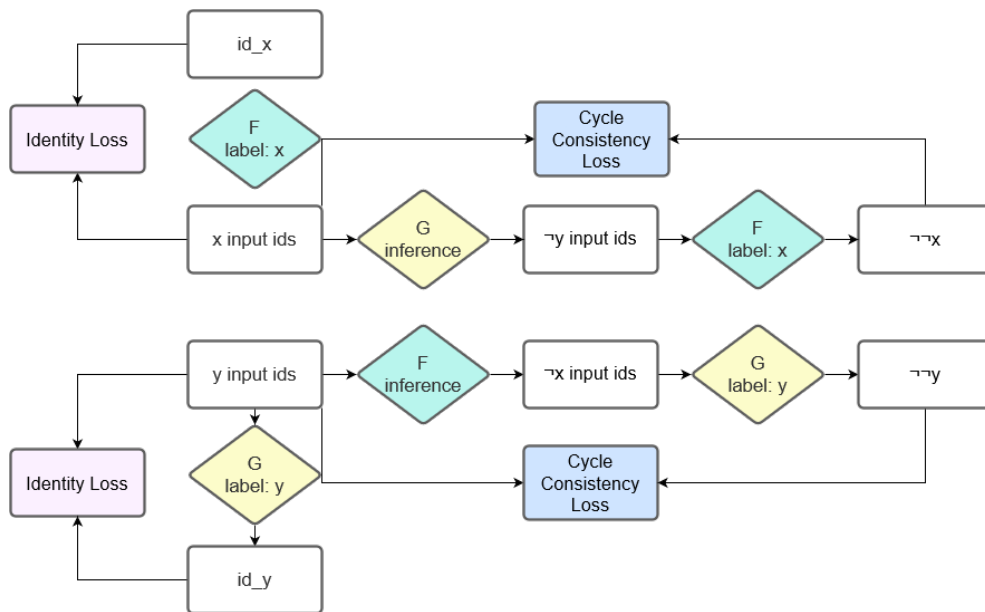


FIGURE 4.16: CycleGN training process

4.4 Challenges

At the start of training, the intermediate translations do not correspond to the source data. For example, one can observe the evolution of a translation pair.

The original German sentence “Mit diesem Kalender will die EZB Voraussetzungen schaffen, die es den Geschäftspartnern in allen Ländern des Euroraums ermöglichen, an den Hauptrefinanzierungsgeschäften und den längerfristigen Refinanzierungsgeschäften des Eurosystems teilzunehmen.” corresponds to the English sentence “In compiling this calendar, the ECB aims to ensure that counterparties in all euro area countries can participate in the main and longer-term refinancing operations.” according to the dataset.

Table 4.14 presents the evolution of the generated translation depending on the number of batches the model has been trained on. Although initially it produces out-of-vocabulary tokens, it ends up producing an approximate translation in 100,000 batches.

TABLE 4.14: Evolution of the translation of a test sentence

Batch number	Generated sentence
1,000	<unk> <unk> <unk> <unk> the<unk> <unk> <unk> <unk> the<unk> <unk> <unk> <unk> s in<unk> <unk> <unk> <unk> s<unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> s and the<unk> s<unk> <unk> <unk> <unk> s<unk> <unk> <unk> <unk>.
10,000	With this calendar, the<unk> deadlines can be made now the commercial<unk> s in all countries that they are now at the mainre<unk> s hotels and the incremental Pro<unk> system<unk> s that crossborder healthcare.
100,000	With this calendar, the ECB deadlines created it the commercial partners in all countries that have Eurozoneed them at the main- reinvestment business and the Occupational ne investment business of the Euro system.

The *English* $\mapsto$ *German* Generator also progressively learns how to translate the sentence. Although the generated sentences only have a slight semantic connection with the original sentence at first, the [Neural Machine Translation](#) model gradually improves. The translation service DeepL.com, is used to translate the generated sentences back into English to facilitate the comprehension for non-German speakers. Table 4.15 displays the evolution of the test sentence output depending on the number of batches.

It is crucial for a CycleGAN that the two models perform approximately equally well. Indeed, since the output of one model is dependent on its input, and the input of one model is the output of the other, it is necessary to ensure consistency between the models in the training.

Without a strategy in place to prevent the performance of the models from diverging, it is possible for one model to gain the upper hand over the other, as can be seen very clearly in Figure 4.17, where the [Cycle-Consistency Loss](#) of one of the two generators suddenly increases. $\mathcal{F}$, ends up performing much better than its counterpart $\mathcal{G}$, which blocks future training.

TABLE 4.15: Evolution of the translation of a test sentence and its backtranslation obtained from Deepl.com

Batch number	Generated sentence	Deepl backtranslation
1,000	A laut Antrag Der diesersreformen, die Zucker 2007 in so zuBANK in Rat finden G durch In Kunden in den Auswirkungen und Seiten- Bestimmungen Beziehungen Mitgliedern Ungarn.	A according to the request of the This reforms, the sugar 2007 in so toBANK in Council find G by In customers in the impact and side provisions relationships members Hungary.
10,000	Die EU verurteilt auf diese Kalender, die EZB 2007 darauf, dassBANK in allen Bundesrepublik Jugoslawien können parallel zu den wichtigsten und Seiten Bestimmungen Verlegungsprojekten führen.	The EU condemned on this calendar, the ECB in 2007 thatBANK in all Federal Republic of Yugoslavia can lead in parallel to the main and side provisions relocation projects.
100,000	Ich geheilte diesen Kalender, den die EZB angestrebt hat, damit die Gegenparteien in allen Steuer-Ländern an den wichtigsten und längerestmöglichen Versandaktivitäten teilnehmen können.	I cured this calendar that the ECB has sought to allow counterparties in all fiscal countries to participate in the most important and longest possible dispatch activities.

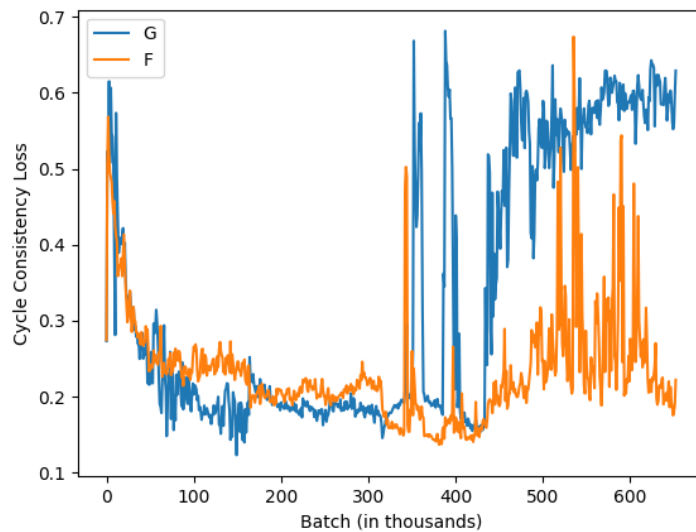


FIGURE 4.17: Divergence in the Cycle-Consistency Loss of CycleGN training

At this stage, even if some translating capabilities can be observed, the imbalance inherent in this architecture prevents any further training, and therefore any improvement. It is therefore essential to find strategies for moving from a roughly working prototype to a credible unsupervised learning architecture for [Neural Machine Translation](#). The first thing to do is to make learning more stable and ensure that neither Generator outperforms the other. Once stability has been achieved, performance and learning speed can be optimised.

As discussed in Section 4.4, both $\mathcal{F}$ and $\mathcal{G}$ need to learn and progress at approximately the same rate, otherwise the training degrades and stops. Thus, it is crucial to implement a mechanism to avoid this instability.

4.4.1 Punishing the Model

One way of avoiding a large disparity between the effectiveness of the two generators is to constantly balance the training speed of the models. By keeping a running average of the latest losses for each model, and then comparing this average between the two models, it is possible to determine whether one model is “ahead” of the other. Then, a threshold and slowdown constants are chosen, the first of which determines the level of disequilibrium at which we slow down the learning of a model, and the second concerns the size of this slowdown.

In the case in point, during the present research, the imbalance threshold was 20%, that is, the training of a model was slowed down as soon as there was a difference of more than 20% between the two moving averages of the losses. The gradients of the model with the lowest average are divided by the slowdown constant, in this case 10, until the difference between the averages falls below the threshold. Since training a CycleGN model is expensive, that this method of punishing the model slows down the training process even more, and that a better method to prevent the instability was found shortly after, these are the only values that were tried.

There are several inherent problems with such a solution. The first is that determining the optimal values of the threshold and the slowdown constants requires training multiple models, which is a computationally expensive process. It is also possible that for optimal training, these figures need to vary over the course of the training session, so that the values for the imbalance threshold and the slowdown might even be continually reevaluated.

Secondly, slowing down the training speed of a model means slowing down the training of the entire architecture. In a context where the type of architecture presented here already requires a longer training time than conventional transformer methods, such a great loss of performance would reduce the viability of the experiments.

4.4.2 Gradient Clipping

Gradient clipping is a method employed in the training of machine learning models, particularly in deep learning scenarios, to counteract the issue of exploding gradients. The problem of exploding gradients arises when gradients become excessively large during training, causing numerical instability and hindering convergence to an optimal solution.

In a formal sense, gradient clipping involves constraining the magnitude of gradients during the training process. This is achieved by setting a threshold or maximum allowable value for gradients. If, during backpropagation, the gradient vector exceeds this

threshold, it is scaled down proportionally to ensure that it remains within the specified limit. This approach prevents gradients from becoming excessively large, thereby mitigating the risk of numerical instability and enhancing training stability.

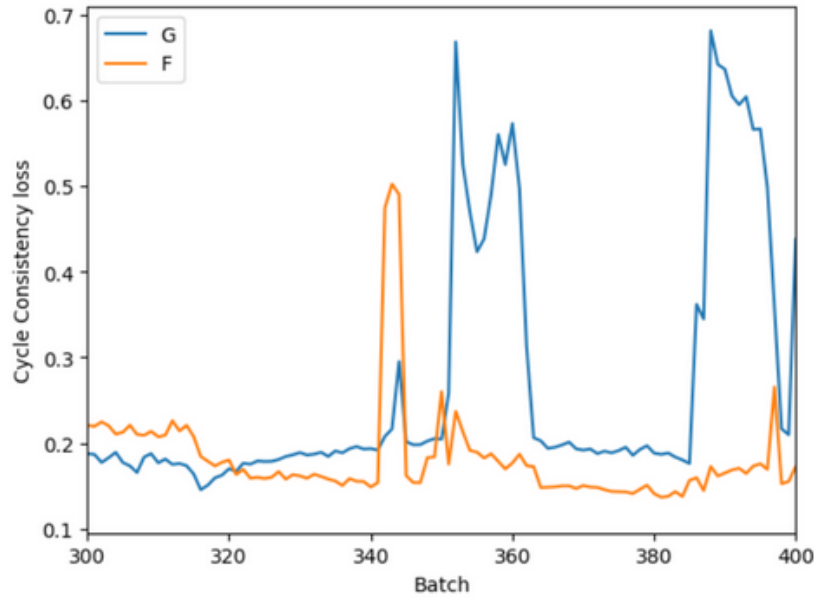


FIGURE 4.18: Divergence in the [Cycle-Consistency Loss](#) of CycleGN training

Several peaks can be observed in Figure 4.18, which is a magnified view of Figure 4.17 between batches 300 and 400, that is, before the models diverge. Although these massive losses could have had several causes, the preventive use of gradient clipping appeared reasonable as a means of limiting these rapid and unpredictable fluctuations. Similarly, while the divergence might be completely unrelated to these massive fluctuations, there appears to be a temporal proximity between these events.

Clipping by Norm and Clipping by Value

Clipping gradients can be done in two main ways: clipping by norm and clipping by value. In clipping by norm, a threshold is set on the overall norm (magnitude) of the gradient vector, and if the norm of the gradient vector exceeds the specified threshold, the entire vector is scaled down proportionally to ensure that it does not exceed the limit. In the case of clipping by value, any particular element that exceeds the specified threshold is truncated or scaled to the threshold value.

The choice between the two techniques is typically influenced by the characteristics of the optimisation problem and the particular needs of the model under training. While clipping by norm stands out as a more prevalent and direct method, the decision might differ depending on the specific context and the inherent properties of the data.

Choosing Sensible Defaults

There are two decisions to be made: to select the type of gradient clipping, i.e. either by norm or by value, and to find the threshold above which to apply the clipping. The HuggingFace library has a feature to facilitate model training, which they describe as follows: “The Trainer class provides an API for feature-complete training in PyTorch for most standard use cases”. By default, this class clips the gradients by norm, with a threshold of 1.0, so these parameters were chosen for the work here.

The Effect of Gradient Clipping on CycleGN

Clipping the gradients does not eliminate sudden increases in losses, but it reduces their absolute value, as shown in Figure 4.19. No divergence between the two models is observed, which seems to confirm the effectiveness of this technique. Furthermore, should a divergence occur, the previously mentioned technique of limiting a model’s training speed according to the performance difference can be added to gradient clipping.

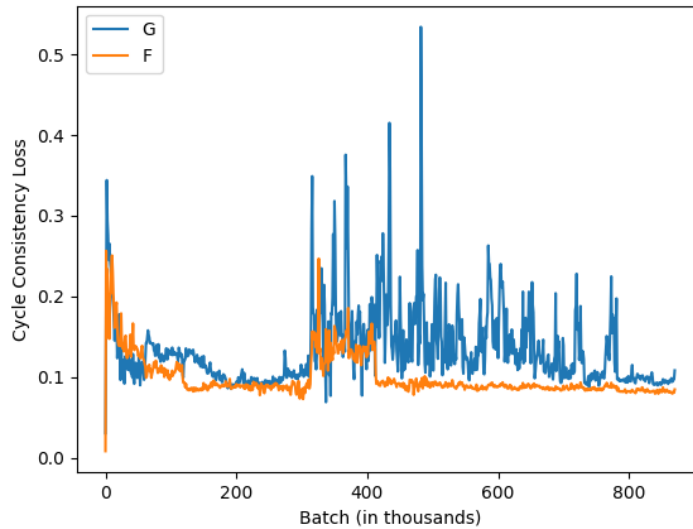


FIGURE 4.19: Cycle-Consistency Loss of CycleGnN training with gradient clipping

4.4.3 Batch Size

The original CycleGAN research mentions using a batch size of one, and while the authors did not state the reason in the research paper, one of the authors explained it in a GitHub issue⁸. They did not try with any other batch size; it was only because they had limited GPU memory.

Previous work [289] compared the effect of batch size in CycleGAN architecture and found a noticeable degradation when it is increased. Indeed, whether by the example images shown in the paper or the cosine dissimilarity that they computed, the model that was trained with a larger batch size performed worse. However, having a batch size

⁸<https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/27>

of one increases the learning time. Although it took the authors eight hours of training to reach 200 epochs with a batch size of one, it only took two hours with a batch size of 64. As this previous work proved, the batch size can be a factor in the quality of the generated hypotheses. With an available memory of 24GiB, a maximum of 32 sentences can be processed at a time. A batch size of one was tried to train CycleGN, but this did not improve the results. In fact, by only changing this hyperparameter, the model stops learning quickly and stays stuck on the same word over and over. The batch size was chosen to be the largest possible on current hardware, which in the present case was 32. An intermediate rate of 16 was tried, but the learning speed was slower and decreased the quality of the translations.

4.4.4 Per-Epoch Permutation

In the theoretical case of a neural model that contains sufficient parameters in relation to the size of the dataset, it is possible to encounter an overfitting situation. In this situation, the model acts as a simple key-value database instead of assimilating the links between inputs and outputs and the relationships between tokens. Even if this case remains theoretical in the context of this research, measures can be taken to prevent this from happening.

Since the aim is to train a [Machine Translation](#) model from unaligned data, the dataset can be shuffled differently for each epoch. This should prevent the same sentence from being translated the same way over and over again and is inexpensive. This should be considered as a form of data augmentation, since the size of the dataset is artificially increased.

Permuting the entire corpus after the end of each epoch means that the pairs of sentences x, y obtained from languages X and Y would be renewed at each iteration, so that the same exact pair of sentences does not appear frequently enough to lead to overfitting.

To test the present hypothesis, two models were trained. As training a model is particularly costly, only a subset of the dataset, the first 0.2%, were used for 5 epochs. The English sentences were permuted at each epoch in the first condition, whereas this was not the case in the other condition. The [COMET](#) scores of the models are presented in Table 4.16.

TABLE 4.16: [COMET](#) score of the CycleGN framework depending on data augmentation

	With per-epoch permutation	Without per-epoch permutation
English $\rightarrow$ German	0.2411	0.2268
German $\rightarrow$ English	0.2635	0.2458

Per-epoch permutation is a form of data augmentation that is proving effective, since the [COMET](#) scores obtained are higher. Naturally, it is difficult to know whether this result can be extrapolated to larger and more realistic datasets.

4.4.5 Size of the Training Dataset

The datasets used in [Machine Translation](#) are generally text files of several gigabytes per language. As mentioned in [Section 4.1.3](#), for prototyping purposes, the dataset used in this work is extracted from the English-German dataset from the [WMT2023](#) conference, where each of the two languages contains almost 300 million sentences.

Since, in a parallel dataset, the model receives the exact corrections it should apply in each batch, sentence-based learning is inherently faster than in the CycleGAN framework, where the correct solution is never explicitly provided and can only be approximated through trial and error. Thus, for the same model architecture and the same dataset size, it is expected that using a parallel dataset will result in a better performing model than a network trained with a non-parallel dataset.

Although 300 million sentences may seem a reasonable size for conventional neural network training, it is disproportionately large for this project. In fact, with the equipment used, training two traditional parallel models for one epoch would only take four days. However, the training duration increases to 60 days per epoch using the proposed CycleGN architecture on this dataset.

Previous academic work has explored whether it is preferable to reduce the dataset in order to carry out several epochs, or to carry out a single epoch on the entire dataset while maintaining the same number of sentences.

- One single epoch: For unsupervised [Natural Language Processing](#) training such as [BERT](#) and [GPT](#), training on a single epoch rather than multiple smaller ones results in a shorter training time and better performance [\[184\]](#). This is likely because a larger training volume increases diversity and diversity improves performance [\[133\]](#) [\[287\]](#). Although the risk of overfitting tends to increase with more training iterations, overfitting is prevented with one-epoch training. Larger models are more prone to overfitting [\[404\]](#), as by containing more parameters, they are more susceptible to “remembering” the exact training sentences, especially if they are exposed to the same example multiple times. Regarding [Neural Machine Translation](#), with the same number of iterations and parameters, a single epoch seems to result in a better [BLEU](#) score [\[272\]](#);
- Multiple smaller epochs: Repeated training on a smaller dataset in multiple epochs involves using the same examples multiple times, which can enhance the ability of the model to identify and reinforce learnt patterns.

In the non-parallel case, using a larger dataset certainly allows the model to be exposed to a greater diversity of phrases and words, capturing large representation of the languages, including rare phrases and structures, which might enhance the ability of the model to generalise to new data.

Although in theory the greater diversity induced by a larger dataset is an advantage, experiments have been carried out in this work on a subset of the dataset to demonstrate

or even quantify the real advantages. For example, by training a model on 1% of the dataset, its performance can be compared with that of models trained for 5 epochs on 0.2% of the dataset during tests on per-epoch permutation. Table 4.17 displays the results obtained in Table 4.16 and compares them with the results obtained by training a model on a larger dataset.

TABLE 4.17: COMET score of the CycleGN framework depending the number of epochs

	Single large epoch	Multiple smaller epochs
English $\rightarrow$ German	0.2727	0.2411
German $\rightarrow$ English	0.2715	0.2635

It is difficult to be certain that these results can be extrapolated to larger datasets. However, in line with theoretical predictions and previous work, training a model over a larger epoch seems to result in a better performing model.

4.4.6 Diminishing the Cost

To accelerate model training and inference, there are two possibilities that can also be combined. The first option is to reduce the number of model parameters, which can be achieved by decreasing the number of layers or the size of these layers. Another option is to reduce the size of each parameter by a method known as quantisation.

Quantisation

Quantisation, discussed in Section 2.8, is a mechanism by which a model’s weights are converted from an m -bit representation to an n -bit representation, with $n < m$. All previously trained models have used float32, but they can be converted, for example, to bfloat16 or float16, thereby halving the size of each weight and, depending on the hardware used, potentially doubling the computation speed.

To examine the effect of parameter size reduction on CycleGN performance, four trained models were quantised to float16 and bfloat16, and the results are summarised and available in Table 4.18. The models were quantised at inference time using the method “autocast (dtype = torque.dtype)” from “from torch.cuda.amp” where the dtype is float16 or bfloat16. PyTorch offers another method, “half()”, to quantise models to float16. The results obtained using this method did not differ from those achieved using “autocast()”.

Quantising weights in float16 or bfloat16 has little to no effect on the performance of a model trained in float32. Quantizations to int4 and int8 could not be tested due to the Marian architecture used with HuggingFace. An encountered error, “*RuntimeError: 'addmm_cuda' not implemented for 'Char'*” could not be resolved, preventing the trial of these quantisation methods.

TABLE 4.18: COMET score obtained by models trained using the CycleGN framework depending on the dataset and quantisation

Language Pair	float32 (COMET)	float16 (COMET)	bfloat16 (COMET)
English → Spanish (WMT24, permuted)	0.611	0.611	0.611
Spanish → English (WMT24, permuted)	0.627	0.627	0.627
English → French (WMT14, permuted)	0.573	0.573	0.573
French → English (WMT14, permuted)	0.639	0.638	0.639
English → German (WMT16, permuted)	0.499	0.499	0.499
German → English (WMT16, permuted)	0.550	0.550	0.550
English → German (WMT24, non-intersecting)	0.604	0.604	0.604
German → English (WMT24, non-intersecting)	0.638	0.638	0.637
English → Chinese (WMT24, permuted)	0.427	0.427	0.427
Chinese → English (WMT24, permuted)	0.445	0.445	0.445
English → Chinese (WMT24, non-intersecting)	0.323	0.323	0.323
Chinese → English (WMT24, non-intersecting)	0.378	0.378	0.378
Japanese → Chinese (WMT24, permuted)	0.489	0.489	0.488
Chinese → Japanese (WMT24, permuted)	0.486	0.486	0.486

Smaller Models

Table 4.19 references the changes that were made from the default architecture, detailed in Table 4.8, to further reduce the computational cost of the architecture. Although the default architecture contains 158,769,152 trainable weights, the smaller model only has 39,735,296, which corresponds to a reduction in the number of parameters by a factor of four.

TABLE 4.19: Differences in the architecture of the default CycleGN model and the small CycleGN model

Setting	Default model	Small model
Vocabulary size	32,000	16,000
Dimensionality	1024	512
Encoder feed-forward	2048	1024
Decoder feed-forward	2048	1024

Reducing the number of parameters results in halving the model training time, both for [Masked Language Modeling](#) pre-training and for cyclical training itself. This is in part due to the fact that a larger batch size of 128 can be processed at a time, compared to the maximum batch size of 32 with the default model. Although quantisation can be performed without further training, it is necessary to restart training from scratch to test the capabilities of the smaller model. Although this operation is considerably faster, it remains costly, which explains why this architecture was trained on only three language pairs.

As demonstrated in Table 4.20, decreasing the number of parameters decreases the performance of the model compared to the default model when the dataset is identical. This reduction in capability due to the smaller number of parameters was anticipated and appears to be in the range of 15 to 75%, although this range is necessarily imprecise due to the limited number of models trained.

Similarly to what is observed with the default architecture, quantising the parameters of the small architecture does not further decrease the model’s performance.

Figures 4.20, 4.21, and 4.22, respectively, illustrate the evolution of COMET scores achieved by the small models in the pairs of languages English-French, English-German, and English-Spanish. Unlike earlier trends, significant fluctuations in performance are

TABLE 4.20: COMET score obtained by models trained using the CycleGN framework depending on the number of parameters

Language Pair	Default (COMET)	Small (COMET)	Ratio
English → Spanish (WMT24, permuted)	0.611	0.518	1.18
Spanish → English (WMT24, permuted)	0.627	0.543	1.15
English → French (WMT14, permuted)	0.573	0.426	1.34
French → English (WMT14, permuted)	0.639	0.482	1.33
English → German (WMT24, permuted)	0.604	0.431	1.40
German → English (WMT24, permuted)	0.638	0.368	1.73

less frequent and of lower magnitude compared to the default model. This configuration allows for the quadrupling of the batch size, which probably helps counteract substantial deviations during training. This assumption is further supported by the findings in Section 4.4.3, where it was observed that a batch size of 1 does not facilitate learning, and a batch size of 16 results in lower translation quality compared to a batch size of 32.

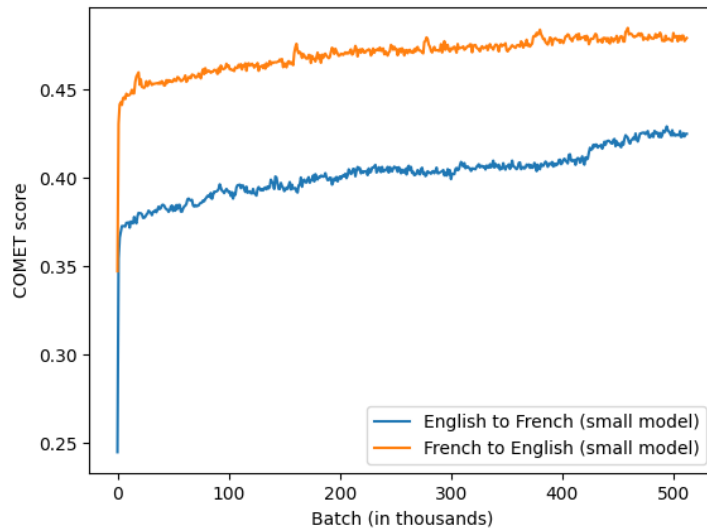


FIGURE 4.20: Evolution of the COMET score during the permuted training of the small CycleGN framework on the WMT14 English-French language pair

However, while the increased stability is desirable, as it allows for more predictable training and the ability to halt training without the risk of stopping at a performance low point, this stability comes at the cost of translation quality because of the reduced number of parameters. This trade-off between stability and quality reflects a complex balance in model optimisation, where larger batch sizes, due to a lower number of parameters, contribute to training consistency but may limit the model’s overall performance potential.

4.4.7 Semantic Confusions

In developing CycleGN, one notable tendency is its inclination to occasionally substitute words with others that are semantically close yet contextually incorrect. For example, it can replace the name of one country with another, swap dates, or substitute a person’s name with a similar but incorrect alternative. This behaviour reflects the model’s

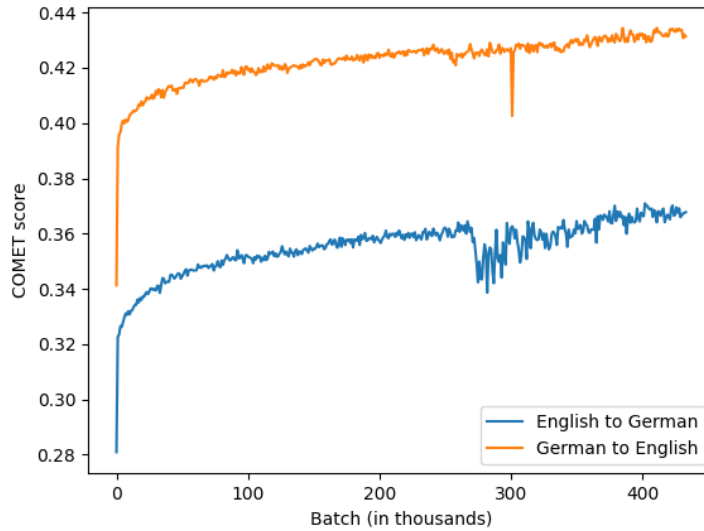


FIGURE 4.21: Evolution of the [COMET](#) score during the permuted training of the small CycleGN framework on the [WMT24](#) English-German language pair

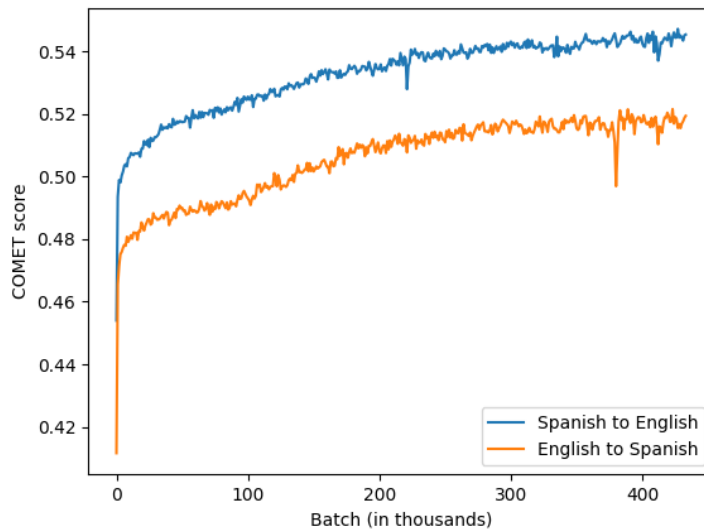


FIGURE 4.22: Evolution of the [COMET](#) score during the permuted training of the small CycleGN framework on the [WMT24](#) English-Spanish language pair

prioritisation of semantic proximity over strict lexical accuracy. Although such substitutions may seem minor, they can have significant implications for the reliability of the model, especially in contexts where specific entities or precise information are critical. This pattern of semantically approximate word choices warrants a deeper exploration to understand its underlying causes and address its impact on translation accuracy.

Given the limitations of [BLEU](#) in capturing the nuances of translations that deviate lexically while retaining semantic accuracy, it is not an appropriate metric for assessing CycleGN’s performance. [BLEU](#) fundamentally emphasises surface-level word overlap, heavily favouring literal translations where n -grams align closely with the reference text.

This approach does not account for the cases where synonym substitutions, paraphrasing, or structural adjustments convey the intended meaning more naturally or effectively. As a result, CycleGN, which often uses flexible lexical choices and can confuse certain words with others that are semantically close, receives disproportionately low [BLEU](#) scores. These low scores inaccurately reflect the translation quality of the model, penalising meaningful divergences that enhance readability and coherence. This is the reason why another neural metric, the [COMET](#) score, is preferred to [BLEU](#) for CycleGN evaluation.

Locations

To illustrate what is meant by confusion between places, a trivial but representative example is as follows. The German sentence:

Ich habe immer in Alaska gelebt.

which means “I have always lived in Alaska.” is translated by CycleGN as:

I’ve always lived in the United States.

This confusion by the model is frequently observed in place names. Table 4.21 shows the results obtained when varying the location of the sentence above used as input. Although some place names are well translated, the most frequent case is the observation of a semantically close change, i.e. a country by another country, or a city by another city, even if this is not systematic, as shown by the example of the city of Algiers, translated by an entire geographical region.

TABLE 4.21: Examples of location confusion produced by CycleGN from German to English

Input location	Target location	CycleGN translation
Alaska	Alaska	United-States
Ungarn	Hungary	Finland
Finnland	Finland	Belgium
Namibia	Namibia	Napa
Irland	Ireland	Spain
China	China	China
Japan	Japan	Japan
Thailand	Thailand	Nigeria
Paris	Paris	London
Dublin	Dublin	Lisbon
Algier	Algiers	eastern Europe

Names

As with places, surnames and first names differ radically. The phrase “My name is William.” is:

Ich heie William.

or alternatively:

Ich heie Wilhelm.

in German, depending on whether the first name is translated into the language. However, the model radically changes the first name and generates:

My name is Tony.

Table 4.22 shows examples of name and surname changes. First names and surnames are replaced by other first names and surnames, which demonstrates a certain understanding of the semantic proximity of the concept of first name, but does not prevent a significant loss of information and precision during translation.

TABLE 4.22: Examples of name confusion produced by CycleGN from German to English

Input location	Target location	CycleGN translation
Wilhelm	William	Tony
Peter	Peter	Michael
Smith	Smith	Wilson
Lavoisier	Lavoisier	Lerail
Kubrick	Kubrick	Brenus
George	George	Victoria
Bertha	Bertha	Max
Olga	Olga	Fabio Capello
Stanley	Stanley	Stanley

Numbers and Dates

The information contained in numbers and dates is particularly impaired, even in simple sentences. In German, to “I am 18.” is expressed as:

Ich bin 18.

However, CycleGN translates this sentences as:

I’m 20.

Table 4.23 shows the confusion of CycleGN depending on the age in the sentence above. Although there seems to be some consistency in the age changes, i.e., a teenager does not become a senior citizen, the loss of information is particularly significant.

This confusion over ages is also reflected in the dates, with a further addition in the names of the months. Similarly, one month is replaced by another, which indicates an understanding of semantic proximity but leads to significant inaccuracies when translating. For example, the German sentence:

TABLE 4.23: Examples of age confusion produced by CycleGN from German to English

Input location	Target location	CycleGN translation
18	18	20
15	15	15
33	33	41
4	4	13
96	96	91
150	150	120
1	1	1
47	47	38

Das Datum ist der 18. September 2020.

which means “The date is 18 September 2020.” is incorrectly translated by CycleGN as:

The date is June 20, 2020.

More examples are shown in Table 4.24.

TABLE 4.24: Examples of date confusion produced by CycleGN from German to English

Input location	Target location	CycleGN translation
14. Februar 1844	14 February 1844	May 12, 2049
2. Dezember 1142	2 December 1142	April 4, 1158
27. August 475	27 August 475	18th 1370
3. Januar 2145	3 January 2145	January 3rd 2947

These confusions introduced by CycleGN could, however, be rectified through post-processing. The implementation of a secondary model, such as [Named Entity Recognition](#), could mitigate this limitation of CycleGN and enhance translation quality. However, this approach was not pursued in order to evaluate the raw performance of CycleGN.

COMET

Since CycleGN tends to confuse words that are semantically close, and any score based on token overlap cannot take all this proximity into account, the results obtained by CycleGN when measured by token overlap will be systematically low. [COMET \[294\]](#), in contrast, evaluates translations through a learnt context-aware framework that captures semantic fidelity and fluency without being restricted to specific lexical choices. Because it assesses translations based on how well they align with the intended meaning of the source text rather than on strict word-level correspondence, [COMET](#) is more suited to evaluating models like CycleGN, which prioritise meaning over form. This metric provides a nuanced understanding of translation quality by recognising valid lexical and syntactic variations that enhance the readability and naturalness of the translated text.

Similarly to the previous analysis, in all further analyses, the [COMET](#) scores are used to benchmark the performance obtained by CycleGN. This choice ensures that evaluation is

aligned with the model’s core objectives: producing fluent and meaning-equivalent translations rather than exact lexical matches. The decision to use COMET22 was driven by its proven reliability, widespread adoption, and computational efficiency. The ensemble model, while potentially offering marginal improvements in performance, comes with significantly higher computational requirements, both in terms of memory usage and inference time. Given the scale of evaluations conducted in this research, the additional computational cost and complexity of using the larger ensemble model were not justified for the scope of this study. Furthermore, COMET22 remains highly competitive and is highly regarded for its robust evaluation capabilities, making it a practical and credible choice to assess translation quality in this context.

4.5 Results

To demonstrate the learning capabilities of the CycleGN framework, multiple models were trained on language pairs from a diverse range of linguistic backgrounds. This section presents the results achieved by the architecture on the prototyping dataset, as well as the numerous models trained for WMT24. The choice of the evaluation metric, specifically the COMET score, is discussed in Section 4.4.7.

4.5.1 Results on the English-German Language Pair

The model presented here was trained for one epoch on the dataset described in Section 4.1.3, incorporating all optimisations detailed in Section 4.3. This configuration allows for a direct comparison with the baseline architecture-matched parallel models discussed in Section 4.2.1, which were conventionally trained on the same parallel sentences. The same dataset splits were used to ensure that there is no overlap between the training set, the development set, and the test set. To track the progress of CycleGN, the entire development set was translated every 1000 batches.

Consequently, this model is fully aligned with the baseline models in terms of architecture, dataset, and training conditions, enabling a clear assessment of the impact of the implemented optimisations on performance outcomes.

Figures 4.23 and 4.24 display the evolutions of the COMET score of the development set for both generators $\mathcal{G}$ and $\mathcal{F}$, respectively, on the permuted English-German dataset and on the non-intersecting English-German dataset. Table 4.25 presents the COMET scores obtained on the test set after the Generator training was complete and compares the scores to those obtained by the baseline architecture-matched models, reproduced from Table 4.9.

TABLE 4.25: COMET scores of the CycleGN and baseline models

Language Pair	Permuted (COMET)	Non-intersecting (COMET)	Baseline (COMET)
English $\rightarrow$ German	0.604	0.595	0.780
German $\rightarrow$ English	0.638	0.615	0.775

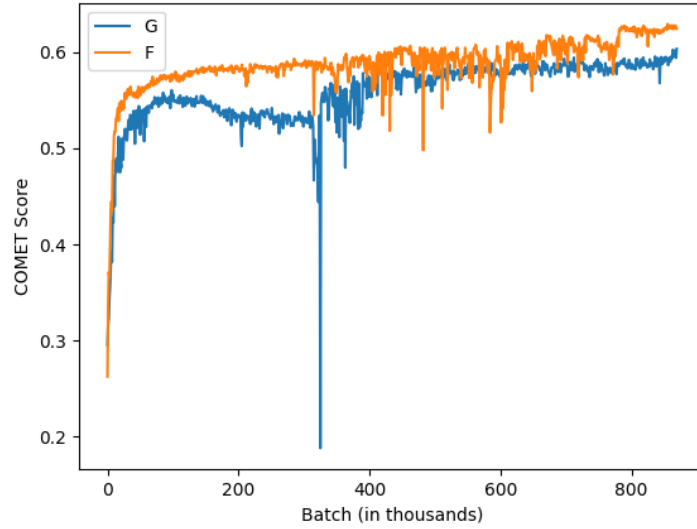


FIGURE 4.23: Evolution of the COMET score during permuted CycleGN training

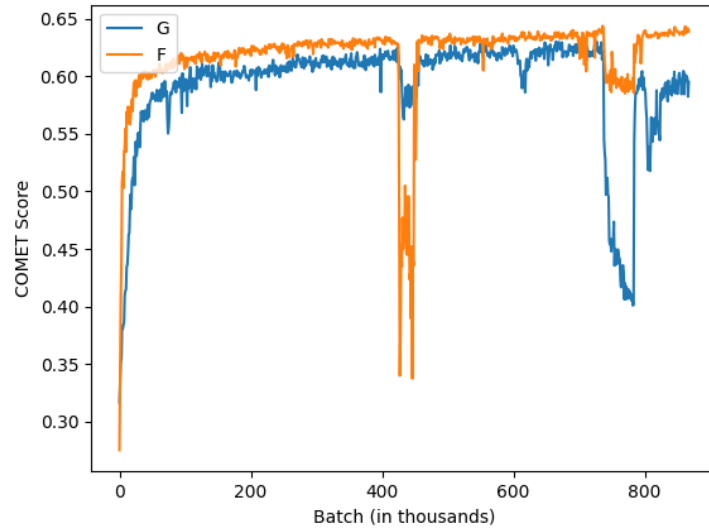


FIGURE 4.24: Evolution of the COMET score during non-intersecting CycleGN training

As expected, the results obtained are inferior to those obtained by the parallel models. However, the CycleGN framework is still capable of learning from a non-parallel dataset. The results also demonstrate that it is possible to train an unsupervised [Machine Translation](#) model from non-intersecting data.

The results of the two generators are highly correlated, as each needs the other, and the training, although slow, shows an improvement in the COMET score with the number of samples examined. However, a very discernible quality drop is observed at one point during the permuted training process after the 325,000th batch mark. Similar collapses are observable during the training of the non-intersecting CycleGN model. As the observations are similar for both permuted and non-intersecting models, only the permuted training is examined for brevity.

Sudden Collapse

Delving into the translations of the development set performed at every 1,000th batch interval reveals substantial and abrupt modifications. Table 4.26 presents the evolution of a translation of the development set.

These alterations in the translations, despite their detrimental effect on the translation’s quality, ostensibly do not exert a significant influence on the aggregate translation score at first. However, they are accurate in predicting the drop in quality that ensues.

Examining the progression of alterations without delving into the translation quality, one can discern a clear pattern.

- Initially, an inverted comma is introduced at the onset of each sentence;
- Another inverted comma then appears at the termination of most sentences as well;
- This is then substituted with a “(3)” at the start of each sentence;
- The “(3)” is itself eventually being replaced by the sequence of characters “(b)”.

This phase, primarily characterised by superficial degradation in quality, gives way to a more pronounced collapse. In fact, in the next evaluation, a significant portion of the sentences is rendered in parentheses followed by a repeated letter “k”. At this point, the collapse is almost complete, and most of the translations generated are identical to this series of “k”. However, it is surprising that some sentences in the development set are still translated.

Recovery

Remarkably, this phase of gradual decline vanishes in the subsequent batch. This shows that the training process is robust and can withstand even major disturbances to one of the two generators. This also shows the importance of accurately monitoring the accuracy achieved, to avoid stopping the training during such a drop in quality.

4.5.2 Results on the WMT24 Main Challenge

As has been the case every year since 2006, the general task at WMT is to pit teams against each other to train translation models for several pairs of languages, which are trained in the same dataset and then compared in the same set of tests. The systems are then ranked according to their performance in several automatic evaluation tools such as BLEU and COMET, as well as by human evaluators.

At the time of writing the initial version of this thesis, the EMNLP 2024 conference and the results of WMT24 were not yet available, which initially prevented a comparison

TABLE 4.26: Generated development translations at specific batch indexes

[illegible]

between CycleGN and models developed by other teams. Consequently, the results presented earlier were computed independently and based on internal benchmarks.

Since then, the WMT24 results have been released [181], allowing for a more thorough comparison. It is important to note that, given that CycleGN is based on unsupervised learning, it is expected to underperform compared to models trained on large-scale parallel datasets, which aim for the highest translation quality. The following provides an updated comparison between CycleGN and state-of-the-art models based on the latest WMT24 benchmarks, while retaining the earlier findings for context and reproducibility.

The diversity of language pairs highlights the variability in the results achieved, including cases where the framework fails to reach an acceptable level of translation. In fact, the framework does not consistently achieve satisfactory translation quality across all language pairs.

Training Limitations

Since training a CycleGN model is particularly costly, there is a trade-off between training models on all language pairs, or choosing a subset of these pairs to train fewer models with more iterations and on a greater number of examples. In order to demonstrate the effectiveness of CycleGN on a wide range of language pairs, the first option was selected, i.e. to train models on all pairs, even if this means obtaining inferior results.

Unlike the previous edition [178], where most language pairs were bidirectional, i.e. the evaluations were to and from, the WMT 2024 General Translation task is unidirectional.

This means that for each pair of languages, it is sufficient to train a model that translates from the source to the target. This is not a change that is favourable for CycleGN, since it is a bidirectional training architecture. Indeed, its cyclical nature means that one model must be trained from one language to another, and another model must complete the cycle, i.e. from this second language to the first. In other words, half the time spent training CycleGN is spent training a model which only serves to train the first, but which will never be evaluated in the contest. Moreover, there was an increase in the number of language pairs, from 6 bidirectional and 2 unidirectional in 2023 to 11 unidirectional in 2024.

During the [WMT24](#) challenge, the teams receive monolingual datasets. They can be used, for example, to continue training a model that is no longer progressing on the parallel dataset via back-translation. In this way, by automatically translating a monolingual dataset, a parallel dataset is obtained which can be used to synthetically increase the dataset and add new training samples, which can improve [Machine Translation](#) quality [88].

Although monolingual datasets are perfectly suited for CycleGN training, they have been discarded for two reasons. The first is that, for the majority of the language pairs, the datasets supplied have been truncated to reduce training time. The second is related to the construction of permuted and non-intersecting datasets, as it is preferable to build them from non-parallel datasets, as detailed in [Section 4.1.2](#).

The datasets were truncated to obtain a maximum of 27,801,496 sentences for training and 100,000 sentences for the development set. The final size of the datasets used and the number of epochs are shown in [Table 4.27](#) for permuted models and [Table 4.28](#) for non-intersecting models. Although the permuted models could all be trained, this was not the case for the non-intersecting models, due to lack of time.

TABLE 4.27: Comparison between the number of sentences available in the [WMT24](#) dataset and the number of sentences used to train the permuted models depending on the language pair

Language pair	Parallel sentences in WMT24 dataset	Sentences used	Number of epochs
Czech-Ukrainian	10,757,756	10,657,756	1
English-Chinese	55,216,751	27,801,496	1
English-Czech	56,288,239	27,801,496	1
English-German	295,805,439	27,801,496	1
English-Hindi	315,070	314,070	10
English-Icelandic	23,434,361	23,334,361	1
English-Japanese	33,875,119	27,801,496	1
English-Russian	75,961,169	27,801,496	1
English-Spanish	626,076,911	27,801,496	1
English-Ukrainian	16,062,359	15,962,359	1
Japanese-Chinese	22,642,571	22,542,571	1

TABLE 4.28: Comparison between the number of sentences available in the [WMT24](#) dataset and the number of sentences used to train the non-intersecting models depending on the language pair

Language pair	Parallel sentences in WMT24 dataset	Sentences used	Number of epochs
English-Chinese	55,216,751	17,676,442	1
English-Czech	56,288,239	27,801,496	1
English-German	295,805,439	27,801,496	1
English-Russian	75,961,169	27,801,496	1

Training Time

Although the [WMT24](#) competition provides teams with parallel datasets, favouring models that can take advantage of them, CycleGN was nevertheless trained on all the language pairs available in its permuted configuration, and four language pairs in its non-intersecting configuration.

To make it possible to train so many models, several machines were used, with different technical characteristics, in particular different GPUs. However, by estimating the training time according to the number of rows in the dataset and the GPU used, the total training time for all these models represents approximately 3,700 hours on an NVidia 4090.

Estimated COMET scores

To calculate the [COMET](#) scores, the last 10,000 lines of each dataset were extracted and removed from the training data to create a parallel dataset, enabling the generation of target outputs to compare against the model hypotheses. As during the prototyping phase, a development set was also extracted and translated every 1,000 batches to monitor the progression of [COMET](#) scores throughout training.

Table 4.29 shows the [COMET](#) scores obtained by all CycleGN models trained on the [WMT24](#) dataset. It is crucial to note that these are not the official results, but the scores obtained on a subset of 10,000 sentences, extracted from the training sets. The hypothesis formulated in Section 4.2, suggesting that models trained on permuted datasets would achieve better translations than those trained on non-intersecting datasets, is confirmed across all four language pairs for which non-intersecting models were trained.

TABLE 4.29: [COMET](#) scores obtained by CycleGN models depending on the language pair and the dataset type on the [WMT24](#) dataset. Note that these are not official [WMT24](#) results, as the official scores have not yet been released. The scores shown are based on a non-official test set extracted from the training data

Language pair	Source to Target (COMET)	Target to Source (COMET)
Czech-Ukrainian (permuted)	0.528	0.460
English-Chinese (permuted)	0.425	0.439
English-Chinese (non-intersecting)	0.325	0.378
English-Czech (permuted)	0.364	0.415
English-Czech (non-intersecting)	0.395	0.461
English-German (permuted)	0.604	0.638
English-German (non-intersecting)	0.595	0.615
English-Hindi (permuted)	0.474	0.564
English-Icelandic (permuted)	0.479	0.476
English-Japanese (permuted)	0.412	0.423
English-Russian (permuted)	0.384	0.402
English-Russian (non-intersecting)	0.341	0.387
English-Spanish (permuted)	0.612	0.627
English-Ukrainian (permuted)	0.439	0.453
Japanese-Chinese (permuted)	0.482	0.481

The instabilities observed during the CycleGN prototyping phase are present across numerous language pairs, indicating that these issues are not specific to the English-German language pair. This instability represents one of the primary limitations of

CycleGN. To maintain conciseness, the evolutions of the COMET scores will be presented for only a subset of language pairs, as including all would be excessively lengthy.

Between the WMT23 and WMT24 editions, the dataset for the English-German pair remained unchanged. Thus, the model trained during the CycleGN prototyping phase in a subset of WMT23, whose COMET score progression is shown in Figure 4.23, was also used for WMT24.

Figures 4.25 and 4.26 illustrate the COMET score progression, respectively for the permuted and for the non-intersecting models. Unlike training the permuted model, where the performance of the two generators is very close and progresses together, when training the non-intersecting model the performance no longer progresses from batch 220, mainly due to a significant performance gap between the two generators, which is observed from the very first batches.

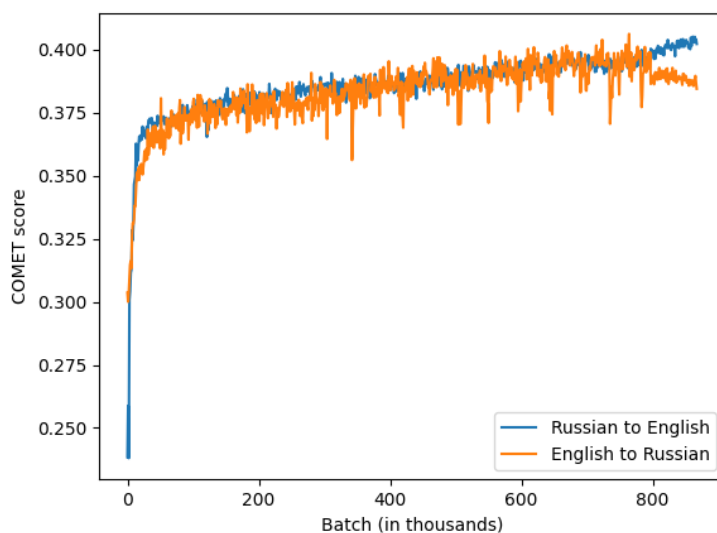


FIGURE 4.25: Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT24 English-Russian language pair

This result can also be seen for the English-Chinese language pair, where Figures 4.27 and 4.28 show the evolution of the COMET score on a permuted dataset and a non-intersecting dataset respectively.

Figure 4.29 shows the instabilities encountered when training the English-Spanish model on a permuted dataset. Despite some particularly marked collapses, the model continues to progress until the end of the training.

As shown in Figure 4.30, in the English to Icelandic direction, large instabilities are observed between batches 175 and 300, preventing more progress in the Icelandic to English direction.

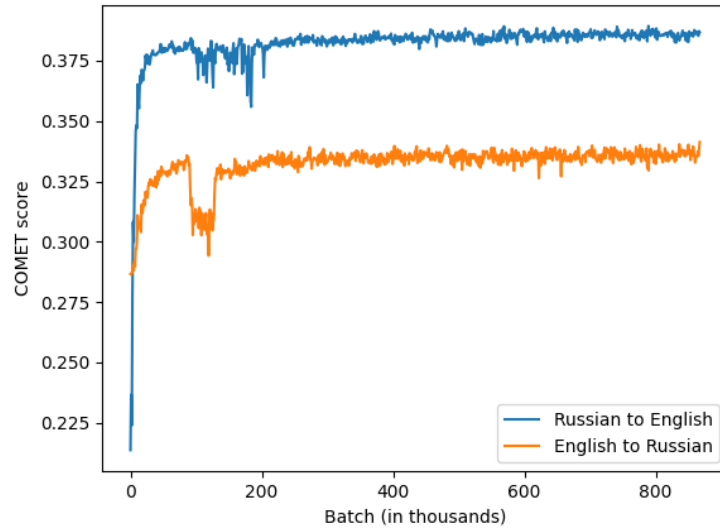


FIGURE 4.26: Evolution of the [COMET](#) score during the non-intersecting training of the CycleGN framework on the [WMT24](#) English-Russian language pair

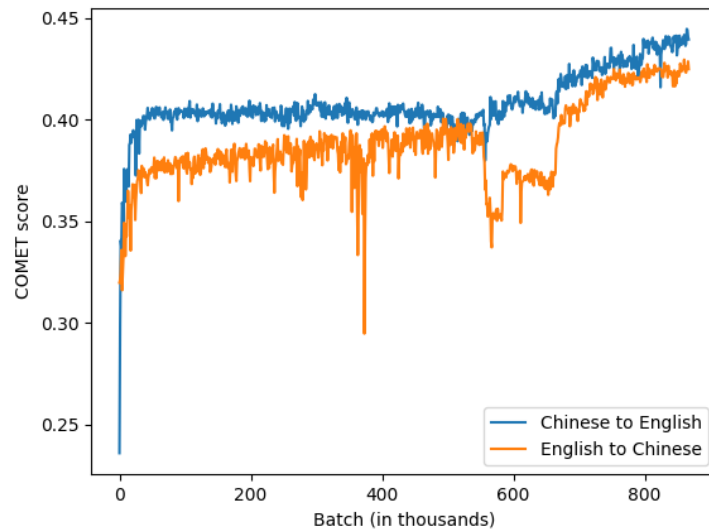


FIGURE 4.27: Evolution of the [COMET](#) score during the permuted training of the CycleGN framework on the [WMT24](#) English-Chinese language pair

Official Results

The official mtme tool⁹ was used to obtain the [COMET](#) for each language pair. Table [4.30](#) presents the results for all models evaluated across all language pairs, as well as CycleGN in its non-intersecting modality, which could not be trained on every pair due to time constraints. As expected, the performance of CycleGN is lower than that of other models, which benefited from training on parallel datasets and employed significantly larger architectures.

⁹<https://github.com/google-research/mt-metrics-eval>

TABLE 4.30: Model performance across language pairs (caption not rotated)

model	cs-uk	en-cs	en-de	en-es	en-hi	en-is	en-ja	en-ru	en-uk	en-zh	ja-zh
CycleGN (permuted)	0.292	0.346	0.420	0.336	0.351	0.437	0.382	0.369	0.372	0.379	0.369
CycleGN (non-intersecting)		0.402	0.420					0.340		0.320	
Aya23	0.866	0.828	0.815	0.824	0.740	0.533	0.862	0.813	0.833	0.823	0.814
Claude-3.5	0.905	0.873	0.833	0.842	0.787	0.817	0.878	0.837	0.865	0.849	0.852
CommandR-plus	0.883	0.854	0.830	0.837	0.757	0.657	0.871	0.833	0.855	0.840	0.831
GPT-4	0.882	0.861	0.835	0.849	0.764	0.797	0.873	0.837	0.857	0.849	0.841
IKUN	0.868	0.829	0.806	0.815	0.694	0.796	0.846	0.813	0.839	0.819	0.791
IKUN-C	0.859	0.821	0.796	0.811	0.720	0.788	0.834	0.800	0.816	0.809	0.762
IOL_Research	0.881	0.853	0.821	0.838	0.768	0.768	0.862	0.823	0.853	0.849	0.837
Llama3-70B	0.857	0.824	0.813	0.830	0.754	0.719	0.851	0.812	0.829	0.824	0.820
Mistral-Large	0.870	0.837	0.827	0.837	0.687	0.646	0.845	0.824	0.847	0.818	0.808
Phi-3-Medium	0.664	0.584	0.797	0.818	0.612	0.513	0.850	0.798	0.600	0.815	0.797
Unbabel-Tower70B	0.903	0.888	0.849	0.857	0.800	0.845	0.889	0.859	0.882	0.868	0.855

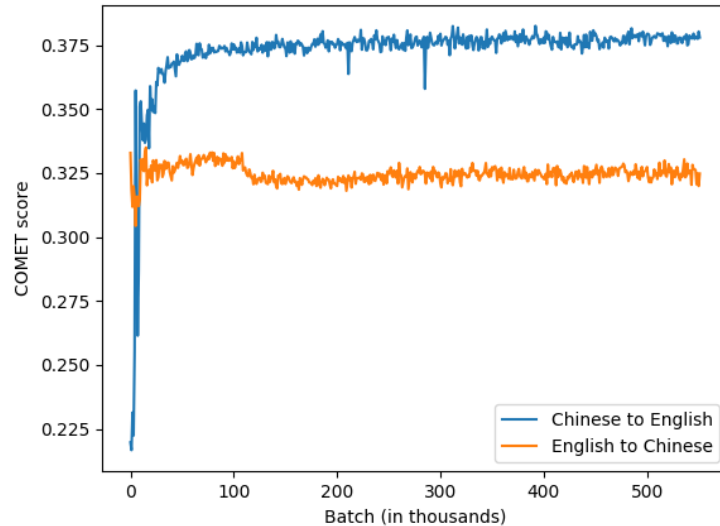


FIGURE 4.28: Evolution of the [COMET](#) score during the non-intersecting training of the CycleGN framework on the [WMT24](#) English-Chinese language pair

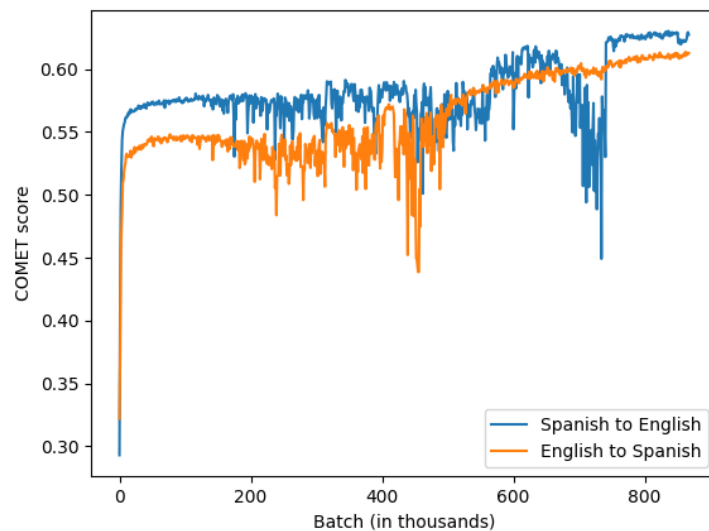


FIGURE 4.29: Evolution of the [COMET](#) score during the permuted training of the CycleGN framework on the [WMT24](#) English-Spanish language pair

4.5.3 Low-Resource Languages

In addition to having, by definition, only a limited amount of parallel translations, some low-resource languages also suffer from a scarcity of monolingual datasets. Although CycleGN does not require parallel data, its framework still relies on a substantial amount of monolingual data. As a result, it is not a suitable approach for unsupervised [Machine Translation](#) training when monolingual data is insufficient.

While the main [WMT](#) challenge, discussed in Section 4.5.2, consists of training translation models using datasets containing tens or hundreds of millions of parallel sentences, low-resource tasks focus on languages where resources are scarce. The first [WMT](#) to

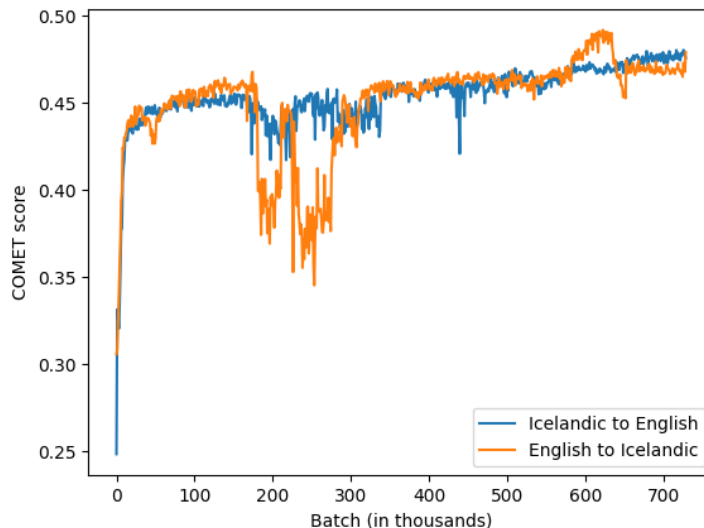


FIGURE 4.30: Evolution of the [COMET](#) score during the permuted training of the CycleGN framework on the [WMT24](#) English-Icelandic language pair

offer this challenge took place in 2020 [98] and consisted of translating German into Upper Sorbian and vice versa—Upper Sorbian being a Slavic language related to Czech.

With the exception of Icelandic, the languages in the main challenge are spoken by several million people. They are all national languages of at least one country, making official translations of international treaties accessible. Furthermore, 9 of the 11 language pairs have English as their source language, which is the most widely used language and has the most resources.

By way of contrast, low-resource languages are usually regional, and may have state-recognised status, but are not used on a national scale. Parallel sentence examples are often limited to just tens of thousands. In the 2024 edition of the [WMT](#), two different tasks are labelled as “low-resource”.

Indic languages encompass a wide range of different languages, which are collectively spoken by more than 800 million people, mainly in India, Bangladesh, Sri Lanka, Pakistan, and Nepal [43]. The “Low-Resource Indic Language Translation” task covers four different language pairs, with English present in all of them:

- Assamese is an official Indo-Aryan language of the Assam state in India and has over 15 million native speakers [382];
- Mizo is an official Tibeto-Burman language of the Mizoram state in India [388];
- Khasi is an Austroasiatic language [386];
- Manipuri, also known as Meitei, is the official language of the Manipur state in India and belongs to the Tibeto-Burman family [387].

The other low-resource task, “Translation Into Low-Resource Languages of Spain”, consists of languages native to the Iberic region.

- Aragonese is a Romance language spoken by around 10,000 people [380];
- Aranese is a variety of Gascon, which itself is one of the two main dialects of Occitan and is one of the official language of Catalonia, a province in Spain, [381];
- Asturian is a Romance language natively spoken by more than 100,000 people [383].

Due to the computationally expensive nature of CycleGN, we could not train models in all seven pairs of languages. The models were trained in the order that they appeared on the WMT24 website. Table 4.31 shows the six language pairs that were selected, along with the number of lines in the monolingual datasets and the number of epochs.

TABLE 4.31: Number of sentences for each language pair and number of epochs during training

Language Pair	Number of lines	Number of epochs
Spanish-Aragonese	84,703	10
Spanish-Asturian	38,869	10
English-Assamese	2,624,715	1
English-Khasi	182,737	3
English-Manipuri	2,144,897	1
English-Mizo	1,909,823	1

Contrary to what we had hoped for on the basis of previous results obtained, the trained models did not learn as expected. Although the [Cycle-Consistency Loss](#) is supposed to progressively decrease for both models $\mathcal{G}$ and $\mathcal{F}$, this was not the observed behaviour.

We identified two reasons for this discrepancy, which also corresponds to the two tasks in the WMT24 low-resource category. The first is an imbalance of classes; that is, completing the cycle is more difficult in one direction than in the other, and the second is a high linguistic proximity, where two languages are so similar that the [Cycle-Consistency Loss](#) stays low even when the sentence is not modified by the generators.

Class Imbalance

The term “class imbalance” in [Machine Learning](#) is polysemic. Here, it refers to the fact that one of the two classes, or languages, requires one of the generators to encode more information than the other. This would be analogous to training the original CycleGAN model on a dataset where the images for the first class are considerably larger than the images of the other class.

Figures 4.31, 4.32, 4.33 and 4.34 display the respective evolution of the Cycle Consistency Loss during the training of the language pairs English-Assamese, English-Khasi, English-Manipuri and English-Mizo.

This class imbalance can be observed by looking at the median lengths of the sentences depending on the language. For example, in the English-Khasi dataset, while the English sentence has a median length of 82 characters, the median Khasi sentence contains 304 characters. Longer sentences in Khasi imply greater linguistic complexity or verbosity,

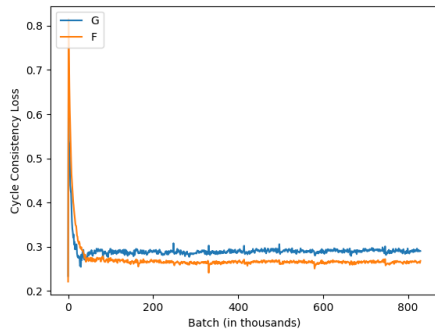


FIGURE 4.31: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) English-Assamese language pair

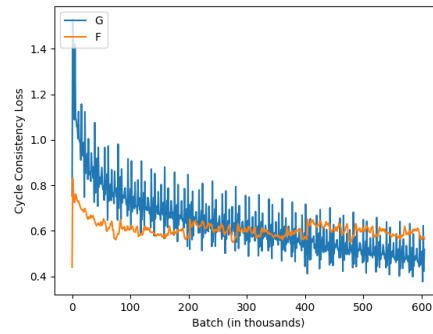


FIGURE 4.32: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) English-Khasi language pair

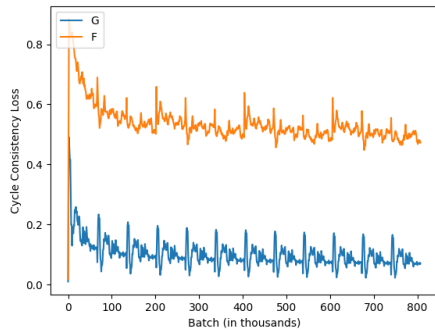


FIGURE 4.33: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) English-Manipuri language pair

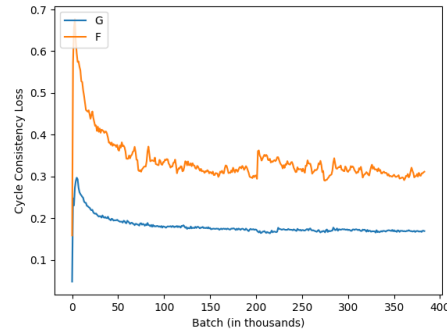


FIGURE 4.34: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) English-Mizo language pair

requiring the network to encode and decode disproportionately more information for one language compared to the other. With imbalanced lengths, the CycleGN may either overfit to the shorter language or fail to generalise for the longer one.

Addressing this issue might involve strategies like sentence-length normalisation, alternative tokenization methods that equalise token counts, or even data augmentation techniques to create synthetic balance.

The other difficulty encountered when training low-resource languages is quite different, since it concerns languages that are too similar.

High Semantic Proximity

As in the Indic languages, CycleGN was unable to learn from the datasets provided. However, it was not due to an imbalance of classes in this case, but rather because the classes were too close together, since the Iberian languages are very close to the

source language, Spanish. Figures 4.35 and 4.36, respectively, show the evolution of the [Cycle-Consistency Loss](#) on the Spanish-Aragonese and Spanish-Asturian language pairs.

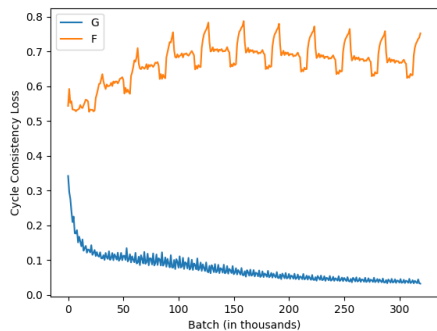


FIGURE 4.35: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) Spanish-Aragonese language pair

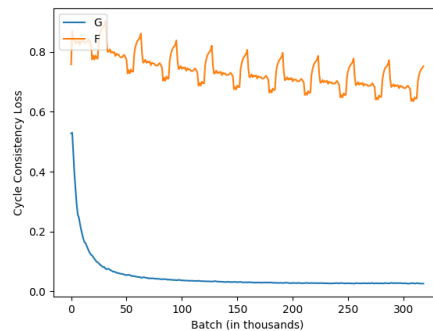


FIGURE 4.36: Evolution of the [Cycle-Consistency Loss](#) during the permuted training of the CycleGN framework on the [WMT24](#) Spanish-Asturian language pair

For both of these languages, one of the generators does not escape the global minimum and learns not to change its input sentence at all. Since the source and target languages are so semantically similar, there is not a sufficient divergence for learning. The structural and lexical overlap between Spanish and these Iberian languages limits the distinct features that the network can learn, hindering the development of nuanced translation capabilities.

A potential solution involves translating through a third, more distant, language. As we already know from the results on the [WMT24](#) datasets, it is possible to successfully train a CycleGN English-Spanish model. Thus, by training two CycleGN models, where the first one maps Spanish to English and the second one English to the target Iberian language, it would be possible to successfully map Spanish to the target Iberian language through an intermediate representation, namely English.

The identified issues, particularly in relation to both class imbalance and semantic proximity, indicate that further refinement and investigation are necessary to enhance the capabilities of CycleGN for low-resource languages.

We demonstrated that CycleGN is capable of learning from permuted datasets as well as non-intersecting datasets. Training conducted on low-resource languages revealed additional challenges associated with the CycleGN learning process. Addressing these challenges could potentially improve performance not only for low-resource languages but also for other language pairs. It is now essential to compare CycleGN with other frameworks for unsupervised [Machine Translation](#) to evaluate its capabilities relative to the current state-of-the-art. Unfortunately, previous research on unsupervised [Machine Translation](#) has not been conducted on low-resource languages.

4.6 Comparison with Existing Frameworks

In recent years, three prominent frameworks have emerged for unsupervised [Neural Machine Translation](#): [XLM](#), [MASS](#) and [CBD](#). This prior work is compared with CycleGN to evaluate the quality of its translations. However, none of these studies used non-intersecting datasets, relying exclusively on monolingual datasets. Although some models, such as [MASS](#), have demonstrated effectiveness in tasks beyond unsupervised [Machine Translation](#), such as text summarization and supervised [Machine Translation](#), this comparison focusses solely on their performance in unsupervised [Machine Translation](#).

Previous work [17] has demonstrated that a framework similar to CycleGAN can be utilized for training [NMT](#) models. Instead of adopting the identity loss from CycleGAN, they introduce a language model loss, which estimates the per-word entropy in the target language corpus using an n -gram language model and penalises higher per-word entropies in machine-translated text. However, this approach was not included in our comparison, as its [BLEU](#) scores for the English-French, French-English, English-German, and German-English translation tasks—36.2, 33.5, 26.9, and 34.4, respectively—were lower than those achieved by the previously discussed methods.

Although [LLMs](#) are highly effective in translation tasks, as discussed in Section 2.5.3, they are excluded from this analysis due to fundamental differences in their training paradigm and architecture. Unlike dedicated [Neural Machine Translation](#) models, which are explicitly optimised for sentence-level translation quality, [LLMs](#) are trained on vastly broader objectives, including general-purpose language modeling, instruction following, and dialogue-based interactions. This distinction makes a direct comparison methodologically problematic, as these models may rely on memorisation rather than true generalisation, and their translation performance is often dependent on prompt engineering rather than dedicated fine-tuning.

Additionally, [LLMs](#) are trained on significantly larger datasets and require substantially more computational resources, leading to higher training and inference costs. Given these differences, a fair comparison would require alternative evaluation methodologies, such as assessing performance across varying prompt structures or model fine-tuning, which is beyond the scope of this work.

The [Cross-lingual Language Model Pretraining \(XLM\)](#) [69] framework from FacebookAI leverages monolingual corpora for pre-training and subsequent training to achieve unsupervised [Machine Translation](#). [XLM](#) separates the pre-training of the encoder and the pre-training of the decoder. The encoder is trained using the [Masked Language Modeling](#) objective, similarly to CycleGN, where it learns to reconstruct sentences where some tokens are removed or masked. This step aligns word and sentence representations across languages, enabling a shared embedding space that facilitates downstream translation tasks. In the case of a translation between two languages from cultures where the cat is a common domestic animal, even in the absence of parallel data, because the use of the word cat is similar between the two languages, the representation in the embeddings space will be similar.

The decoder, on the other hand, is trained in a [CLM](#) task, where it learns to predict the next token from past ones, similarly to [LLMs](#) such as [GPT](#). This ensures that the model is capable of generating coherent sentences.

During training for unsupervised [Machine Translation](#), [XLM](#) employs denoising auto-encoding and backtranslation techniques. Denoising auto-encoding reconstructs original sentences from corrupted versions, such as shuffled words or missing tokens, enabling the model to generate coherent outputs despite noisy inputs. Back-translation extends this by iteratively translating sentences between two languages. Sentences in one language are translated into another, and the translated output is reconstructed back into the original language. This approach dynamically generates translations during training, enabling the model to refine its translation capabilities iteratively and adaptively by minimising reconstruction errors.

[MAssed Sequence to Sequence pre-training \(MASS\)](#) [331] is a Microsoft framework, built on top of [XLM](#), in which the pre-training of the encoder and the decoder is unified over a special form of [Masked Language Modeling](#), in which multiple consecutive tokens are masked and the model is tasked with retrieving all of them. Masking consecutive tokens forces the decoder to learn better contextual representations.

Contrary to the training of [XLM](#), the [MASS](#) training solely relies on backtranslation and not on denoising auto-encoding.

[Cross-model Back-translated Distillation \(CBD\)](#) [244] is built on top of both [XLM](#) and [MASS](#) and relies on two models, similar to CycleGN, rather than a single model that can translate backward and forward. This creates diversity in the synthetic data generated for backtranslation, since the data used to train a model have been generated by the other, and vice versa. These “corrupted” sentences, as the authors refer to them, generate new linguistic insights and thus create new information.

The original [CBD](#) work does not specify any specific pre-training method, but relies on the pre-trained models from [XLM](#) and [MASS](#), which were still available at the time.

At the time of writing, [CBD](#) is the State-of-the-Art unsupervised [Machine Translation](#) technique, excluding [LLMs](#).

4.6.1 CycleGN Training

As [XLM](#) was trained on permuted data, and as training a model takes a particularly long time on such a large dataset, only permuted datasets were used in the research work presented here to train CycleGN models. Of the three language pairs present in the original work, only two were chosen due to training time limitations, English-French and English-German. As each of these training sessions took several weeks, it was not possible to train the English-Romanian model.

The English-French CycleGN model was trained over 27 days to achieve a single epoch. Although it is possible that an additional epoch could have continued to improve the

model’s performance, spending a further 27 days on a second epoch would have limited the number of models trained on WMT24 and therefore the diversity of languages examined. Figure 4.37 shows the evolution of the COMET score during training.

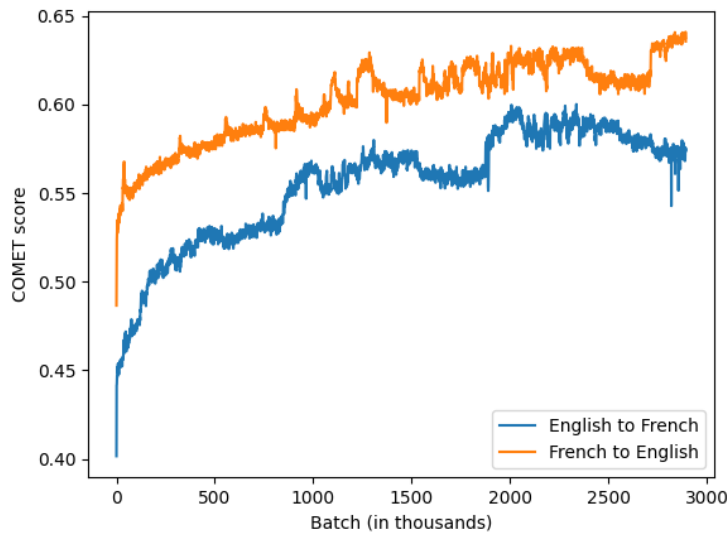


FIGURE 4.37: Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT14 English-French language pair

Since the dataset used to train the English-German model comes from WMT16, rather than WMT14 as for the English-French pair, it contains many more sentences. As a result, training for a single epoch took 41 days. Figure 4.38 shows the evolution of the COMET score during training.

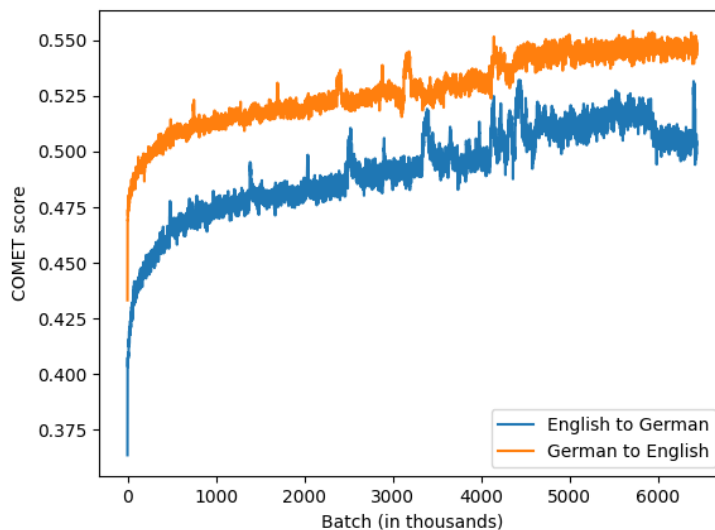


FIGURE 4.38: Evolution of the COMET score during the permuted training of the CycleGN framework on the WMT16 English-German language pair

4.6.2 Reproducibility

The original publications relied on the [BLEU](#) score to evaluate the models. This metric heavily penalises CycleGN due to its confusions, as explained in Section 4.4.7, and the [COMET](#) score is a better and more recent metric than [BLEU](#), which obtains much better correlations with human evaluations, as detailed in Section 2.5.2.

However, none of the three models that we compared against provided the sentences generated during model evaluation, making it impossible to calculate the [COMET](#) score on these models. Furthermore, none of these three frameworks provides fine-tuned models. Thus, it is essential to train to reproduce these models, generate the test sentences, and calculate the [COMET](#) score of the [XLM](#), [MASS](#) and [CBD](#) models.

However, the reproducibility of previous work was made difficult by a phenomenon known as software “decay”, or software “ageing” [263]. This is the gradual degradation of software reliability over time due to evolving requirements and environmental changes, even without modifications to the software itself. This can be caused by backward-incompatible changes in programming languages, libraries, or even operating systems, and in some cases hardware changes, such as changes in the instruction set architectures.

Although the official [XLM](#) code repository¹⁰ does not provide trained translation models, it is possible to download pre-trained models and vocabularies, and the repository offers the code necessary to train the checkpoints in unsupervised [Machine Translation](#). In order to reproduce the original work, the instructions were followed to train two models, the English-French one and the English-German one. The training parameters were all kept identical with the instructions provided, and the original vocabularies and pre-trained models were used.

To reproduce the work precisely, we needed to modify the *N_MONO* variable in the *get-data-nmt.sh* file to increase the number of sentences per language from 5 million to 265 million for German, 200 million for English and 65 million for French¹¹.

As in the original work, the training was stopped when there was no increase in the [BLEU](#) score after 10 consecutive epochs. This took 26 epochs for the English-French model and 39 epochs for the English-German model.

However, even after making these modifications to use the entire dataset and performing the training with the parameters provided in the repository, the results obtained by the model do not correspond to what was obtained in the original work, as for all four models, the scores obtained by our trained models are inferior to those found in the original work. Table 4.32 reproduces the [BLEU](#) scores obtained from Table 2 of the original [XLM](#) publication and compares these scores with the one obtained by the reproduced models.

Although the official repository¹² of [MASS](#) provides links to download pre-trained and

¹⁰<https://github.com/facebookresearch/XLM>

¹¹This is not specified in the repository, but it is also necessary to uncomment the lines that download additional fragments of the dataset to download the whole of [WMT14](#) for training the English-French model and the whole of [WMT16](#) for training English-German

¹²<https://github.com/microsoft/MASS>

TABLE 4.32: Comparison between the BLEU scores obtained in the original XLM publication and the reproduction

Language Direction	Original Work (BLEU)	Reproduction (BLEU)
English → French	33.4	33.1
French → English	33.3	31.8
English → German	26.4	25.8
German → English	34.3	30.8

fine-tuned models, the links have not worked since at least 2023 according to a GitHub issue¹³. Similarly, the email address listed in the original publication for correspondence is no longer assigned and can no longer receive emails. Due to a lack of resources, it has not been possible to retrain both the English-French and English-German models, and only the former has been both pre-trained and fine-tuned.

The steps required to reproduce MASS are identical to those for reproducing XLM, with a few additional modifications. These adjustments were necessary because the provided *get-data-nmt.sh* and *evaluator.py* scripts are not compatible with *bash*, the default shell for Ubuntu and most GNU/Linux distributions.

It was also necessary to modify the parameters of the training script compared to those provided in the repository. The parameters specified in the repository terminate the training after 30 epochs, at which point the model achieves BLEU scores below 24, far from the results reported in the original publication. For this thesis, we adopted the training parameters used in XLM, specifically halting training if the BLEU score does not improve for 10 consecutive epochs.

The pre-training process required 100 epochs, while the fine-tuning phase required 34 epochs. However, as with the reproduction of XLM, the BLEU scores achieved are not similar to those of the original publication. Table 4.33 shows the reproduction scores, which are even lower than the reproduction scores of XLM.

TABLE 4.33: Comparison between the BLEU scores obtained in the original MASS publication and the reproduction

Language Direction	Original Work (BLEU)	Reproduction (BLEU)
English → French	37.5	21.2
French → English	34.9	23.3

Similarly to the reproduction of the MASS work, the fine-tuned CBD English-French model is no longer available for download in the official repository¹⁴ and we did not receive a response when contacting the original authors. The English-German model, obtained from the repository, required modifications to update the parameter key names, as they did not match. This mismatch was likely due to the absence of dependency pinning of PyTorch.

We were unable to reproduce the training of an English-French CBD model; therefore, only the results for the English-German language pair are presented. We wish to highlight that, although the model is labelled as English-French in the repository, it is only

¹³<https://github.com/microsoft/MASS/issues/183>

¹⁴https://github.com/nxphi47/multiagent_crosstranslate

upon downloading and executing the model that it becomes apparent that it is, in fact, an English-German model. Table 4.34 shows the BLEU scores obtained by this model in the original publication. This discrepancy indicates a lack of quality control in the repository documentation or file organisation, which undermines reproducibility efforts.

TABLE 4.34: BLEU scores obtained in the original CBD publication

Language Direction	Original Work (BLEU)
English → German	30.1
German → English	36.3

This inability to reproduce the results obtained when training the XLM and MASS models could be due to differences in the versions of the libraries used, since not all dependencies are pinned, or to differences in hardware.

Further discussion of reproducibility and some paths toward solutions are offered in the Conclusion, in Section 7.4.

4.6.3 Results

Although CycleGN models were trained in a single epoch and the XLM models benefited from several dozen epochs, the performance as measured by the COMET metric shows an advantage for CycleGN over XLM on unsupervised Machine Translation. Table 4.35 shows the difference in the number of parameters between the CycleGN, XLM, MASS and CBD models.

TABLE 4.35: Comparison between the number of parameters of CycleGN, XLM, MASS and CBD

CycleGN	XLM	MASS	CBD
317,538,304	308,899,094	315,196,694	375,772,534

However, there was an important difference in the training time of the XLM model, which took less than 24 hours to complete, and of the MASS model, which required 72 hours, compared to several weeks for the CycleGN models. This difference in training duration is attributed to two factors: first, the XLM and MASS models preprocess sentences through tokenization, with the time required for this step excluded from the training duration; and second, CycleGN employs four distinct loss functions—one Cycle-Consistency Loss and Identity Loss for each of its two models—for every batch.

Table 4.36 shows the COMET scores obtained by the reproduced XLM models, the reproduced English-French MASS models, the downloaded English-German CBD model, and the trained CycleGN models. Similarly to the results computed in Section 3.2.4, as the translated sentences are paired, and there is no reason to assume that the COMET scores follow a normal distribution, the Wilcoxon signed-rank test. The results obtained by CycleGN are promising as they outperform those achieved by previous models. However, as the reproduced models fell short of the performance levels reported in scientific publications, further research is needed to validate these findings.

TABLE 4.36: Comparison between the COMET scores obtained by the CycleGN models and XLM, MASS and CBD

Language Direction	CycleGN (COMET)	XLM (COMET)	MASS (COMET)	CBD (COMET). * indicates statistically significant difference
English → French	0.573*	0.493	0.468	NA
French → English	0.639*	0.542	0.558	NA
English → German	0.499*	0.482	NA	0.498
German → English	0.550*	0.541	NA	0.544

The BLEU score remains the most commonly used metric to evaluate the performance of unsupervised Machine Translation models. However, the title of the WMT22 Metrics Task publication [99], “Stop Using BLEU – Neural Metrics Are Better and More Robust”, highlights the need to adopt more modern evaluation metrics. Future research in the field should prioritise the use of these advanced metrics.

In addition, to improve reproducibility, trained models should be made publicly available for download. As an initial straightforward step, researchers could at least provide the translations generated by their models on the test set. This would allow future studies to compare models using different metrics and foster progress in the field.

We examined how CycleGN performs unsupervised Machine Translation and analysed its performance relative to competing frameworks.

4.7 Limitations

CycleGN, as an unsupervised translation framework, exhibits several limitations that affect its reliability and scalability across different language pairs. Although it performs well with certain language pairs, it does not produce satisfactory results with others. One significant challenge is the instability of the model during training. This instability is evident in the progression of the COMET score in the development set, as the score fluctuates markedly throughout training. Such variability requires frequent model checkpointing to avoid substantial performance degradation, as well as continuous monitoring to ensure that training does not conclude at a point of poor performance. This need for intensive oversight makes the training process inefficient and resource-intensive.

Predicting which language pairs CycleGN will effectively translate remains difficult. For instance, the model struggles with closely related languages, such as Spanish and Asturian, as well as with pairs where there is a substantial token imbalance between source and target sentences. The latter issue resembles a class imbalance problem, where the architecture requires a balanced distribution of tokens across the paired languages for optimal performance. Consequently, CycleGN’s architecture is fragile, as it depends on a precise balance of input conditions to function effectively. These limitations highlight the need for improvements to increase the framework’s stability and broaden its applicability across diverse language pairs.

4.8 Future Work

Future research efforts will focus primarily on enhancing the stability and efficiency of the CycleGN training process. Addressing training instability is crucial as it would enable more reliable performance across a wider range of language pairs. One promising approach is to increase the batch size, which has the potential to smooth training dynamics and reduce the likelihood of erratic fluctuations in model performance. Additionally, optimising the framework to accelerate training times is essential as it would make it feasible to scale CycleGN to larger datasets and to train over more epochs. This, in turn, could significantly improve [COMET](#) scores by allowing the model to achieve deeper, more robust learning and generalisation across languages. These improvements are expected to expand the usability and effectiveness of CycleGN for a wider spectrum of translation tasks.

4.8.1 Gradient Accumulation

As mentioned in [Section 4.4.3](#), batch size is an important hyperparameter in CycleGN training, since a size of one does not allow convergence, and as demonstrated in [Section 4.4.6](#), smaller models, which can use a larger batch size, have a smoother progression. Gradient descent, discussed in [Section 2.1.4](#), can be used to artificially increase the batch size by accumulating gradients over multiple batches before performing a single backpropagation step. Instead of updating the model parameters after every batch, the gradients are accumulated over several batches and only backpropagated once a predefined number of batches have been processed.

4.8.2 Model Architecture

Given that training on a single language pair is costly and requires several days or even weeks, and that model configurations were kept constant during training for [WMT24](#) to allow for easier comparisons, it was not possible to fine-tune the parameters to find optimal trade-offs. Future research should therefore explore the effects of certain modifications in model configurations.

Layers

Marian’s default configuration for the HuggingFace library implements 12 layers for the encoder and a further 12 for the decoder, each with 16 heads and a size of 4096. In addition, the vocabulary size is 50,000. The current number of parameters in the models trained in this work is 158,769,152 per model. With this new augmented architecture, this number would rise to 412,210,176, that is, an increase of more than 150%. However, this would significantly increase the training time for the models, which is already particularly time-consuming.

Learning Rate and Scheduler

In the current work, a learning rate of $1e-4$ was used during training. This seems higher than the default parameters used in the HuggingFace library, which is $5e-5$.

More importantly, it seems that using a cyclic learning rate can improve training. The use of a warm-up, such as the linear warm-up phase presented in Figure 4.39, is common in transformer training and more generally in [Natural Language Processing](#), a warm-up phase being a rapid and significant increase in the learning rate at the beginning of training, before a gradual decrease.

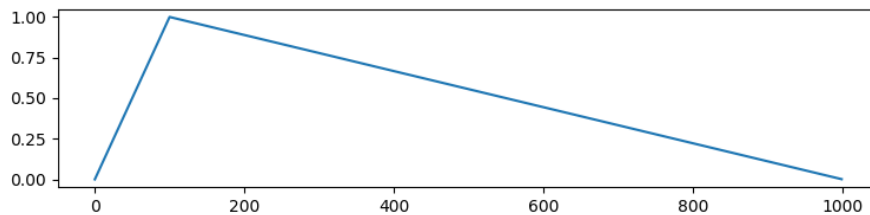


FIGURE 4.39: Linear scheduler with warm-up phase

Bert and GPT2

Further research is needed on a combination between the [BERT](#) and [GPT2](#) architectures. This model may be improved by advances made during the development of CycleGN. Initially, it would be essential to shift the tokens so that identifiers are globally unique, ensuring that each identifier corresponds to only one subword, regardless of language or model. Secondly, pre-training on a [NTP](#) task could enable the framework—starting from more informed parameters rather than random initialisation—to begin learning with a knowledge base that can be derived from a non-parallel dataset.

4.8.3 Quantisation-Aware Training

As the work in Section 4.4.6 demonstrated that post-training quantisation does not compromise the performance of CycleGN, further research should investigate whether using [Quantization-Aware Training \(QAT\)](#) could accelerate model training without degrading performance, thereby enhancing the feasibility of this framework.

4.8.4 Larger Datasets

As CycleGN is a resource-intensive architecture, it was not feasible to exhaustively investigate the impact of dataset size on performance. Future hardware will likely enable faster CycleGN training, making the use of larger datasets more accessible. Furthermore, the training process improvements mentioned above could optimise and shorten the duration of the training.

4.9 Conclusion

CycleGN represents an innovative approach in unsupervised [Machine Translation](#), leveraging a cyclic loss framework to achieve translation across diverse language pairs. Its design enables it to function on both permuted and non-intersecting datasets, demonstrating flexibility in dataset structure. In permuted datasets, where each sentence is paired with its corresponding translation at an unknown index, CycleGN effectively learns translation mappings despite the lack of explicit alignment. Similarly, in non-intersecting datasets, where no sentence is aligned with its translation anywhere in the dataset, the framework has shown adaptability in learning generalised translation patterns.

We showed why different iterations of the framework during the development process failed to learn from non-parallel datasets and detailed the CycleGN architecture in details, including its vocabulary organisation, its different loss functions, its pre-training, and its absence of discriminator.

Although CycleGN achieves promising results, particularly in language pairs that maintain certain structural and lexical similarities, we also discuss the challenges faced. The performance of CycleGN remains variable across different languages and different datasets, and is sensitive to training conditions, which can hinder performance if not managed carefully.

However, we show that CycleGN has the ability to learn from unaligned data, which makes it a valuable addition to the field of unsupervised translation, with the potential for further refinement in stability and scalability to broaden its applicability.

Future work will focus on addressing these limitations, with efforts to improve training stability and efficiency through adjustments in batch size and accelerated processing for larger datasets. These advances are expected to improve CycleGN performance and make it a more robust framework for unsupervised [Machine Translation](#) across an increasingly diverse set of languages and dataset configurations.

How can techniques from image style transfer, such as CycleGAN, be adapted for unsupervised [Machine Translation](#) in the absence of parallel datasets? Techniques from image style transfer, such as CycleGAN, can be adapted for unsupervised [Machine Translation](#) in the absence of parallel datasets by applying the principle of cycle consistency. In the CycleGN framework, two models are trained to translate between two languages in opposite directions using non-parallel data. The core idea is that a sentence translated from the source language to the target and then back (retro-translation) should reconstruct the original sentence. This cycle consistency acts as a supervisory signal to optimise both models without relying on paired examples.

Chapter 5

Unsupervised Style Transfer

The difference between the
almost right word and the right
word is really a large
matter—it’s the difference
between the lightning bug and
the lightning.

Mark Twain

In Chapter 4, we explored unsupervised [Neural Machine Translation](#), highlighting how a cyclic loss approach can be effectively used to train translation models on non-parallel datasets. This approach led to the development of the CycleGN framework, a novel method we created and published that demonstrates the potential of unsupervised models to handle diverse translation tasks. In this Chapter, we explore the text style transfer capabilities of CycleGN. We aim to answer the Research Questions 3. How can style transfer methods originally designed for images be adapted for text style transfer tasks?

5.1 Introduction

As mentioned in Section [2.7.3](#), text style transfer is the task of modifying or adapting the stylistic expression of a sentence while maintaining the original meaning. Similarly to text translation, this is a [Sequence-to-Sequence](#) task, but this time it is intralanguage, as there is no transition from one natural language to another. The measurement of text style transfer focusses on two main points: the preservation of meaning and the transfer itself.

Although CycleGN is first and foremost a model specialising in translation, since the underlying idea behind CycleGN is to consider text translation as a text style transfer problem, CycleGN has been trained on this text style transfer task in order to evaluate its performance.

There are several possible categories of text style transfer, such as modifying offensive text to make it more appropriate for content moderation, rewriting informal text into a formal tone, or simplifying complex text for better readability. The work carried out as part of this thesis is based mainly on two different style transfers, namely formality, that is, the degree of adherence to conventional linguistic norms, tone, and structure typically associated with a specific context or audience and sentiment, which is the emotional tone or attitude of a document.

5.2 Formality

As mentioned in Section 2.7.3, [Grammarly’s Yahoo Answers Formality Corpus](#) [291] is a subset of Yahoo! Answers L6 corpus [407], which contains English pairs of Formal and Informal sentences and is dedicated to text style transfer. XFORMAL [36] is an extension of [GYAFC](#) that offers translations in French, Italian, and Brazilian. These two datasets offer reference scores obtained by unsupervised models. However, they do not rely on the same automated metrics and the authors note that “Recent work on ST evaluation highlights the lack of standard evaluation practices (Yamshchikov et al., 2020; Pang, 2019; Pang and Gimpel, 2019; Mir et al., 2019).”.

One of the main difficulties encountered in evaluating the CycleGN models in this dataset is the difficulty in reproducing the results. Indeed, a considerable part of the metrics is based on the use of pre-trained models that are not made available by the authors. While some works make the scripts and training methods for these models publicly available, making it possible, with some effort, to reproduce the results, others provide neither models nor scripts, or even rely on closed, pay-per-use datasets. Even in cases where models and code are accessible, technical hurdles persist. For instance, many style transfer studies rely on outdated software environments, such as Python 2.7, which has reached end of life. Additionally, many dependencies are based on legacy software and libraries that are no longer available in the repositories of modern Linux distributions. Although it is still possible to compile these components manually, doing so is time consuming and presents an unnecessary barrier to research progress.

Only the metrics that could be calculated have been included in the results. The results provided in both [GYAFC](#) and XFORMAL focus on the Informal to Formal direction. Small CycleGN models, as defined in Section 4.4.6, were trained in the English, French, and Italian languages.

Excluding human evaluations, [GYAFC](#) relies on (i) a pre-trained classifier [266] that is fine-tuned on in-domain data to assess the formality of a sentence, (ii) a pre-trained statistical model [131] to evaluate the fluency, (iii) a pre-trained [Convolutional Neural Network](#) [127] to determine the meaning preservation and (iv) [BLEU](#), [TERp](#) [330] and [PINC](#) [50] for the overall ranking.

As no reference implementation for the [PINC](#) score was found, we contacted the authors of the [PINC](#) score but received no response. As a result, the code snippet below was

used to calculate the score. We tried to find a dataset with a known [PINC](#) score to validate our implementation but could not find one.

```
import nltk
from nltk.util import ngrams

nltk.download('punkt')

def get_ngrams(sentence, n):
    tokens = nltk.word_tokenize(sentence)
    return list(ngrams(tokens, n))

def pinc_score(hypothesis, reference, max_n=4):
    pinc = 0.0
    for n in range(1, max_n + 1):
        hyp_ngrams = get_ngrams(hypothesis, n)
        ref_ngrams = get_ngrams(reference, n)

        hyp_ngrams_set = set(hyp_ngrams)
        ref_ngrams_set = set(ref_ngrams)

        unique_ngrams = hyp_ngrams_set - ref_ngrams_set
        unique_count = len(unique_ngrams)

        if len(hyp_ngrams) > 0:
            pinc += unique_count / len(hyp_ngrams)

    pinc /= max_n
    return pinc
```

CODE LISTING 5.1: Python implementation of the PINC score

The baseline [Neural Machine Translation](#) model consists of a [Long Short-Term Memory](#) with an attention mechanism, implemented using the OpenNMT-py [176] framework, with a vocabulary size of 50,000 and embeddings of size 300. The latter are pre-trained on the Yahoo! Answers L6 corpus using GloVE [269], a method similar to Word2Vec, discussed in Section 2.2.3. The [Neural Machine Translation](#) Copy is a copy-enriched model that includes additional dictionaries [156]. The NMT Combined model was not included in the comparison as it was trained using additional source data.

Table 5.1 shows the results obtained by CycleGN and by the unsupervised [Neural Machine Translation](#) models from the original work. The [BLEU](#) and [PINC](#) scores for CycleGN are lower and the TERp score is higher (a lower score is better), which is expected since CycleGN tends to confuse words, as explained in Section 4.4.7. The formality metric shows that the transfer from formal-to-informal is better in the case

of CycleGN compared to the other two models, demonstrating capabilities in text style transfer. This tendency to confuse words also decreases the meaning preservation score.

TABLE 5.1: Scores obtained on the [GYAFC](#) dataset in the Formal to Informal direction depending on the model

Model	Formality (pre- trained mBERT)	Meaning Preserva- tion (pre- trained CNN)	Overall (BLEU)	Overall (PINC)	Overall (TERp)
CycleGN	-0.56	3.47	43.95	36.11	0.67
NMT Baseline	0.07	3.89	56.61	56.92	0.38
NMT Copy	0.10	3.87	58.01	56.39	0.38

The self- [BLEU](#) and multi- [BLEU](#) were obtained using the official scripts available on GitHub for the XFORMAL dataset¹. Self- [BLEU](#) measures the diversity of generated outputs by computing [BLEU](#) scores between each generated sentence and all other generated sentences. In contrast, multi- [BLEU](#) compares generated sentences to multiple human-written reference texts, assessing content preservation rather than diversity. A higher multi- [BLEU](#) score indicates that the generated output remains closer in meaning to human references, ensuring that style transfer does not alter the original intent. The formality score was calculated using the same pre-trained mBERT used in the original [GYAFC](#) work. Tables 5.2 and 5.3, respectively, show the results obtained by CycleGN on the French and Italian datasets. The results obtained by CycleGN are compared with those obtained by UNMT [340] and DLSSM [128], two unsupervised text style transfer models.

Similarly to [XLM](#), mentioned in Section 4.6, UNMT is based on a denoising auto-encoding approach and adds a pooling mechanism in the latent space where larger pooling windows encourage more significant style transformations and smaller ones help retain original content. DLSSM employs variational inference to approximate intractable probabilities, which are complex or high-dimensional probability distributions. For both the overall score and the formality metric, CycleGN performs better than the other models.

Concerning the meaning preservation score, while the score obtained by CycleGN on the French language demonstrates excellent preservation, this behaviour was not observed for Italian. Although initially it was assumed that it was due to training stopping during an instability, as observed in Section 4.4, this relatively low self-BLEU score was observed. at multiple times during training.

The difference in meaning preservation scores between French and Italian could be based on several linguistic factors. Italian’s more nuanced verb conjugations, flexible syntax, and higher reliance on contextual cues may introduce greater semantic ambiguity, making it harder for models to retain meaning during style transfer.

Reproducing results from previous research in text style transfer presents significant challenges, highlighting a gap in accessibility and transparency within the field. The

¹<https://github.com/Elbria/xformal-FoST-meta>

TABLE 5.2: Scores obtained on the XFORMAL French dataset in the Formal to Informal direction depending on the model

Model	Meaning Preservation (self-BLEU)	Formality (pre-trained mBERT)	Overall (multi-BLEU)
CycleGN	73.3	0.18	87.6
UNMT	19.7	-1.91	17.0
DSLM	21.3	-1.90	17.4

TABLE 5.3: Scores obtained on the XFORMAL Italian dataset in the Formal to Informal direction depending on the model

Model	Meaning Preservation (self-BLEU)	Formality (pre-trained mBERT)	Overall (multi-BLEU)
CycleGN (epoch 100)	14.8	0.05	79.3
CycleGN (epoch 200)	15.4	0.16	83.6
CycleGN (epoch 300)	15.2	0.20	81.8
UNMT	15.1	-1.84	17.0
DSLM	19.6	-1.61	17.4

reproducibility of results is crucial in scientific research, as it enables direct comparison and validation of methodologies. However, the current state of reproducibility in text style transfer research remains inadequate. Many studies rely on models and datasets that are not publicly available. Authors often do not respond to requests for model downloads, and the datasets used in training are frequently closed source, with some requiring payment for access. This lack of open resources obstructs the replication of published findings and prevents researchers from benchmarking against existing results.

Moving forward, the field would benefit from greater emphasis on open-source resources and standardised environments, allowing future researchers to build upon existing work with greater ease and reliability. Addressing these issues will improve the integrity and progress of research in this rapidly evolving area.

In the original [GYAFC](#) work, the meaning preservation is calculated using a pre-trained [CNN](#) [127]. Although the model is not publicly downloadable, the authors of the model have given access upon request. Since this work was published nine years ago, it relies on obsolete software such as Python2.7, which is no longer available in the repositories of recent Ubuntu versions, and Torch, which has not been developed for seven years. This problem of using software and libraries that are no longer in official repositories is also found with TERp, the last version of which was released nine years ago.

The model used to assess the fluency of the formal sentences generated in the original [GYAFC](#) [131] is trained on GigaWord 5 [262], which is a paid dataset costing \$6,000 for non-members of the Linguistic Data Consortium of the University of Pennsylvania. We contacted the authors of the pre-trained model to obtain a copy, but received no reply, making it impossible to analyse CycleGN’s fluency for comparison with the original [GYAFC](#) dataset. The inability to obtain this score also makes it impossible to obtain the “Combined” score, which is a normalisation of fluency, meaning preservation and formality scores.

Finally, it was not possible to obtain the fluency score of the models trained on the XFORMAL datasets, since it requires training models on datasets that are no longer accessible and we did not obtain a response when he contacted the original authors to obtain the trained models.

Of the seven automatic text style transfer evaluation metrics used in the original GYAFC work, only five could be reproduced here, and of the four from XFORMAL, three could be calculated.

5.3 Sentiment

Style transfer refers to the transformation of text while preserving its fundamental content but altering stylistic elements. Sentiment polarity shift, where the emotional tone of a sentence is modified, is a style transfer example as the core content (factual information, entities, and structure) remains unchanged and only the stylistic aspect—in this case, sentiment polarity—is altered.

As mentioned in Section 2.7.3, the Yelp dataset [413] consists of restaurant reviews, along with notes from 1 to 5, where reviews with notes equal to 2 or less are negative and reviews with notes equal to or greater than 4 are positive. As this dataset is non-parallel, CycleGN has been trained and compared with other models on this set.

Previous works [198] [275] [102] [322] [399] [410] [222] have explored text style transfer on this non-parallel dataset and rely on (i) a binary classifier tasked with distinguishing positive from negative reviews to assess style transfer and on (ii) self-BLEU for content preservation.

However, as different work rely on different sentiment classifiers, and some were not publicly released, a separate RoBERTa-based model [308] was used to assess the style transfer of CycleGN. This model was fine-tuned specifically on the Yelp dataset and achieves 98.08% accuracy on the Yelp test set. As noted in the original work, the classifier score is defined as “the fraction of outputs classified as having the target attribute”.

Table 5.4 shows that CycleGN achieves the best meaning preservation, meaning it retains the original content more effectively. However, this comes at the cost of lower transfer strength, as only 38.48% of the sentences generated by CycleGN are classified as having the target sentiment. Despite this, the transfer is still meaningful, as random test set reviews score below 2% when evaluated with the same classifier.

The following list briefly describes the models in Table 5.4:

- CrossAligned uses adversarial training with a shared latent space assumption to disentangle style and content, ensuring that content representations are aligned across different styles;
- StyleEmbedding incorporates an additional embedded representation of the target style into a GRU to generate a sentence in the desired style;

- MultiDecoder relies on multiple [GRU](#), each dedicated to a specific style, allowing for explicit style control in text generation;
- TemplateBased is a method that replaces style markers in a sentence with words retrieved from a corpus;
- RetrieveOnly is a simple non-neural method that retrieves the closest sentence from a corpus with the target style and outputs it as the transformed sentence;
- DeleteOnly is an encoder-decoder model that removes identified style markers from the input sentence before generating a new sentence with a different style;
- DeleteAndRetrieve extends DeleteOnly by first deleting style markers and then inserting new ones retrieved from a corpus before generating the final output;
- SGM (Style-Guided Model) integrates explicit style information into the generation pipeline to enhance both fluency and content preservation;
- DualRL (Dual Reinforcement Learning) is a reinforcement learning-based approach that uses dual policy updates to improve content preservation and style strength simultaneously.
- TextStyleTransfer is a [AutoEncoder](#)-based model trained on content-style disentanglement.

TextCycleGAN [218], mentioned in Section 2.7.3, is, similarly to CycleGN, a CycleGAN-derivative. However, it relies on a pair of [LSTM](#) models and requires discriminators. Compared to CycleGN, it achieves a higher transfer strength and a lower meaning preservation score.

TABLE 5.4: Scores obtained on the Yelp dataset depending on the model. The results are averaged on both directions

Model	Meaning Preservation (self-BLEU)	Sentiment (pre-trained classifier)
CycleGN	67.0	38.48%
CrossAligned	3.1	73.7%
StyleEmbedding	11.8	8.7%
MultiDecoder	7.1	47.6%
TemplateBased	11.8	81.7%
RetrieveOnly	0.4	95.4%
DeleteOnly	7.5	85.7%
DeleteAndRetrieve	8.4	88.7%
SGM	44.5	95.4%
DualRL	55.2	85.6%
TextStyleTransfer	32.42	68.04%

CycleGN scores very well in terms of content preservation compared to other models, at the expense of a less pronounced transfer style. To understand these excellent self-BLEU scores, the examples of transfer style are illuminating.

The first list shows examples from the Yelp dataset test set, while the second shows the hypotheses generated by CycleGN. The sentences change very little, but the model

demonstrates, for example, that it can switch between “reliable” and “unreliable”, “pleasant” and “unpleasant”, and “best” and “worst”. The model also adds negations, such as “I highly recommend”, which becomes “I would not recommend”, or “They do clean”, which becomes “They do not clean”.

- Original: “These folks are consistently reliable and polite. They do clean work, and are careful with your belongings and house. I’ve used them for three repairs and one washer installation. I strongly recommend them.”
- CycleGN: “These folks are consistently unreliable and lazy. They do not clean work, and are not careful with your belongings or house. I’ve used them for three repairs and one washer installation. I strongly do not recommend them.”
- Original: “Omg! This is the worst meal I’ve had in a long time. The food is so incredibly dry and the servers are so unpleasant. I would not recommend the pastrami Reuben special or the cinnamon buns!”
- CycleGN: “Omg! This is the best meal I’ve had in a long time. The food is so incredibly fresh and the servers are so pleasant. I highly recommend the pastrami Reuben special and the cinnamon buns!”

Similarly to the positive to negative transfer, CycleGN focusses mainly on converting words, such as “Terrible” to “Excellent”, “horrible” to “amazing” and negations, such as “I will not” to “I will”. This insistence on the meaning of preservation prevents CycleGN from modifying a sentence such as “Employees walking right in front of me without asking how I was doing or even taking the time to say excuse me”, classified as negative by the classifier.

- Original: “Terrible customer service. Cashier texting on his phone and not friendly. Employees walking right in front of me without asking how I was doing or even taking the time to say excuse me. Just all around rude. I will not be returning”
- CycleGN: “Excellent customer service. Cashier texting on his phone and great. Employees walking right in front of me without asking how I was doing or even taking the time to say excuse me. Just all around friendly. I will be returning”
- Original: “We had two combination plates. The sushi and tempura were good. The yakatori and teriyaki were amazing.”
- CycleGN: “We had two combination plates. The sushi and tempura were good. The yakatori and teriyaki were horrible.”

5.4 Reflections and Future Directions in text style transfer

The sentences generated by CycleGN in the context of text style transfer tend to be particularly literal, achieving high BLEU scores at the expense of more effective style transfer.

Although CycleGN originates from techniques used in style transfer, notably in the domain of image processing with CycleGAN, as discussed in Section 2.6.4, the current implementation of this architecture does not appear to be the most effective for text style transfer.

In addition to the reproducibility challenges in previous work, the lack of standardised evaluation metrics hinders the development of text style transfer techniques. Unlike MT, which has well-established evaluation criteria such as fluency, adequacy, and overall translation quality, style transfer currently requires at least two: preservation and style transfer effectiveness.

We are currently drafting the initial version of a forthcoming publication on CycleGN for text style transfer. Additional models are being trained on different types of style transfer tasks using diverse datasets to gather new data and achieve more conclusive results regarding CycleGN’s performance in text style transfer. We aim to submit this work to a conference in the near future.

5.5 Limitations

By improving performance on unsupervised text style transfer, we hope that performance on unsupervised Machine Translation will also be improved, and vice versa.

5.6 Conclusion

CycleGN, originally designed as an unsupervised Machine Translation system, has proven to be a versatile framework capable of tackling broader text transformation tasks. Using its style-transfer-orientated architecture, we demonstrated that CycleGN can also be effectively applied to text style transfer, addressing both formality and sentiment-based tasks without requiring parallel corpora. The experiments presented in this chapter highlight the strengths of CycleGN in content preservation, which makes it particularly adept at maintaining semantic integrity while performing stylistic transformations. However, this fidelity to the source text occasionally comes at the expense of transfer intensity.

Despite these trade-offs, the results establish CycleGN as a promising baseline for unsupervised text style transfer, with competitive performance compared to established models across several languages and datasets. Moreover, the simplicity of CycleGN, which does not require discriminators or complex disentanglement modules, makes it an accessible alternative for scenarios where meaning preservation is paramount.

However, this work also underlines key challenges in the field, including fragmentation of evaluation standards and the frequent difficulty in reproducing published results. These systemic issues slow down progress in text style transfer research and highlight the need for more open, standardised benchmarks and reproducible experimental setups.

Ultimately, CycleGN bridges the gap between translation and style transfer, reaffirming the interconnected nature of these tasks within the broader landscape of natural language generation. With further refinement, we believe that CycleGN has the potential to become a valuable tool not only for research, but also for practical applications in content moderation, personalised communication, and adaptive dialogue systems.

How can style transfer methods originally designed for images be adapted for text style transfer tasks?

Style transfer methods from the image domain, such as CycleGAN, can be adapted to text by applying the concept of cycle consistency to ensure that stylistic transformations preserve the original meaning. This approach enables unsupervised text style transfer without the need for parallel datasets. However, this adaptation often results in high meaning preservation at the expense of weaker style transfer, limiting the intensity of the stylistic shift.

Chapter 6

Neural Text Compression

Perfection is achieved, not when
there is nothing more to add,
but when there is nothing left to
take away

Antoine de Saint-Exupéry

In Chapter 4, we present CycleGN, a novel unsupervised [Neural Machine Translation](#) framework that is based on a cyclic loss. In Chapter 5, we apply this framework to text style transfer tasks and demonstrate that CycleGN can be trained on such tasks and successfully transfer a style without altering the content, albeit at the cost of producing slightly overly literal translations. In this chapter, we focus on lossless text compression using [Large Language Models](#) and the effects of quantisation on the compression ratio. Our objective is to answer the Research Question 4. How can [Large Language Models](#) be adapted for both efficient text compression and dataset membership inference through quantisation analysis?

6.1 Introduction to Compression

The aim of data compression is to transform a sequence of bits into another, where the shorter sequence can reproduce the same or similar information. There are two types of data compression. The first is lossless compression, where the compressed data can be restored to their exact pre-compressed state, since the decompression operation is fully reversible. The second type is lossy compression, which converts data to data that are close but not identical, where some of the initial information is lost.

The compression ratio is a measure of the efficiency of a compression algorithm. It expresses the ratio between the size of the data before compression and the size of the data after compression. The compression ratio formula is as follows:

$$\text{Compression\_ratio} = \frac{\text{Size\_of\_the\_original\_data}}{\text{Size\_of\_the\_compressed\_data}}$$

The following sections do not aim to provide an exhaustive account of compression techniques but rather offer a concise overview to facilitate the understanding of later sections.

6.1.1 Lossy Compression

Lossy compression functions are non-reversible functions, since the original data cannot be reconstructed identically from the compressed data. Nonetheless, they enable the size of a series of bits to be reduced considerably while retaining data close to the source data.

This type of compression is frequently used for types of data where the accuracy of human senses is a limiting factor.

Sound

Information about the target's hearing ability is used to adjust the compression level of an algorithm. Since the human ear can only perceive sounds in a frequency range from 20 Hertz to 20 Kilohertz, when compressing music intended for human hearing, it is possible to discard all frequencies outside this data range, in order to reduce the size of the series of bits. This lost information cannot be reconstructed at a later date, but this simple algorithm makes it possible to reduce the amount of data without compromising the listener's experience.

In order to capture the entirety of an audio signal and convert it into a digital format, the sampling rate must be at least twice the maximum frequency [248], a theorem known as the Nyquist-Shannon Sampling Theorem. This is why 44,100 hertz is a commonly used sampling frequency in digital audio [379]. The size of the file can be diminished by reducing the sampling frequency. Rather than recording the frequency of a sound 44,100 times per second, it is possible to record it only 22,050 times per second to divide the quantity of information by up to 2. However, higher frequencies will be lost during operation, which will degrade the quality of the audio. Modern compression algorithms such as OPUS rely on a variable sampling rate to adjust the frequency in real time based on needs [255].

The sampling resolution is also another massive factor in lossy compression. While the sampling frequency determines the number of samples extracted from an audio source in one second, the resolution is the number of unique values that can be attributed to a sample. In audio file with a depth of 32-bits, a sample can take 2^{32} possible values. Reducing the bit depth of an audio file by a single bit can theoretically cut the file size nearly in half, although actual savings depend on the file format and other factors. More sophisticated compression techniques, such as those used in OPUS or MP3, involve, for example, automatically and continuously adjusting the depth according to the perceptual importance of the audio signal.

Depending on the level of compression, lossy compression may not affect a user's hearing comfort at all, or it may render a sound completely inaudible, with all the possible nuances in between, the compromise often being to reduce the quality while it is still barely noticeable to the listener, in order to save on storage and bandwidth.

Image

The sampling frequency of an audio signal is analogous to the definition of an image, i.e. the total number of pixels in the image. In the same way, audio depth is almost equivalent to image depth, except that depending on the image format, it is possible to have several channels.

As mentioned in Section 2.1.1, digital image in greyscale is nothing more than an image with a single channel, which determines the brightness for each pixel. A colour image can be recorded in several ways depending on the colour space used, the best known being RGB and requires 3 channels, one for Red, one for Green, and one for Blue. In a 32-bit RGB image, each pixel records three 8-bit values (one for each colour), with 256 possible values per channel, resulting in 24 bits for colour and typically 8 bits for transparency. Similarly to lossy audio compression, reducing the bit depth of an image can reduce file size, though this also decreases color precision and can degrade image quality.

6.1.2 Lossless Compression

Lossless compression is, by definition, a completely reversible operation. One of its main use case is in archival and backup, since the objective is to be able to recreate an exact copy of the data at a specific time while saving on storage costs compared to a non-compressed archive. Lossless compression is also one of the cornerstone of web performances. Indeed, more than 88% of websites rely on lossless text compression to reduce the required bandwidth and accelerate loading times [364].

Lossless compression is a trade-off between savings on data storage and bandwidth, since compressed data requires less space, and the computing time needed to recover the data, since instead of merely accessing the data it is first necessary to decompress it. However, modern hardware and software algorithms are highly efficient, and the decompression time is often negligible compared to the benefits.

Sound

Patterns or repetitions in the waveform can be compressed using techniques such as entropy coding, further discussed in Section 6.2.1, where frequently occurring elements are represented with shorter codes. In addition, predictive coding can be used to encode only differences between samples, rather than full data. Popular lossless audio formats such as FLAC employ these methods to provide high-fidelity sound with reduced storage requirements.

Image

[Run-Length Encoding](#) is an algorithm particularly adapted to data with frequent successive occurrences of the same elements. [RLE](#) replaces consecutive identical sequences as a single sequence and the number of occurrences. Thus, the chain “AAAABBBCC” is compressed as “4A3B2C”. The BMP file format can optionally rely on [RLE](#) to increase the compression ratio of an image.

6.2 Lossless text compression

Lossless text compression refers to the process of reducing the size of a text file without loss of original information, ensuring that the data can be perfectly reconstructed from its compressed form. This type of compression is essential in contexts where data integrity is crucial, such as legal documents, software code, and archive storage.

Many lossless compression algorithms exist, which use a wide array of techniques. As with other compression algorithms, there is usually a trade-off between the compression and decompression speed and the compression ratio, and slower algorithms usually are able to decrease the final size more than faster algorithms.

6.2.1 Algorithms

Some lossless text compression algorithms rely on the use of dictionaries to replace whole words with shorter indexes, and others try to predict the next words according to the previous ones. In addition, some methods focus on identifying long and frequently recurring strings and replacing them with shorter codes. Different families of algorithms can compress strings without loss and are frequently combined to obtain superior results.

Repeated sequences

In a character string, the probability that a continuous sequence of characters will be repeated increases with the size of the string. This is particularly likely if the character string represents natural languages, where repetitions of words and roots are frequent.

[LZ77](#) [436] keeps a fixed-sized sliding window of up to 32 KiB, the history buffer, to store previously seen text. It analyses the look-ahead buffer, and if a sequence in this buffer is a repetition of a sequence in the history buffer, it is replaced by a pointer to the position of this repetition and the length of the sequence. Thus, the character string “ABCABCD” could be compressed as “ABC03D”, “0” being the position of the first occurrence of the string “ABC”, which is of length 3.

[Lempel–Ziv–Markov chain Algorithm](#) [267] is an improvement of [LZ77](#) which features a larger sliding window along with models based on Markov chains to optimise the compression ratio, since it relies on continuous characters to predict upcoming symbols,

based on previous sequences. It also relies on range coding to also compress the ranges and lengths of the pointers, further optimising the compression ratio.

Entropy Coding

Unlike algorithms based on the detection of repeated symbols such as [LZ77](#) and [LZMA](#), Entropy Coding algorithms target the frequency of symbols and sequences of symbols. Huffman coding [148] creates a binary tree in which the most frequent symbols are given the smallest number of bits and the less frequent symbols are given longer codes.

Entropy coding can be used in addition with other algorithms. For example, DEFLATE [79] is a compression file format which first compresses text using the [LZ77](#) algorithm and then applies the Huffman coding to the compressed text, compressing it even further. Bzip2 [319] relies on [RLE](#), mentioned in Section 6.1.2 and Huffman coding, in addition to other algorithms that will not be discussed in the interest of brevity.

Next-Token Prediction

In a similar manner to the way that Markov chains can be used in lossless text compression, since they compress more efficiently by predicting the next characters, recent advances in [Machine Learning](#) have also greatly improved the predictive capabilities of the models.

As discussed in Section 2.4.2, [Next Token Prediction](#) is the task of predicting the next token based on all or some of the previous tokens. Although [NTP](#) is usually employed as a form of pre-training in many [Natural Language Processing](#) models such as [GPT](#), it can also be used directly for text compression. nncp [30] is a collection of [LSTM](#) and Transformer models that have been trained on [NTP](#) to perform lossless text compression. nncp is the current State-of-the-Art on the Large Text Compression Benchmark, as mentioned in Section 6.2.2.

6.2.2 General Benchmarks

The efficiency, as measured by the compression ratio, of the different string compression algorithms varies considerably depending on the nature of the data to be compressed [120]. While some algorithms are better suited to images, others focus on natural languages. Furthermore, the compression ratio should generally be compared with the time required for compression and decompression, which depends on the algorithm's computational complexity.

Many algorithms have different options for adjusting the balance between compression and decompression speed and the final size of the compressed data. For example, Zstandard¹ offers a *-fast* option where the higher the number, the faster the compression

¹<https://github.com/facebook/zstd>

and decompression at the cost of a smaller compression ratio. However, the focus of this research is on compression efficiency, and the speed of compression and decompression is beyond the scope of the analysis presented here.

Wikipedia

The Large Text Compression Benchmark [224] is a competition that has been running since 2006, with the goal of obtaining the best compression ratio on the first 10^9 bytes of the English Wikipedia dump on March 3, 2006, also known as enwik9. This dataset contains mainly English text; therefore, the results of this benchmark are not fully representative of the compression ratios for text in other languages.

It is important to note that the submissions are ranked according to the total size, that is, the size of the program required to decompress the data, including any dictionaries, is added to the size of the compressed enwik9.

Table 6.1 references the most common text compression algorithm and the corresponding compressed enwik9 sizes. Although nncp [30] is not a well-known algorithm, it is included in the table as the current winner of the competition.

TABLE 6.1: Size of the compressed enwik9 dataset depending on the compression algorithm

Algorithm	Size of compressed enwik9 (in bytes)	Compression ratio
ncpp v3.2	106,632,363	9.38
paq8h	132,045,026	7.57
zpaq	142,252,605	7.03
bsc	160,258,936	6.24
7zip	178,965,454	5.59
xz	197,331,816	5.07
WinRAR	198,454,545	5.04
zstd	215,674,670	4.64
brothli	223,597,884	4.47
bzip2	253,977,839	3.94
gzip	322,591,995	3.10
lz4	379,999,522	2.63

Silesia Corpus

The Silesia Corpus [77] is a collection of 12 documents, each between 6 MiB and 51 MiB, for a total size of 211,938,580 bytes. These documents include English texts, images, source codes, and executable files.

The official Zstandard documentation² includes the compression ratios obtained by different algorithms, which are referenced in Table 6.2. Since compressed sizes are not available, they are estimated from the compression ratios provided.

²<https://github.com/facebook/zstd>

TABLE 6.2: Size of the compressed Silesia Corpus depending on the compression algorithm

Algorithm	Size of compressed Silesia corpus (in bytes)	Compression ratio
zstd	73,411,354	2.89
brotli	78,437,668	2.70
lz4	100,875,097	2.10

Alice29

The Canterbury corpus [16] is, similarly to the Silesia corpus, a collection of files of different types, used to benchmark lossless compression algorithms. The size of the corpus is 2,810,784 bytes and includes the file “alice29.txt”, which is a text version of the novel well-known “Alice’s Adventures in Wonderland” from Lewis Carroll and contains 148,479 bytes. The Squash Compression Benchmark [243] maintains a leaderboard for compression algorithms on a collection of files, including alice29.txt.

Table 6.3 lists the compressed sizes of the alice29.txt file, estimated from the compression ratio provided, for the most common lossless compression algorithms, using the highest compression level available for each algorithm.

TABLE 6.3: Size of the compressed alice29.txt file depending on the compression algorithm

Algorithm	Size of compressed alice29 file (in bytes)	Compression ratio
bzip2	42,181	3.52
brotli	45,968	3.23
xz	47,286	3.14
gzip	52,839	2.81
zstd	58,687	2.53
lz4	62,125	2.39

We examined data compression in general, both lossy and lossless, and frequently used datasets used to compare compression algorithms, particularly lossless text compression algorithms. We now present Llamazip, a lossless text compression algorithm based on the use of a pre-trained LLM.

6.3 Llamazip

As mentioned in Section 3.3.7, LLaMA3 is a recently released Open Source LLM that relies on NTP for pre-training. Llamazip is an LLaMA3-built lossless text compression algorithm dedicated to data archival and storage, as it trades compression and decompression speed for maximal reduction in the size of the data. Using the predictive capabilities of LLaMA3, Llamazip only stores mistaken forecasts to minimise storage requirements.

Although Llamazip is not directly related to the core focus of this thesis, which primarily revolves around neural machine translation, it emerges from the use of LLMs, which were essential for developing Embed_llama, described in Chapter 3. Additionally, it stems

from a deeper exploration of unsupervised training, a crucial component of CycleGN, discussed in Chapter 4.

More than just a compression algorithm, Llamazip might also be considered a tool to detect whether a document was part of the training set of the underlying model, in this case LLaMA3. By comparing its compression performances, defined by its compression ratio, using its highest and lowest performing hyperparameters, a pattern appears to emerge between known documents and new text. Finally, Llamazip can also be considered as a benchmark for models trained on NTP tasks, which acts as a side-effect as a lossless text compression algorithm.

The work presented in this section also examines the effects of LLM quantisation and context window size on the performance of NTP task. Quantisation, as introduced in Section 2.8, is a technique that reduces the size of the parameters and can potentially speed up both inference and training by simplifying calculations, although it can result in a slight decrease in the precision of the parameters. The context window refers to the number of tokens considered when predicting the next token; a larger window requires more memory and longer computation times. Analysing the impact of these two variables reveals performance variations in Llamazip, thereby reflecting the performance of the underlying model.

6.3.1 Framework

Similarly to the published work conducted on Machine Translation Evaluation and on Unsupervised Machine Translation, the HuggingFace [391] library was used to predict tokens using “meta-llama/Meta-Llama-3-8B”³ and to generate documents using “meta-llama/Meta-Llama-3-8B-Instruct” model and the “mistralai/Mistral-7B-Instruct-v0.3” model⁴.

6.3.2 Parameters

The Llamazip algorithm has several parameters that affect the efficiency of the algorithm, measured by the compression ratio. The exact parameters used to compress a file must be known at the decompression time to retrieve the original data.

Model

LLaMA3 is available in two different sizes: both an 8 billion version and a 70 billion parameter version. The choice of the model is one of the main factors in memory requirement. As the 70 billion LLaMA3 parameters cannot run on consumer-grade hardware, it falls outside the scope of this work. While other LLMs trained on NTP tasks could have been considered, such as Mistral [158] and Qwen [21], their use falls outside the scope of this research, which focusses specifically on LLaMA3.

³https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

⁴<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3>

Quantisation

As mentioned in Section 2.8, recent advances in quantisation allow for a reduction in the memory requirement of large models, at the cost of a decrease in the compression ratio. Different quantisations are available: float16, bfloat16, int8 and int4, each representing different levels of precision. Float32 (no quantisation) was not tested, as it cannot fit on consumer-grade hardware.

Initial Window Size

This parameter determines the minimum number of tokens required to predict the next token. This parameter is equal to 10 for this entire work, a number chosen arbitrarily by the author, since its impact is limited for any text that is at least longer than the maximum window size.

Maximum Window Size

Although the compression ratio would probably be maximum using all previous tokens to predict the next token, the size of the available memory restricts the number of tokens that can be exploited for NTP. Although the window size can take any value in $\mathbb{N}^*$, it was restricted to the set [16, 32, 64, 128, 256, 512, 1024, 2048] in the context of the current work, since each additional token increases the memory requirements of the algorithm, as discussed in Section 6.3.8.

Final Compression Algorithm

This option determines the format in which the file will be stored on the disk. Llamazip offers the option of performing a final compression with a classic algorithm such as gzip or xz, to reduce the file size considerably one last time. As Llamazip is particularly computationally intensive, the additional cost of this final compression step is negligible in comparison.

6.3.3 Algorithm

In this section, we present a detailed explanation of Llamazip, a lossless text compression algorithm. The algorithm is designed to efficiently reduce the size of a text file while ensuring that the original data can be fully recovered without any loss of information. The compression transforms text into a denser representation, which consists of token identifiers and integers representing the number of consecutive correct predictions, while decompression converts these compact data back into the original text.

The quantisation type, the size of the context window, the initial size of the context window, and the final compression algorithm are directly encoded in the name. “file_name”

becomes “file_name_quantisation_context_window_10.txt.compression_algorithm” where “quantisation” is one of [4, 8, bf16, f16], “context_window” is one of [16, 32, 64, 128, 256, 512, 1024, 2048] and “compression_algorithm” is one of [gz, br, xz, zst, zpaq].

Compression

Compression relies mainly on [LLaMA3](#)’s predictive capacity, since once a token is correctly predicted, there is no need to record it, and therefore there is only a need to record errors.

1. The counter is initialised with the value 0.
2. The text to compress is tokenized.
3. The first tokens, the number of which is determined by the initial window size, are added to a list and saved directly at the beginning of the output file.
4. The model tries to predict the next token from the tokens stored in the list until the file is fully compressed. There are two possibilities:
 - If the prediction is successful, the value of the counter is incremented, and the predicted token is appended to the list.
 - If the prediction is wrong, the counter is recorded in the output file and then set to 0. The index of the token, preceded by the character “@” to distinguish it from the value of the counter, is recorded in the output file and appended to the list.
5. After each prediction, the size of the list is examined, and once again there are two possibilities:
 - If the size of the list is less than the maximum window size, no further operations are required.
 - If the size of the list is equal to the maximum window size, the oldest value of the list is discarded.
6. Once all the files have been read, the counter value is written for the last time to the end of the file.
7. The file is then further compressed using the final compression algorithm.
8. The parameters used to compress the data are recorded in the file name.

Decompression

Decompression involves following the pattern recorded during compression, i.e. alternating between prediction sequences and recovery sequences for tokens written as they are in the file.

1. The file is first decompressed using the final compression algorithm.
2. The first tokens, the number of which is determined by the initial window size, are added to a list.
3. The next value is read. There are two possibilities:
 - If the first character of the value is “@”, it means that it is an index of a token and not a counter value. The index of the token is appended to the list and to the output file. If the size of the list corresponds to the maximum window size, the oldest token is discarded from the list.
 - Else, the LLaMA3 model predicts n tokens, where n is the value of the counter. After each prediction, the token is added to the list and to the output file, and if required, the oldest token is removed from the beginning of the list.
4. The output file is then detokenized in order to retrieve the characters.

Binary Format

If the file compressed by Llamazip does not undergo a final compression algorithm, the optimised storage method writes the file bit by bit. Since LLaMA3 has a vocabulary size of 128,256 tokens, the nearest higher power of 2 is 2^{17} , and therefore 131,072. Llamazip then assumes that the counter value will never be greater than $131,072 - 128,256 = 2816$.

Llamazip therefore records the token indices as they are, and the counters are recorded in the range [128,256, 131,071]. The final file will have a size in bits which will be a multiple of 17, since each token or counter value is recorded over 17 bits. However, this file format is less efficient than using a final compression algorithm.

6.3.4 Similar Work

LLMZip [357] is a compression algorithm that relies on LLaMA1, in its 7 billion parameter version, to compress text data. Similarly to Llamazip, LLMZip relies on the NTP capabilities of a pre-trained LLM to avoid recording every token. However, while Llamazip only registers tokens that were accurately predicted and discards inaccurate hypotheses, LLMZip saves the rank of the correct token. In fact, at each step, an LLM can output the probability that each token in the vocabulary is the next token, so it is possible to sort the tokens by probability and record the position of the correct token. In this manner, LLMZip only records integers, where the lowest values mean that the tokens are among the most likely and the highest values mean that the LLM estimated that the token only had a low probability of appearing at this emplacement.

6.3.5 Results

Various text files, all in English, have been compressed using Llamazip. Each file was compressed with 8 different window sizes, as well as 4 different quantisation settings, for a total of 32 combinations per file, and the final compression algorithm is gzip for all the iterations. The sizes and compressions ratios between the optimal Llamazip parameters and different compression algorithms are then compared.

Alice29

As mentioned in Section 6.2.2, `alice29.txt` is a text representation of “Alice’s Adventures in Wonderland”, a 148,479 bytes novel commonly used in text compression benchmarks. Figure 6.1 displays the size of the novel compressed by Llamazip depending on the parameters used. It is immediately apparent that a long context window and quantisation that only halves the initial size of the model are factors that improve the compression ratio. Both `float16` and `bfloat16` display similar efficiencies.

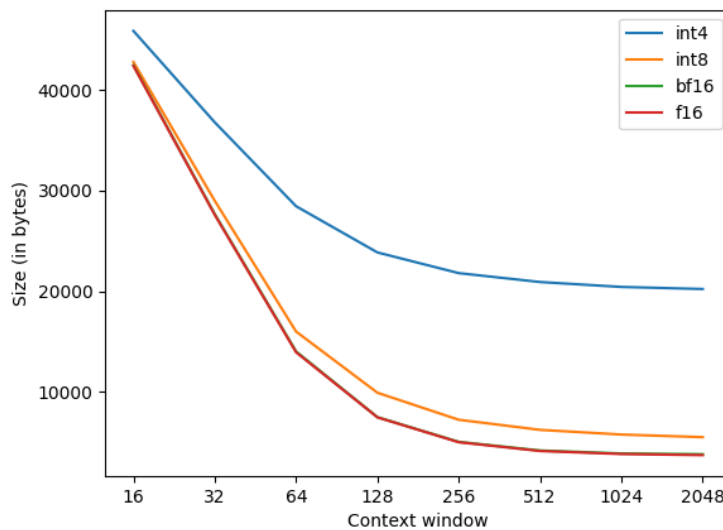


FIGURE 6.1: Compression of `alice29.txt` depending on the context window and the quantisation method. The `bfloat16` and `float16` performances are so similar that the two lines are overlapping

Table 6.4 presents a comparison between the best parameters of Llamazip and other common text compression algorithms. Although the compression ratio of Llamazip is impressive, it is biased by the fact that the novel is very likely part of the training data, as are all books made available by the Gutenberg Project [125], a free online library that mainly collects public domain books, and because this precise file is freely downloadable from many sources.

TABLE 6.4: Size of the compressed alice29.txt file depending on the compression algorithm

Algorithm	Size of the compressed alice29 document (in bytes)	Compression ratio
Llamazip	3,682	40.32
brotli	45,999	3.23
xz	47,872	3.10
gzip	53,429	2.78
zstd	56,245	2.64
zpaq	65,312	2.27
nncp	73,472	2.02

Frankenstein

Frankenstein is an iconic novel from Mary Shelley published in 1818 in which Victor Frankenstein, a young scientist, creates a living being by assembling and reanimating body parts. The file was obtained from Codebreaker Github repository⁵, which was first obtained from Project Gutenberg. The original file contains 441,034 bytes.

As shown in Figure 6.2 and similarly to the alice29.txt file, the length of the context window and the quantisation method have a massive impact on the size of the compressed file.

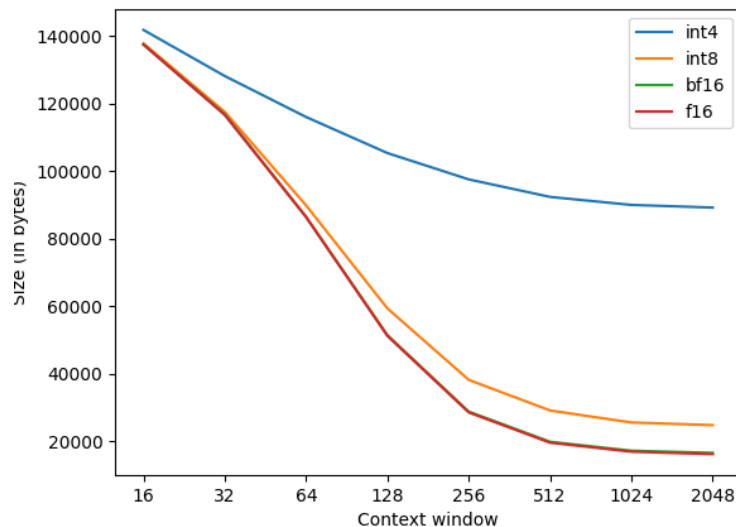


FIGURE 6.2: Compression of the Frankenstein novel depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping

Similarly, Table 6.5 shows how Llamazip outperforms other algorithms on data that are likely part of the LLaMA3 training set.

⁵<https://github.com/asweigart/codebreaker>

TABLE 6.5: Size of the compressed Frankenstein.txt file depending on the compression algorithm

Algorithm	Size of the compressed Frankenstein novel (in bytes)	Compression ratio
Llamazip	16,273	27.10
brotli	137,788	3.20
xz	141,644	3.11
zstd	164,549	2.68
gzip	166,868	2.64
zpaq	188,821	2.34
nncp	194,258	2.27

FSDFS

The excellent compression ratios of Llamazip in previous texts are probably due to the fact that these documents were part of the dataset on which the underlying model, [LLaMA3](#), was trained.

It is crucial to check the effectiveness of Llamazip on a document more recent than the [LLaMA3](#) training date, as this is the only way to ensure that the document is not part of the training set. Archive of Our Own [96], also known as AO3, is a “fan-created, fan-run, nonprofit, noncommercial archive for transformative fanworks, like fanfiction, fanart, fan videos, and podfic”. AO3 provides a search engine with a wide range of criteria.

FSDFS [310] is a novel published on AO3 by user “Scarpoile” after the release of [LLaMA3](#), which has been marked as not containing content marked as “Not Safe For Work” and is in English. Its size of 1,253,946 bytes was shortened to 441,034, the same size as the Frankenstein novel, deemed adequate for fairly fast compression.

Following the same pattern as previously observed, Figure 6.3 demonstrates the impact that the context window and quantisation method have.

On text that should not be part of the training set, Llamazip obtains a better compression ratio than other algorithms, as demonstrated by the results shown in Table 6.6. This shows that Llamazip is not only effective on previously seen data.

TABLE 6.6: Size of the compressed FSDFS.txt file depending on the compression algorithm

Algorithm	Size of the compressed FSDFS novel (in bytes)	Compression ratio
Llamazip	119,197	3.77
brotli	130,780	3.43
xz	134,428	3.34
zstd	157,801	2.84
gzip	160,121	2.80
zpaq	179,020	2.51
nncp	189,197	2.33

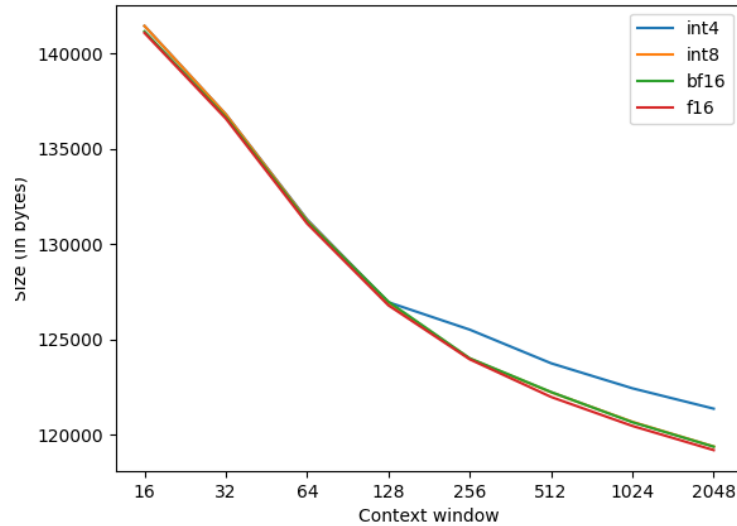


FIGURE 6.3: Compression of the FSDFS novel depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping

Generated by Llama3

Since [LLaMA3](#) is an [LLM](#), it is a generative model that is perfectly capable of being fine-tuned to write a short story. Fortunately, Meta, the creators of [LLaMA3](#), provide the “Meta-Llama-3-8B-Instruct” model, which is fine-tuned for dialogue and answering questions. The document was generated using two prompts, namely the system prompt, which remained constant during the generation, and the user prompt.

- System prompt: “You are a novelist chatbot who always write long novels. You are specialized in original fairy tales. You do not commit the wrongful act of plagiarism and invent your own stories. You write long fairy tales.”;
- Initial user prompt: “Once upon a time,”;
- The following user prompts are the last 50 tokens generated during the last generation.

The novel was generated in 160 steps, using a temperature of 0.5 and the model generated at least 256 tokens during each step. Since the seeds of the [RNGs](#) were set, the generation is fully reproducible. The final file was shortened to 441,064 bytes.

Consistently with previous results, Figure [6.4](#) shows how a longer compression window with less lossy quantisation outperforms other parameters.

In accordance with the results of other documents, Llamazip performs the best. However, as mentioned, the text was generated by a fine-tuned version of the underlying model, and it is not surprising that the generated text has similarities with the training data.

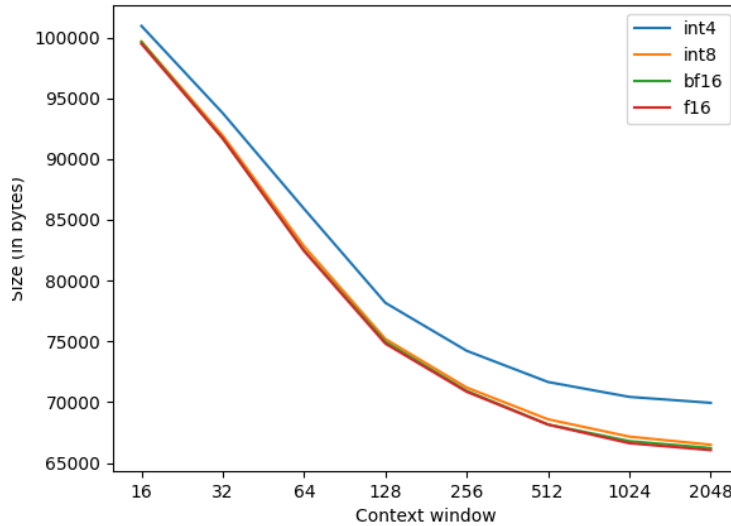


FIGURE 6.4: Compression of the novel generated by LLaMA3 depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping

TABLE 6.7: Size of the compressed LLaMA3 novel file depending on the compression algorithm

Algorithm	Size of the compressed LLaMA3 novel (in bytes)	Compression ratio
Llamazip	66,055	6.68
brothli	88,409	4.99
xz	90,360	4.88
zstd	117,813	3.74
gzip	122,395	3.60
zpaq	127,767	3.45
nncp	174,121	2.53

Generated by Mistral

Llamazip is expected to outperform other lossless compression algorithms on text generated by the underlying model. The next step is therefore to generate a document with another model, in this case Mistral-7B-Instruct-v0.3 [158]. The generation parameters were kept the same, and the file contains 565,146 bytes. The results obtained in this file are similar to those obtained on the document generated by LLaMA3 and are shown in Figure 6.5.

Table 6.8 shows that the compression ratio obtained on this document is very close to the compression ratio obtained on the document generated by LLaMA3. Although Mistral and LLaMA3 are models with different architectures, trained on different datasets by different companies, it is likely that the two training sets intersect, thus explaining this similar compression ratio.

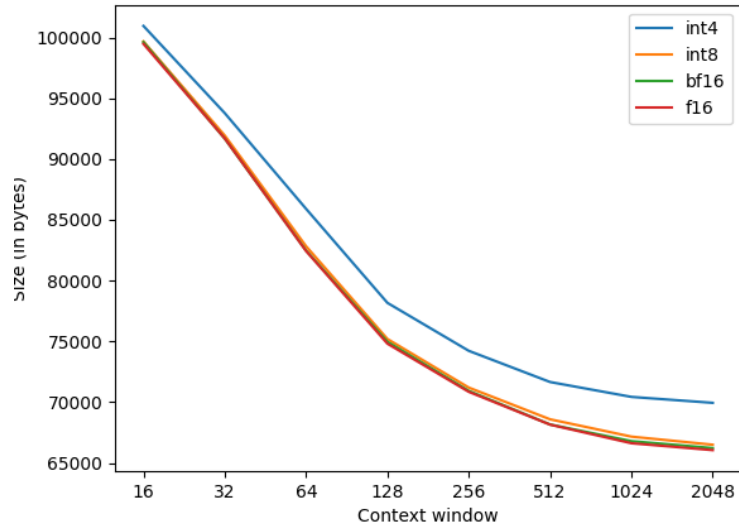


FIGURE 6.5: Compression of the novel generated by Mistral depending on the context window and the quantisation method. The bfloat16 and float16 performances are so similar that the two lines are overlapping

TABLE 6.8: Size of the compressed Mistral novel file depending on the compression algorithm

Algorithm	Size of the compressed Mistral novel (in bytes)	Compression ratio
Llamazip	85,632	6.60
brotli	120,549	4.69
xz	122,596	4.61
zstd	159,709	3.54
gzip	164,880	3.43
zpaq	173,084	3.43
nncp	218,203	2.59

enwik9

As Llamazip is an expensive algorithm and since the size of enwik9 is 1 GiB, as its name suggests, it would unfortunately take too long to compress it in its entirety. Using the [LLaMA3](#) tokenizer, the enwik9 dataset is 288,204,082 tokens long, which is quite substantially more than the 300,972 of the novel FSDFS. Thus, similarly to LLMZip [\[357\]](#), the performance of Llamazip is measured over 10 batches of 100,000 tokens each, which are sampled uniformly on the 13,147,026 lines of the dataset.

Rather than compressing it 32 different times to measure the differences depending on the context window and the quantisation, as was done with the previous files, the 10 batches were compressed using a context window of 2048 tokens and the models were quantised into float16. Table [6.9](#) presents the compression ratios of the different batches of 100,000 tokens.

TABLE 6.9: Size of the different compressed batches of the enwik9 dataset using Llamazip. For all algorithms other than Llamazip presented in this table, the compression ratios were sourced from [224], and the compressed sizes were extrapolated based on these ratios

Index of first token (in lines)	Size in bytes of the batch	Size of the com- pressed batch (in bytes)	Compression ratio
0	374295	72811	5.14
1,314,702	361,735	75,118	4.82
2,629,404	247,737	20,719	11.96
3,944,106	353,510	73,806	4.79
5,258,808	327,783	45,000	7.28
6,573,510	334,072	35,649	9.37
7,888,212	368,903	86,412	4.27
9,202,914	340,115	68,056	5.00
10,517,616	345,954	71,151	4.86
11,832,318	361,407	81,016	4.46
Llamazip Total	3,415,511	629,738	5.42
ncpp v3.2	3,415,511	364,126	9.38
paq8h	3,415,511	451,190	7.57
zpaq	3,415,511	485,847	7.03
bsc	3,415,511	547,357	6.24
7zip	3,415,511	611,003	5.59
xz	3,415,511	673,670	5.07
WinRAR	3,415,511	677,680	5.04
zstd	3,415,511	736,101	4.64
brotli	3,415,511	764,096	4.47
bzip2	3,415,511	866,880	3.94
gzip	3,415,511	1,101,777	3.10
lz4	3,415,511	1,298,673	2.63

6.3.6 Effects of the Final Algorithm

In order to investigate the effect of the final compression algorithm, for each document, the average size over all quantisations and context windows is obtained for the default final algorithm, gzip, and the alternatives, namely brotli, xz, zstd and zpaq.

Table 6.10 demonstrates the large difference in the size obtained depending on the context window and the quantisation used. These two parameters alone can result in a 12-fold difference in the number of bits of the compressed text on a document seen during training.

TABLE 6.10: Variations in the size of the compressed alice29 document depending on the quantisation and context window of Llamazip

Largest size (in bytes)	Smallest size (in bytes)	Average size (in bytes)	Standard de- viation (in bytes)	Ratio between largest and smallest
45,911	3,682	17,406	13,509	12.47

Although gzip is the default final compression algorithm for Llamazip, Table 6.11 demonstrates that brotli and xz perform much better and reduce the size of the compressed file by an additional 5-10%.

TABLE 6.11: Average size of the alice29 file over all context windows and quantisations depending on the final compression algorithm

Final compression algorithm	Average size of the compressed alice29 file (in bytes)	Ratio compared to gzip
brotnli	15,858	0.91
xz	16,075	0.94
zstd	18,594	1.05
zpaq	25,961	1.60

Similarly to the results obtained on the alice29 document, the ratio between the worst quantisation and window size and the best is substantial, as shown in Table 6.12. This similarity between the results of the compression of these two documents is probably explained by the fact that both these novels were part of the training set of LLaMA3.

TABLE 6.12: Variations in the size of the compressed Frankenstein document depending on the quantisation and context window of Llamazip

Largest size (in bytes)	Smallest size (in bytes)	Average size (in bytes)	Standard deviation (in bytes)	Ratio between largest and smallest
141,876	16,273	72,890	44,119	8.72

Brotli and xz also outperform gzip on the Frankenstein novel. Table 6.13 indicates that the difference between xz and brotnli is much smaller in this document compared to alice29.

TABLE 6.13: Average size of the Frankenstein file over all context windows and quantisations depending on the final compression algorithm

Final compression algorithm	Average size of the compressed Frankenstein file (in bytes)	Ratio compared to gzip
brotnli	65,431	0.91
xz	65,500	0.91
zstd	77,085	1.05
zpaq	99,568	1.42

In a document that is not part of the LLaMA3 training dataset, since it was published after the release of the model, the ratio between the worst compression and the best compression is vastly inferior to the other documents. Although a larger window size and less lossy quantisation lead to better compression, Table 6.14 demonstrates that these parameters have a smaller influence on compression compared to previously seen documents. Although this observation is not surprising, it may nevertheless reveal another use for Llamazip which is discussed in Section 6.3.11.

As shown in Table 6.15, brotnli outperforms xz by a small margin on the FSDFSD novel.

Although the ratio between the lowest performing parameters and the highest performing parameters is higher in the document generated by LLaMA3, as shown in Table 6.16, it is vastly inferior to the ratios observed on alice29 and Frankenstein. Since the document was generated by the underlying model, or rather from a fine-tuned derivative

TABLE 6.14: Variations in the size of the compressed FSDFSD document depending on the quantisation and context window of Llamazip

Largest size (in bytes)	Smallest size (in bytes)	Average size (in bytes)	Standard deviation (in bytes)	Ratio between largest and smallest
141,446	119,197	127,993	7,280	1.19

TABLE 6.15: Average size of the FSDFSD file over all context windows and quantisations depending on the final compression algorithm

Final compression algorithm	Average size of the compressed FSDFSD file (in bytes)	Ratio compared to gzip
brothli	112,517	0.88
xz	112,579	0.88
zstd	133,889	1.05
zpaq	167,268	1.31

of the underlying model, this result is not surprising, as Llamazip has to predict that it generated itself.

TABLE 6.16: Variations in the size of the compressed document generated by LLaMA3 depending on the quantisation and context window of Llamazip

Largest size (in bytes)	Smallest size (in bytes)	Average size (in bytes)	Standard deviation (in bytes)	Ratio between largest and smallest
100,948	66,055	78,408	11,567	1.53

As with the other files, brotli performs slightly better than xz, and both perform much better than the other algorithms, as shown in Table 6.17.

TABLE 6.17: Average size of the file generated by LLaMA over all context windows and quantisations depending on the final compression algorithm

Final compression algorithm	Average size of the compressed LLaMA-generated file (in bytes)	Ratio compared to gzip
brothli	67,107	0.86
xz	67,119	0.86
zstd	83,711	1.07
zpaq	101,120	1.29

Concerning the document generated by the Mistral model fine-tuned to answer questions, the ratio is similar to the one observed for the document generated by LLaMA3, as shown in Table 6.18. Although Mistral and LLaMA3 do not share the same architecture, the underlying training datasets intersect, and since the prompt was kept identical to generate these documents, the performance of Llamazip is approximately the same.

Table 6.19 shows that brotli is not always the superior choice as the final compression algorithm for Llamazip, as xz performs slightly better.

TABLE 6.18: Variations in the size of the compressed document generated by Mistral depending on the quantisation and context window of Llamazip

Largest size (in bytes)	Smallest size (in bytes)	Average size (in bytes)	Standard deviation (in bytes)	Ratio between largest and smallest
134,335	85,632	102,153	16,208	1.57

TABLE 6.19: Average size of the file generated by Mistral over all context windows and quantisations depending on the final compression algorithm

Final compression algorithm	Average size of the compressed Mistral-generated file (in bytes)	Ratio compared to gzip
xz	87,799	0.86
brothli	88,042	0.86
zstd	107,103	1.05
zpaq	131,300	1.29

6.3.7 Speed

Table 6.20 displays the number of tokens compressed per second depending on the quantisation and context window. The compression speed is perfectly negatively correlated with the window size, with a Spearman’s ρ of -1.

Modern hardware has dedicated float32 accelerators, which can even perform two float16 operations in the same time frame required by a single float32 operation. However, the NVidia 4090, which is still to this day the latest and most high-end NVidia consumer-grade DL accelerator, does not have any int4 or int8 accelerators. Although such processing units might be integrated in the future, with quantisations becoming more and more common, for the time being, Llamazip compresses data at least as fast using float16 and bfloat16 quantisations than integer quantisations at equal context window sizes, while offering a better compression ratio.

TABLE 6.20: Number of tokens compressed per second using Llamazip depending on the context window and the quantisation using a NVidia 4090

Quantisation	Context window size (in tokens)							
	16	32	64	128	256	512	1024	2048
int4	10	10	9	8	6	4	2	1
int8	12	12	12	12	12	11	8	5
bfloat16	48	46	44	32	31	18	9	5
float16	49	46	44	39	31	18	9	5

6.3.8 Memory Requirements

The memory requirements displayed in Table 6.21 are based on the memory usage obtained using the command “nvidia-smi”, an official program provided by NVidia along the drivers in order. Consistent with expectations, both quantisations and the size of the context window have a positive impact on the quantity of memory used during compression, since an int4 parameter contains 4 times fewer bits than a float16 one and

that the context window has to be entirely kept in memory to predict the next token. As expected, bfloat16 and float16 require the exact same amount of memory, since both formats use 16 bits per parameter. Each larger quantisation size, even with the smallest context window, requires more memory than the smaller size with the longest context window.

TABLE 6.21: Size (in MiB) of the memory required to compress a file using Llamazip depending on the context window and the quantisation

Quantisation	Context window size (in tokens)							
	16	32	64	128	256	512	1024	2048
int4	6,751	6,753	6,759	6,769	6,781	6,815	7,037	7,485
int8	9,275	9,277	9,285	9,297	9,313	9,395	9,881	12,321
bfloat16	15,943	15,945	15,949	15,961	15,993	16,077	16,485	18,291
float16	15,943	15,945	15,949	15,961	15,993	16,077	16,485	18,291

6.3.9 Optimal Parameters

The ranking of the different quantisation methods has remained the same, regardless of the compressed file. Unsurprisingly, the more parameters are converted to a smaller number of bits, the poorer the predictions. This necessarily means that more tokens are stored in the files and that the average values of the counters are low. For example, the int4 quantisation results in files that are heavier than the int8 quantisation, which is itself heavier than the float16 and bfloat16 quantisations. The difference between the two is less than 1%, but is still in favour of float16.

As far as the size of the window context is concerned, here again it comes as no surprise that the longer it is, the more accurately it can predict the next token, thereby reducing the number of tokens to be registered.

On another note, the best final compression algorithm is brotli, which almost saves an additional 10% in size compared to gzip. It follows logically that, of all the parameters tested, Llamazip achieves the best performance with a quantised float16 model, a context window of 2048 tokens, and final compression with brotli.

It is likely that using a larger context window and using the non-quantised model, in float32, would be even more efficient, but the hardware available does not allow us to be sure of this, since the memory required exceeds 24 GiB.

6.3.10 Bytes Per Character

The Bytes Per Character metric is a fundamental measure in text compression as it provides a straightforward way to evaluate the efficiency of a compression method. This metric quantifies the average number of bytes required to represent each character in the compressed text, providing a clear and interpretable indicator of compression performance. This metric is particularly valuable for comparing methods across datasets of varying lengths and content, as it normalises the compression efficiency relative to the input size.

Table 6.22 displays the number of bytes per character required by different text compression algorithms depending on the document. Concerning the enwik9 document, the 10 batches of 100,000 tokens each are concatenated.

TABLE 6.22: Number of bytes necessary to represent one character depending on the document and the compression algorithm

Document	Characters	Llamazip	brothli	xz	zstd	zpaq	nncp
alice29	148,479	0.025	0.310	0.322	0.379	0.440	0.495
Frankenstein	441,034	0.037	0.312	0.321	0.373	0.428	0.441
FSDFFSD	441,034	0.270	0.296	0.305	0.358	0.406	0.438
LLaMA	619,631	0.148	0.191	0.194	0.261	0.280	0.281
Mistral	565,146	0.151	0.213	0.217	0.283	0.306	0.386
enwik9	3,407,633	0.185	0.238	0.242	0.305	0.326	0.235

On a document that is likely not part of the underlying a [LLaMA](#) training dataset, LLMZip compresses a document using 0.8426 bits per character, that is, 0.105 Bytes Per Character. This shows that recording the rank of the prediction rather than recording the token itself if the prediction is wrong is a more effective strategy for compressing the data. However, the work done on LLMZip has focused solely on text compression, whereas the present study finds that Llamazip can have other uses, in particular as a means of detecting that a document is part of the training set of an [LLM](#).

6.3.11 Other Usages of Llamazip

Although Llamazip was designed and thought of as a lossless text compression algorithm, it became apparent during observations that it might have other usages. Since the compression ratio is mainly determined by the number of correct guesses of [LLM](#), Llamazip can serve as a metric for [NTP](#) tasks. Moreover, the results obtained in Section 6.3.5 demonstrate a clear discrepancy in efficiency, measured by the compression ratio, between documents that are part of the training set of the underlying model, and between documents that are not.

Benchmark Algorithm

Any file compressed with Llamazip can be leveraged as a criterion. In fact, any string starting with the character “@” is a failed prediction that had to be recorded directly in the file, while all the other strings represent the number of correct guesses. A simple snippet of code is enough to retrieve the percentage of accurate predictions.

```
with gzip.open(f"alice29_f16_2048_10.txt.gz", "rb") as f:
    content = f.read()
    n_success = 0
    n_total = 0
    for t in content.decode("utf-8").split(" "):
        if t[0] == "@":
            n_total += 1
```

```

else:
    n_total += int(t)
    n_success += int(t)

print(f"Tokens predicted: {round((n_success/n_total) * 100, 2)}")

```

CODE LISTING 6.1: Python implementation to measure the number of correct predictions from a Llamazip-compressed file

Table 6.23 presents the percentage of tokens that are accurately predicted using a float16 quantisation. This percentage varies widely between documents, and the variance is especially large when the document is part of the training set. This percentage is essentially a score that could be used to compare LLMs trained on NTP tasks and also, as is the case here, the impact of quantisation on performance. The scores obtained on the two generated documents are extremely close.

TABLE 6.23: Percentage of correct predictions depending on the document and the context window using float16 quantisation

Document	Context window size (in tokens)							
	16	32	64	128	256	512	1024	2048
alice29	45.57%	67.91%	85.71%	93.23%	95.79%	96.65%	96.92%	97.04%
frankenstein	38.99%	51.79%	66.43%	81.48%	90.55%	93.81%	94.72%	94.98%
FSD FSD	32.01%	35.97%	41.14%	45.02%	47.28%	48.75%	49.89%	50.81%
LLaMA3	46.51%	52.85%	59.58%	65.13%	68.02%	69.71%	71.06%	71.8%
Mistral	46.54%	53.4%	60.74%	66.28%	69.14%	70.71%	71.83%	72.16%

Identifying Documents in Training Data of Large Language Models

The performance of the Llamazip algorithm vary widely depending on the document to compress, with previously seen data being compressed more than data that are outside of the training set.

The hypothesis proposed here is that the observed difference in compression ratios between the highest-performing and lowest-performing parameters of Llamazip, depending on whether the document is present in the underlying training set, is attributed to the model’s memorisation of the document. By intentionally limiting the model’s capacity to recall its training set—through a reduction in parameter precision—its access to memory is diminished. As a result, when the memory of the training set has little or no influence on compressing new data, the difference in compression ratios is minimal. Conversely, when the data has been previously encountered and are stored in memory, this reduction in capacity leads to a significant difference in compression ratios.

Three key indicators emerge:

- As demonstrated in 6.3.6, the ratio between the size of the file compressed with the worse Llamazip parameters, which are a int4 quantisation along a context window of only 16 tokens, and the best parameters, which consist in float16 quantisation and a context window of 2048 tokens, is larger on samples from the training set;

- There is a larger difference in the percentage of correct predictions when the context window is large and when it is small when the file is part of the training set;
- Llamazip outperforms other non-neural algorithms such as xz and brotli more on a previously seen file.

A reliable way of ensuring that a document is not included in the LLM training set is to select it according to its publication date, since publication after the model’s training date ensures that it is not included in the training set. However, it does not exclude documents that are partially or entirely generated with the help of LLMs. Direct to Open [238], or D2O, is an initiative of MIT Press that offers free and open access to recently published books.

At the time the present research was being carried out, the two most recent open access books published are *The Aesthetics of Stealth: Digital Culture, Video Games, and the Politics of Perception* [367] and *Reimagining the More-Than-Human City: Stories from Singapore* [259], titles that will be abbreviated as MIT1 and MIT2, respectively. Although it cannot be fully ruled out, it is assumed that LLM-assisted writing did not play a significant role in the authorship of these books. The python library pypdf⁶ was used to retrieve the text from the PDF files of the books, by applying the “extract_text()” function to all the pages and concatenating the strings.

After conversion to text only, MIT1 contains 622,101 bytes and MIT2 699,783 bytes. Table 6.24 presents the sizes of these compressed text files depending on the settings of Llamazip. The observed ratios are very close and are much lower than those observed in the case of works present in the training dataset, such as Frankenstein and alice29, and are, on the contrary, close to the ratio observed for the FSDFSD file, published after the release of LLaMA3.

TABLE 6.24: Size of the books from MIT Press depending on the parameters from Llamazip

Book	Size using the lowest-performing parameters (in bytes)	Size using the highest-performing parameters (in bytes)	Ratio between the sizes
MIT1	194,050	157,315	1.23
MIT2	222,814	177,092	1.26

Figures 6.6 to 6.8, respectively, display the percentages of accurate predictions on the alice29, Frankenstein, and FSDFSD documents. The quantisation employed has a much larger effect on previously seen files than on new data, graphically showing why the ratio between the best and worst Llamazip parameters is larger for previously seen documents.

More research is required to establish the effectiveness of Llamazip as a tool to verify whether a document has been included in the training datasets of LLMs. Among the datasets used for LLaMA3 training is Common Crawl [67], a large-scale web archive generated by Web crawling. For the months of February and March 2024 alone, the

⁶<https://github.com/py-pdf/pypdfm>

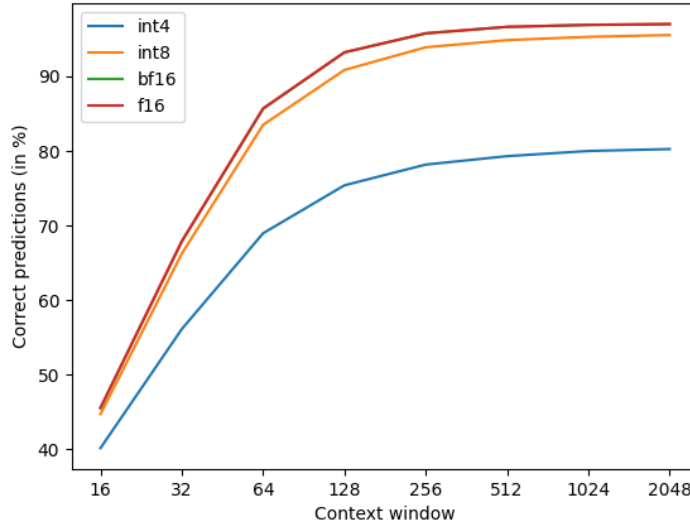


FIGURE 6.6: Percentage of correct predictions on the alice29 document depending on the quantisation and the context window

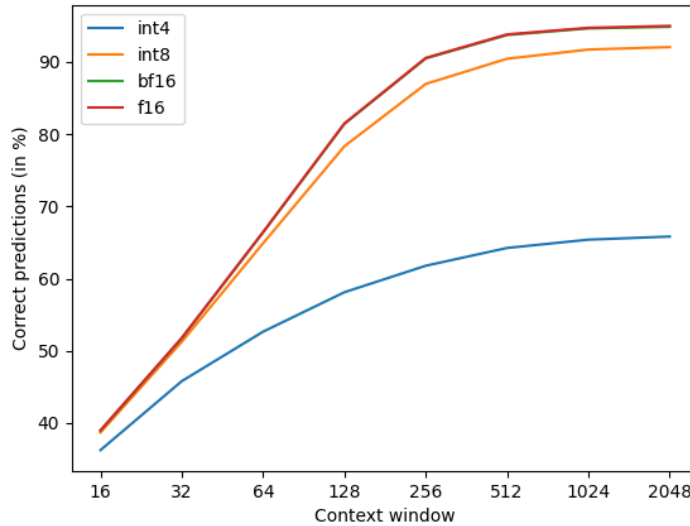


FIGURE 6.7: Percentage of correct predictions on the Frankenstein document depending on the quantisation and the context window

compressed archive exceeds 90 TiB in size, and the total size of the entire archive exceeds several petabytes of data. Due to the sheer scale of this dataset, comprehensive analysis and inspection of its content requires substantial resources, which makes it challenging to determine whether a specific document has been included in the [LLM](#) training set. Furthermore, not all datasets used for training are necessarily publicly accessible, depending on [LLM](#).

Due to concerns related to intellectual property and potential copyright infringement, it is crucial to detect the presence of a specific document within the training set of a [LLM](#), particularly for closed-source models that do not disclose their training data.

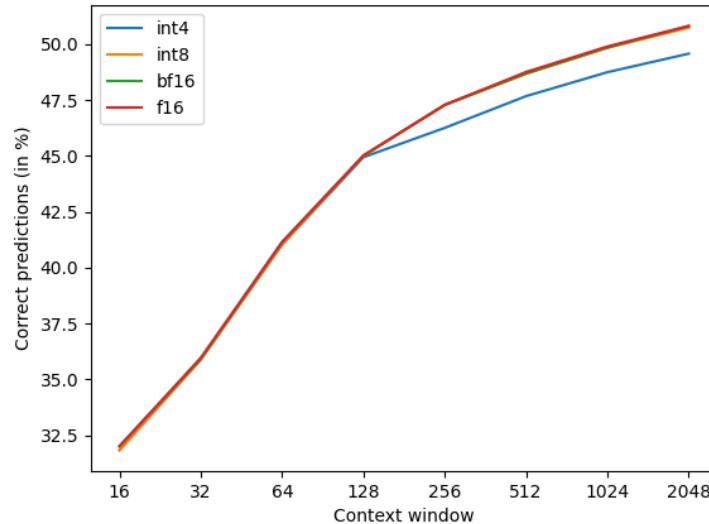


FIGURE 6.8: Percentage of correct predictions on the FSDFSF document depending on the quantisation and the context window

Insofar as the dataset used to train LLaMA3, although extracted from a public source, has not itself been published, and that for technical reasons linked to the size and storage of the data it would not have been possible to analyse it in any case, we cannot prove whether or not a document is in the training set and we can only hypothesise that the performance degradation is a clue to the presence of this document in the training set.

6.3.12 Limitations

As it stands, Llamazip suffers from a number of limitations. As has been pointed out, it is a slow and expensive algorithm that requires a graphics card with at least 24 GiB of memory to be used with the most powerful parameters tested as part of the study.

The size of the unquantised model is more than 15 GiB. Insofar as it is required for both decompression and compression, it considerably increases the size of compressed documents. This limit can be reduced if the same model is used for other purposes or becomes a pre-installed standard.

As LLaMA3 was mainly trained on English-language texts, performance varies widely from one language to another, making it much less flexible than the alternatives. Similarly, as a result of this training, characters in documents that are not in English could be out-of-vocabulary and therefore converted into `<unk>` tokens. With time, new words and idioms continuously appear. Without continuous training, which would break backward compatibility with previously compressed files, the compression ratio of Llamazip will inevitably decrease over time.

6.3.13 Future Work

The unquantised LLaMA3 model has not been tested due to hardware limitations. Further work is required to inspect the performance impact of using float32 weights.

Llamazip should register out-of-vocabulary tokens as they are, rather than registering the index as for other tokens, which would make the code only slightly more complex. This would mitigate any possible data loss due to the replacement of out-of-vocabulary tokens by `<unk>` tokens.

As the name suggests, Llamazip is an algorithm based on the LLaMA3 model. Other open-source models might obtain better compression performances and open the door to a larger language diversity. As mentioned in Section 3.2.6, at the time of writing, LLaMA3.1 is the latest available model from the LLaMA family, but it was not incorporated in this study as its recent release did not allow sufficient time for thorough evaluation.

As demonstrated in Section 6.3.11, it is possible that Llamazip can be used to detect whether a document was present in the original dataset of an LLM, at least in the case where it was trained on a NTP task. However, since the preliminary work is based on a single document, further work is needed to verify the reproducibility of this hypothesis. However, this task is made more difficult by the relative obscurity of the datasets used to train LLMs and the constant increase in the distribution of AI-generated content, which is not always labelled as such. This last point is of particular importance and projects such as Wordfreq⁷, which is a database of word frequencies for various natural languages, will not be updated by its author since generative models have “polluted” the data, which is no longer representative of human usage⁸. For example, it has been observed [321] that ChatGPT overuses the word “delve” compared to humans.

Llamazip could adopt the strategy used by a previous work, LLMZip, which is to save the rank of the correct token rather than saving the correct token in the event of a bad prediction.

Finally, it is also possible that Llamazip can be used to detect that a document has been generated using a LLM, since the compression ratio obtained is higher than a document that is not present in the dataset but lower than a document that is. With a larger sample of documents generated by a greater diversity of LLMs, this hypothesis could allow the detection of artificial text.

6.3.14 Conclusion

In conclusion, this section has demonstrated the multifaceted potential of Llamazip as a novel, lossless text compression algorithm that leverages the predictive capacity of LLaMA3. By only storing the tokens that the model fails to predict accurately, Llamazip achieves significant compression, optimising storage requirements without compromising

⁷<https://github.com/rspeer/wordfreq>

⁸<https://github.com/rspeer/wordfreq/blob/master/SUNSET.md>

data integrity. This approach to compression trades processing speed for maximal data reduction, making it well-suited for archival purposes where storage efficiency takes precedence over rapid decompression.

Beyond its primary function in data compression, Llamazip introduces an innovative capability to verify whether specific documents were included in the training set of an [LLM](#). By comparing its compression performance under various parameter settings, a discernible pattern emerges that differentiates known documents from previously unseen ones. This property positions Llamazip as a promising tool for identifying the provenance of data within an [LLM](#) training context, a crucial capability to address concerns about data privacy, intellectual property, and transparency of training sources. Moreover, this section has also explored the effects of model quantisation and context window size on Llamazip's performance, highlighting how these variables influence next-token prediction accuracy and, consequently, compression efficiency.

In general, Llamazip not only contributes to the field of lossless compression, but also broadens the scope of language model applications, bridging the gap between predictive modelling and practical storage solutions. This work has been submitted to a peer-reviewed conference for consideration.

How can [Large Language Models](#) be adapted for both efficient text compression and dataset membership inference through quantisation analysis?

It is possible to adapt [LLMs](#) for lossless text compression by leveraging their [Next Token Prediction](#) capabilities to store only incorrect predictions, thereby minimising storage requirements. By varying the degree of quantisation and context window size, the model's prediction accuracy—and hence compression efficiency—can be systematically analysed. This quantisation-based performance analysis reveals distinct compression patterns between previously seen and unseen documents, enabling the inference of dataset membership. Thus, quantisation serves as a tool not only to optimise compression trade-offs but also to assess whether a document was likely part of the training set of the underlying [LLM](#).

Chapter 7

Conclusion

7.1 Answers to the Research Questions

The research questions outlined in Chapter 1 are revisited and addressed in light of the progress and advancements achieved throughout this work.

How can non-neural evaluation metrics and Large Language Models be leveraged to improve the assessment of machine translation quality while maintaining efficiency?

Tokengram_F, is a metric based on chrF++ explored in Chapter 3. Rather than comparing complete words between the hypothesis and the reference, Tokengram_F relies on modern tokenization algorithms to divide words into frequent occurrences of subwords. Our work has explored how vocabulary size and choice of tokenization algorithm affect Tokengram_F. Although Tokengram_F was not the best performing metric in the WMT22 competition, in which it participated, it can assess the quality of machine-generated translations in 240 different languages, more than twice the number of languages supported by the COMET family, the current state-of-the-art. This compatibility with a wide range of languages paves the way for future research focused on the application to low-resource language pairs, which are challenging to evaluate using neural metrics that require large datasets.

Our work proved that the use of a pre-trained tokenizer adds valuable information to a simple evaluation algorithm based on the overlap of subwords.

Chapter 3 also introduces **Embed_llama**, a metric which relies on a pre-trained LLM, namely LLaMA2, to evaluate the quality of machine-generated translations. Inspired by Word2Vec, it compares the cosine distance between the vector representations of the hypothesis and the reference to determine the semantic proximity. Our work explored how the number of blocks, that is, the depth of the underlying model, can influence the correlations between Embed_llama and human rankings. This metric enables the exploration of the reusability of large models, which are particularly costly to train,

to diversify the range of tasks and deepen research on transfer learning and feature extraction.

Our work demonstrates that a non-fine-tuned, pre-trained [LLM](#) produces vector representations that are closely aligned for semantically similar sentences from different languages. This capability allows for an estimation of the quality of machine-generated translations.

How can techniques from image style transfer, such as CycleGAN, be adapted for unsupervised [Machine Translation](#) in the absence of parallel datasets?

[CycleGN](#), introduced in Chapter 4, is an adaptation of the CycleGAN framework for text, based on two transformer models without the need for a discriminator. Using a [Cycle-Consistency Loss](#) and utilising pre-training through [Masked Language Modeling](#), the CycleGN framework enables the training of unsupervised translation models, even when trained on datasets that do not share any parallel sentences.

The results obtained across a wide variety of language pairs across different alphabets demonstrate that CycleGN successfully achieves unsupervised [Machine Translation](#) and generates better translations than competing frameworks.

How can style transfer methods originally designed for images be adapted for text style transfer tasks?

[CycleGN](#) has been established to achieve unsupervised [Machine Translation](#) on some but not all language pairs examined. Chapter 5 examines how this framework can be applied, without any modifications, to text style transfer tasks, such as informal-to-formal and negative-to-positive transformations. Although the generated sentences remain particularly close to the originals, resulting in excellent meaning preservation scores, the style transfer is only partial.

These results demonstrate that CycleGN, a framework itself derived from an image style transfer framework, can be effectively adapted for both unsupervised [Machine Translation](#) and text style transfer.

How can [Large Language Models](#) be adapted for both efficient text compression and dataset membership inference through quantisation analysis?

Chapter 6 investigates how a model designed solely to predict the next token based on a sequence of preceding tokens—the foundation of current [LLMs](#)—can be repurposed for text compression. The proposed algorithm, [Llamazip](#), leverages an underlying model, [LLaMA3](#), to compress the text by storing only the mispredicted tokens. This

list of mispredicted tokens is then further compressed using a standard lossless text compression algorithm. Llamazip also examines how the size of the context window influences the compression performance in different textual documents. Similarly to Embed_llama, this work advances research on the transfer learning capabilities of large models, which are often challenging to replicate, aiming for broader generalisation of these architectures.

Llamazip achieves significantly better compression ratios than widely used text compression algorithms, albeit at the cost of considerably higher computational time.

The work presented in Chapter 6 also examines how the quantisation of the underlying model, LLaMA3, impacts the compression ratio achieved by Llamazip. Documents that likely belonged to the LLaMA3 training set exhibited significantly better compression ratios compared to others. Furthermore, it was observed that the compression ratio of these training-set documents degraded much more under strong quantisation than for documents published after the release date of LLaMA3. This phenomenon can be attributed to the model’s memory, which diminishes with increased quantisation, reducing its ability to “recall” a document when heavily quantised.

These findings suggest that it is possible to infer whether a document likely belongs to a training set based on the degradation in compression performance. However, it should be noted that due to technical and hardware constraints, this work was unable to directly explore the training set of LLaMA3 to confirm whether the documents identified as likely members are indeed present, and if so, whether they appear multiple times.

7.2 Research Contributions

Throughout this thesis, the various chapters have explored different aspects of Natural Language Processing. The contributions of this research are summarised in this section.

Chapter 3: Machine Translation Evaluation

- We introduce a novel evaluation metric for Machine Translation, Tokengram_F, based on the premise that comparing subwords extracted from trained tokenizer is more effective than comparing complete words to assess the quality of a machine-generated translation.
- Embed_llama, another evaluation metric for Machine Translation that explores the ability of a non-fine-tuned pre-trained LLM to capture the semantic proximity of sentences independently of their languages.

Chapter 4: Unsupervised Machine Translation

- In this work, we introduce a distinction between two different types of non-parallel datasets. Although permuted datasets are non-permuted, each sentence has its corresponding translation at an unknown index. On the other hand, non-intersecting

datasets are the “worst-case scenarios” of non-parallel datasets, as no sentence has any translation in the dataset.

- We present **CycleGN**, a novel approach for unsupervised [Machine Translation](#), that can learn from both permuted and non-intersecting datasets. CycleGN is based on a [Cycle-Consistency Loss](#), similarly to the original CycleGAN, and is based on [Masked Language Modeling](#) pre-training to avoid discriminator requirement.
- We show how CycleGN performs better than other unsupervised [Machine Translation](#) frameworks.
- We show the performance of CycleGN on Low-Resource languages and present strategies to further improve translations on these specific language pairs.

Chapter 5: Unsupervised Style Transfer

- We introduce the text style transfer capabilities of CycleGN and how this framework can be kept identical for two different [Natural Language Processing](#) tasks.
- We explore how image-based cyclic loss methods, as used in CycleGAN, can be effectively adapted for text.
- We present the experiments conducted on formality transfer using datasets such as Yelp, [GYAFC](#) and XFORMAL on multiple languages, namely English, French, and Italian.
- We report results comparing CycleGN with other state-of-the-art unsupervised models, demonstrating the strength of CycleGN in meaning preservation.
- We provide qualitative examples of CycleGN outputs to better illustrate how the model performs subtle style modifications and reflect on its limitations, especially in achieving higher style transfer intensity, while maintaining semantic fidelity.
- We detail the challenges faced during the evaluation, including reproducibility issues and the reliance on legacy software and closed-source resources.

Chapter 6: Neural Text Compression

- We introduce **Llamazip**, a lossless text compression algorithm that uses a non-fine-tuned pre-trained [LLM](#) to compress text. By only storing tokens that are mispredicted, Llamazip obtains better compression ratios than non-neural algorithms.
- We evaluate thoroughly Llamazip to explore the effects of settings such as the quantisation type and the size of the context window on the compression ratio.
- We demonstrate how Llamazip can be used as a benchmark for any model that is trained on a [Next Token Prediction](#) task.

- We show how the degradation of performances of Llamazip depending on the context window and the quantisation can likely indicate whether the compressed document has been previously seen by the underlying model [LLaMA3](#), and as such whether the document was part of its training set.

7.3 Future Research

In the following list, we identify the areas for improvement that we see for the work carried out during this thesis, as well as more broadly in the field of [Natural Language Processing](#) and [Machine Learning](#) in general.

- **Continuous improvement through the use of more efficient [LLMs](#).** The work carried out on **Embed_llama** and **Llamazip** is evolutionary in nature, since it is based on underlying pre-trained models. These two projects would benefit from larger open-source [LLMs](#) trained on more diverse datasets, in one case to better estimate the semantic proximity between similar sentences from different languages, and in the other to enable better compression regardless of the language. Given that open-source [LLMs](#) have only recently become widespread, we hope that many performance gains remain to be achieved. In addition, these two projects will enable us to further develop the generalisability of these models, with no fine-tuning required, i.e. their ability to be used for different tasks without re-training. A backbone model could be designed to be installed on each computer, allowing it to be reused as-is or with additional layers for a variety of tasks. This approach would leverage dedicated hardware for faster computations while maintaining a small storage footprint, as the same model would support multiple applications.
- **Increasing the volume of monolingual datasets for low-resource languages.** By requiring very little data to train a tokenizer rather than a neural network, **Tokengram_F** is at the crossroads between traditional untrained metrics, which are compatible with all languages, and neural metrics. Although **Tokengram_F** can already be used to evaluate 240 different languages, the increasing number of open-source datasets for low-resource languages should make it possible to increase this number. Future work should focus on evaluating low-resource languages using small datasets, incorporating other tokenisation algorithms such as WordPiece [80], or using tokenisation ensembling techniques. It is also possible that adapting the metric's parameters to each language, rather than using the same parameters for all languages as is currently the case, could improve correlations with human judgements.
- **Optimising CycleGN performance to accelerate training.** The **CycleGN** framework is particularly expensive to train, making it difficult to experiment with and improve, and in particular to compare performance between different iterations. The first step in improving CycleGN is to increase its performance, i.e., the number of sentences processed per second and the training time per epoch.

Although performance should improve with continued hardware improvements, the software optimisations listed in Section 4.8 are worth exploring.

- **Improving stability of CycleGN.** The instability caused by the cyclic nature of CycleGN makes it crucial to closely monitor training, to avoid stopping during a sudden collapse, and probably increases the duration of training. Moreover, particularly in the case of non-intersecting models, this instability prevents the model from achieving better performance. The solutions mentioned in Section 4.8, such as increasing batch size, could alleviate this difficulty and improve translations.
- **Exploring transfer strength style factors in CycleGN.** In its current form, CycleGN retains the meaning of sentences too literally and does not transfer the style sufficiently. Future work should examine the factors that determine this balance, in order to increase text style transfer.
- **Developing a metric to unify text style transfers.** Future research should focus on creating neural metrics capable of evaluating style transfer quality across various domains, similar to COMET, which assesses translation quality for multiple languages and alphabets. Such metrics could assess diverse types of style transfer, including sentiment or formality adjustments, thus fostering more robust and versatile evaluation frameworks for this field.
- **Analyse the contents of the training dataset of an LLM to determine which documents are contained within it.** Future work needs to be carried out on Llamazip’s analysis capabilities, to prove that the degradation of its performance according to context window size and quantisation can indeed identify whether a document is part of the underlying LLM training set. This work could also look at a potential relationship between the number of occurrences of a document in the training set and the compression ratio, as well as refining the degradation thresholds to be reached before considering whether or not a document belongs to the training set.
- **Improving the reproducibility of work in Machine Learning.** To ensure the longevity of the work and enable more reliable and accurate comparisons across projects, it is essential to guarantee that the results can be consistently reproduced. Areas for improvement are mentioned in Section 7.4.

7.4 Closing Remarks

Recent improvements in the field of Natural Language Processing, and Machine Translation to some extent are mainly due to the arrival of the Transformer architecture, which has also revolutionised other areas of Machine Learning with vision transformers and multimodal transformer models. By enabling the entire training process to be parallelised and taking advantage of recent hardware advances, these models can have many more parameters and be trained on larger datasets than previous architectures. In this thesis, we emphasise optimising the training modalities of the current Transformer architecture rather than exploring an alternative one, in order to increase its performance

on different datasets, paving the way towards the training of low-resource languages, and also for the increase of synthetic parallel datasets, generated by backtranslation by models trained exclusively on non-parallel datasets. By demonstrating that it is possible to train a translation model from data where there is not a single pair of sentences in the entire dataset, what we refer to as a “non-intersecting dataset”, which is to our knowledge the first example of such a model, our work considerably increases the volume of data that can be used for [Machine Translation](#), since monolingual datasets are necessarily at least as large as parallel datasets. In addition, although future research is needed to overcome the problems associated with excessive proximity or, on the contrary, class imbalance, CycleGN could be a particularly appropriate architecture for low-resource languages since it does not require parallel data.

Our work on the automatic evaluation of machine-generated translations and text compression aims to extend the applicability of computationally expensive models, specifically to reduce both the financial costs and environmental impact associated with their training. In the future, we hope that a small number of models will be able to perform the vast majority of tasks, in a manner analogous to the Pareto principle [389], according to which 80% of effects are the product of 20% of causes.

In Chapter 4, we discuss the problems associated with the reproducibility of [Machine Learning](#) work. The concerns raised are (i) that the authors are frequently unreachable because email addresses are no longer attributed, (ii) that the authors rarely reply, (iii) that the webpages to download trained models are no longer accessible, (iv) that the source codes or datasets are not always published and that even when they are published, (v) the code is difficult to execute because of “software decay”, or (vi) the results produced by executing the provided scripts often do not match the outcomes reported in the publications. These problems have been identified in the past [271] and are so common that an annual conference, the ML Reproducibility Challenge, has been held since 2018 to try to remedy them [48].

We present possible solutions to these problems. (i) The problem related to the disappearance of email addresses could be corrected by the systematic implementation of email redirection, or by the use of tools such as ORCID iD [256], which provide a unique persistent identifier for researchers. (ii) It seems difficult to “force the authors” to respond, and accidents or career changes can occur. The preferable solution would be to make contact with the authors dispensable insofar as the work should be reproducible without requiring any contact. (iii) It would be possible to ask the academic journal to host the models themselves, but these are large quantities of data, and this would be a costly operation. Furthermore, if the scripts and data are published and enable the model to be replicated, it is not imperative that the models are systematically downloadable, even if this is a practice that should be encouraged. (iv) Problems related to the publication of datasets and code are the easiest to resolve since all that is required is that academic journals mandate the publication of data and code. It would be preferable if the journals also asked the authors to provide a complete list of all the libraries and languages used, as well as to indicate the exact version used for each. It would also be preferable for the journals to host an archive of these data. (v) and

(vi) are by far the most difficult to solve. “Software decay” can be caused by many different components of the code, for example, the lack of backward compatibility in the language itself or a necessary library, or the disappearance of the source code of a dependency. The non-reproducibility of results can also be due to a number of factors, such as changes in hardware or software developments that are not backward compatible but have not been marked as such. Although not without limitations, containerisation solutions like Docker partially address these challenges and have become widely adopted in the industry. A better theoretical solution, although difficult to implement and more computationally expensive, would be the creation of completely virtualised environments, where the hardware used could be emulated, so that the entire software stack on which academic work is based could be reproduced bit for bit, from kernel to firmware to dependencies.

Appendix A

Metric Results

A.1 Tokengram_F Results

TABLE A.1: Pearson correlations (r) and Kandall correlations (τ) obtained by Tokengram_F in the WMT20 challenge

Language pair	r (r)	τ (Tau)
en-cs	0.865	0.485
en-de	0.961	0.371
en-ru	0.981	0.162
en-ta	0.941	0.590
en-zh	0.851	0.403
en-ja	0.949	0.521
en-pl	0.958	0.256
en-iu	0.433	0.340
cs-en	0.872	0.095
de-en	0.997	0.440
pl-en	0.508	0.032
ta-en	0.957	0.184
km-en	0.984	0.281
ps-en	0.894	0.143
ja-en	0.972	0.251
ru-en	0.921	0.055
zh-en	0.960	0.130
iu-en	0.765	0.242

TABLE A.2: Pearson correlations (r) and Kandall correlations (τ) obtained by Tokengram_F in the WMT21 challenge

Language pair	r (r)	τ (Tau)
en-cs	0.978	0.549
en-zh	0.625	0.121
en-ha	0.760	0.185
en-ja	0.967	0.384
en-ru	0.756	0.214
en-de	0.842	0.448
cs-en	0.562	-0.052
zh-en	0.269	0.395
ha-en	0.921	0.021
ja-en	0.823	0.006
ru-en	0.579	-0.123
de-en	0.424	-0.151
fr-de	0.655	0.049
de-fr	0.504	0.111
bn-hi	0.949	0.079
hi-bn	0.877	0.335
xh-zu	0.999	0.306
zu-xh	0.997	0.529

A.2 Embed_llama Results

TABLE A.3: Pearson correlations (r) and Kandall correlations (τ) obtained by Token-gram_F in the WMT22 challenge

Language pair	r (r)	τ (Tau)
en-cs	0.602	0.077
en-zh	0.248	-0.044
en-hr	0.899	0.274
en-ja	0.927	0.241
en-liv	0.989	0.370
en-ru	0.852	0.659
en-uk	0.869	0.178
en-de	0.799	1.000
liv-en	0.985	0.500
zh-en	0.787	0.415
sah-ruh	1.000	0.856
uk-cs	0.971	0.350
cs-uk	0.921	0.311

TABLE A.4: Pearson correlations (r) for the WMT20 system dataset depending on the Llama layer.

The findings pertaining to the second layer are shown in bold, as it represents the prevailing default layer count within the Embed_Llama

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.855	0.866	0.875	0.877	0.873	0.876	0.872	0.863	0.861	0.864	0.867
en-de	0.923	0.928	0.933	0.929	0.928	0.927	0.927	0.926	0.925	0.925	0.928
en-ru	0.856	0.677	0.769	0.772	0.81	0.829	0.821	0.902	0.912	0.899	0.918
en-ta	0.883	0.901	0.914	0.921	0.926	0.932	0.934	0.936	0.939	0.942	0.945
en-zh	0.137	0.019	0.07	0.028	0.038	0.065	0.083	0.135	0.143	0.191	0.193
en-ja	0.898	0.892	0.879	0.871	0.88	0.881	0.874	0.883	0.88	0.876	0.871
en-pl	0.88	0.874	0.88	0.878	0.875	0.864	0.868	0.873	0.873	0.872	0.868
en-iu	0.252	0.226	0.194	0.156	0.137	0.121	0.119	0.106	0.105	0.096	0.094
cs-en	0.795	0.777	0.752	0.771	0.789	0.788	0.796	0.795	0.796	0.789	0.775
de-en	0.992	0.996	0.996	0.994	0.99	0.99	0.99	0.988	0.988	0.988	0.99
pl-en	0.419	0.403	0.428	0.433	0.401	0.395	0.4	0.394	0.388	0.398	0.407
ta-en	0.876	0.889	0.918	0.922	0.919	0.923	0.921	0.919	0.917	0.919	0.918
km-en	0.954	0.969	0.988	0.988	0.988	0.989	0.99	0.989	0.987	0.986	0.985
ps-en	0.926	0.874	0.877	0.862	0.872	0.877	0.873	0.875	0.875	0.875	0.878
ja-en	0.913	0.938	0.947	0.946	0.946	0.948	0.949	0.948	0.946	0.948	0.949
ru-en	0.949	0.939	0.949	0.948	0.953	0.947	0.948	0.949	0.948	0.944	0.939
iu-en	0.64	0.676	0.68	0.662	0.638	0.645	0.638	0.625	0.616	0.615	0.633

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.865	0.866	0.86	0.858	0.851	0.856	0.855	0.857	0.866	0.874	0.874
en-de	0.925	0.926	0.922	0.921	0.917	0.918	0.918	0.921	0.923	0.929	0.929
en-ru	0.922	0.904	0.903	0.898	0.902	0.885	0.862	0.902	0.93	0.941	0.938
en-ta	0.942	0.942	0.944	0.946	0.947	0.947	0.948	0.948	0.948	0.946	0.946
en-zh	0.202	0.181	0.195	0.2	0.193	0.173	0.162	0.163	0.149	0.122	0.125
en-ja	0.87	0.867	0.869	0.865	0.865	0.857	0.858	0.848	0.838	0.836	0.836
en-pl	0.873	0.871	0.875	0.872	0.869	0.86	0.865	0.862	0.858	0.857	0.857
en-iu	0.084	0.085	0.082	0.084	0.083	0.078	0.073	0.082	0.091	0.092	0.086
cs-en	0.781	0.787	0.787	0.794	0.802	0.794	0.8	0.79	0.783	0.776	0.785
de-en	0.989	0.989	0.988	0.988	0.986	0.989	0.989	0.991	0.992	0.994	0.994
pl-en	0.408	0.42	0.421	0.418	0.417	0.414	0.413	0.41	0.406	0.415	0.415
ta-en	0.918	0.917	0.914	0.91	0.907	0.912	0.912	0.913	0.913	0.919	0.921
km-en	0.983	0.982	0.975	0.975	0.967	0.964	0.967	0.968	0.97	0.966	0.968
ps-en	0.876	0.881	0.887	0.892	0.891	0.888	0.889	0.889	0.889	0.895	0.897
ja-en	0.947	0.946	0.944	0.941	0.936	0.94	0.942	0.943	0.942	0.941	0.941
ru-en	0.938	0.938	0.938	0.939	0.94	0.935	0.935	0.935	0.933	0.935	0.937
zh-en	0.95	0.951	0.951	0.95	0.952	0.953	0.951	0.954	0.955	0.958	0.958
iu-en	0.619	0.613	0.606	0.6	0.602	0.627	0.63	0.639	0.644	0.662	0.66

TABLE A.5: Kendall correlations (τ) for the WMT20 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.226	0.231	0.244	0.234	0.232	0.226	0.229	0.225	0.225	0.222	0.226
en-de	0.181	0.19	0.207	0.201	0.205	0.198	0.202	0.201	0.199	0.196	0.198
en-ru	0.028	0.034	0.037	0.045	0.038	0.045	0.042	0.044	0.043	0.046	0.05
en-ta	0.355	0.354	0.323	0.284	0.282	0.267	0.279	0.281	0.282	0.26	0.255
en-zh	0.147	0.141	0.164	0.144	0.138	0.141	0.147	0.152	0.152	0.16	0.165
en-ja	0.277	0.281	0.295	0.29	0.285	0.287	0.283	0.285	0.284	0.284	0.282
en-pl	0.097	0.095	0.103	0.101	0.101	0.102	0.098	0.101	0.098	0.094	0.102
en-iu	0.218	0.224	0.205	0.187	0.18	0.173	0.185	0.187	0.186	0.179	0.18
cs-en	0.064	0.065	0.073	0.072	0.077	0.076	0.076	0.073	0.074	0.078	0.083
de-en	0.349	0.374	0.387	0.384	0.39	0.389	0.385	0.376	0.372	0.375	0.381
pl-en	-0.016	-0.016	-0.0	-0.007	-0.0	0.0	-0.003	0.002	-0.002	-0.005	-0.006
ta-en	0.108	0.118	0.125	0.121	0.122	0.119	0.128	0.121	0.106	0.108	0.123
km-en	0.141	0.145	0.144	0.12	0.13	0.122	0.117	0.114	0.097	0.108	0.114
ps-en	0.088	0.081	0.074	0.076	0.058	0.053	0.06	0.061	0.065	0.061	0.08
ja-en	0.109	0.136	0.136	0.144	0.147	0.149	0.151	0.144	0.137	0.139	0.134
ru-en	0.011	0.02	0.019	0.022	0.026	0.022	0.031	0.023	0.02	0.02	0.017
zh-en	0.065	0.067	0.071	0.072	0.073	0.072	0.072	0.071	0.068	0.069	0.069
iu-en	0.16	0.161	0.175	0.178	0.187	0.191	0.195	0.194	0.195	0.198	0.199

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.23	0.229	0.222	0.223	0.215	0.218	0.221	0.222	0.221	0.222	0.221
en-de	0.19	0.193	0.187	0.183	0.177	0.178	0.184	0.184	0.186	0.186	0.19
en-ru	0.043	0.037	0.043	0.042	0.045	0.036	0.039	0.037	0.039	0.043	0.044
en-ta	0.281	0.29	0.283	0.266	0.261	0.267	0.264	0.232	0.204	0.224	0.237
en-zh	0.166	0.168	0.162	0.165	0.16	0.163	0.166	0.163	0.165	0.166	0.164
en-ja	0.282	0.276	0.28	0.281	0.274	0.272	0.271	0.271	0.263	0.266	0.27
en-pl	0.098	0.096	0.091	0.087	0.093	0.093	0.095	0.093	0.09	0.092	0.092
en-iu	0.17	0.167	0.162	0.163	0.154	0.153	0.149	0.146	0.141	0.145	0.143
cs-en	0.081	0.073	0.073	0.076	0.074	0.079	0.081	0.081	0.077	0.073	0.075
de-en	0.377	0.377	0.371	0.372	0.361	0.37	0.37	0.378	0.374	0.383	0.386
pl-en	-0.004	0.0	-0.001	0.005	0.001	-0.001	0.001	-0.004	-0.006	-0.004	-0.003
ta-en	0.118	0.111	0.107	0.101	0.103	0.11	0.112	0.114	0.112	0.106	0.103
km-en	0.109	0.11	0.092	0.097	0.083	0.091	0.1	0.111	0.116	0.124	0.126
ps-en	0.074	0.081	0.067	0.068	0.064	0.056	0.066	0.07	0.07	0.067	0.07
ja-en	0.128	0.125	0.116	0.112	0.102	0.115	0.117	0.125	0.126	0.129	0.129
ru-en	0.015	0.007	0.009	0.008	0.013	0.012	0.015	0.018	0.02	0.025	0.022
zh-en	0.069	0.072	0.07	0.069	0.066	0.069	0.07	0.072	0.07	0.074	0.074
iu-en	0.192	0.177	0.173	0.176	0.173	0.179	0.184	0.183	0.188	0.189	0.189

TABLE A.6: Pearson correlations (r) for the WMT21 system dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.985	0.984	0.985	0.982	0.981	0.981	0.982	0.982	0.979	0.98	0.982
en-zh	0.497	0.531	0.555	0.559	0.58	0.597	0.591	0.583	0.555	0.553	0.551
en-ha	0.534	0.551	0.567	0.567	0.567	0.553	0.554	0.553	0.563	0.56	0.547
en-ja	0.82	0.83	0.838	0.836	0.83	0.84	0.837	0.839	0.829	0.822	0.816
en-ru	0.567	0.621	0.682	0.674	0.674	0.658	0.674	0.614	0.572	0.576	0.573
en-de	0.818	0.793	0.804	0.801	0.794	0.789	0.794	0.799	0.796	0.792	0.79
cs-en	0.542	0.454	0.435	0.428	0.429	0.432	0.426	0.426	0.429	0.42	0.423
zh-en	0.232	0.186	0.155	0.159	0.172	0.179	0.188	0.196	0.198	0.196	0.192
ha-en	0.825	0.855	0.858	0.855	0.867	0.868	0.867	0.867	0.865	0.86	0.855
ja-en	0.728	0.726	0.748	0.753	0.761	0.76	0.759	0.759	0.761	0.761	0.759
ru-en	0.613	0.606	0.535	0.514	0.5	0.496	0.502	0.506	0.506	0.504	0.509
de-en	0.171	0.22	0.237	0.238	0.221	0.23	0.223	0.223	0.225	0.234	0.243
fr-de	0.564	0.556	0.555	0.556	0.556	0.555	0.557	0.555	0.556	0.554	0.552
de-fr	0.477	0.511	0.578	0.579	0.579	0.58	0.579	0.575	0.563	0.564	0.574
bn-hi	0.908	0.943	0.94	0.942	0.937	0.929	0.941	0.95	0.943	0.941	0.942
hi-bn	0.879	0.872	0.913	0.93	0.925	0.915	0.912	0.907	0.911	0.908	0.915
xh-zu	0.952	0.932	0.934	0.931	0.923	0.909	0.904	0.905	0.898	0.899	0.891
zu-xh	0.95	0.937	0.97	0.973	0.975	0.972	0.971	0.972	0.976	0.976	0.976

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.983	0.984	0.983	0.983	0.979	0.981	0.982	0.983	0.983	0.984	0.983
en-zh	0.548	0.545	0.548	0.545	0.545	0.549	0.554	0.543	0.533	0.524	0.528
en-ha	0.541	0.543	0.554	0.553	0.571	0.562	0.572	0.567	0.556	0.542	0.534
en-ja	0.81	0.804	0.795	0.793	0.781	0.783	0.786	0.777	0.772	0.774	0.779
en-ru	0.581	0.582	0.562	0.58	0.526	0.557	0.565	0.603	0.621	0.625	0.637
en-de	0.796	0.803	0.808	0.808	0.814	0.81	0.81	0.8	0.79	0.787	0.791
cs-en	0.394	0.386	0.383	0.387	0.388	0.395	0.41	0.425	0.422	0.433	0.435
zh-en	0.189	0.189	0.185	0.189	0.19	0.173	0.174	0.168	0.157	0.142	0.146
ha-en	0.846	0.842	0.84	0.841	0.841	0.827	0.828	0.829	0.823	0.818	0.814
ja-en	0.76	0.763	0.768	0.771	0.775	0.772	0.772	0.77	0.769	0.762	0.763
ru-en	0.51	0.512	0.513	0.52	0.521	0.52	0.529	0.534	0.535	0.532	0.528
de-en	0.23	0.217	0.212	0.196	0.184	0.207	0.21	0.205	0.203	0.215	0.214
fr-de	0.553	0.554	0.555	0.555	0.553	0.55	0.55	0.547	0.543	0.541	0.542
de-fr	0.566	0.564	0.558	0.558	0.556	0.567	0.564	0.561	0.563	0.574	0.577
bn-hi	0.935	0.932	0.931	0.931	0.93	0.933	0.933	0.938	0.94	0.93	0.922
hi-bn	0.904	0.892	0.876	0.86	0.869	0.863	0.859	0.865	0.864	0.852	0.829
xh-zu	0.9	0.903	0.906	0.903	0.909	0.905	0.908	0.912	0.903	0.898	0.896
zu-xh	0.977	0.979	0.981	0.981	0.981	0.981	0.977	0.977	0.976	0.969	0.962

TABLE A.7: Kendall correlations (τ) for the WMT21 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.197	0.198	0.243	0.243	0.238	0.223	0.221	0.219	0.219	0.214	0.22
en-zh	0.028	0.028	0.048	0.042	0.042	0.044	0.039	0.041	0.039	0.037	0.037
en-ha	0.073	0.077	0.087	0.081	0.077	0.071	0.071	0.068	0.062	0.061	0.062
en-ja	0.181	0.184	0.204	0.21	0.214	0.204	0.197	0.196	0.204	0.196	0.192
en-ru	0.094	0.101	0.099	0.099	0.089	0.09	0.091	0.086	0.081	0.087	0.084
en-de	0.241	0.31	0.172	0.172	0.241	0.31	0.31	0.241	0.241	0.241	0.241
cs-en	-0.042	-0.048	-0.049	-0.045	-0.041	-0.044	-0.042	-0.05	-0.046	-0.045	-0.051
zh-en	0.319	0.286	0.319	0.384	0.395	0.319	0.308	0.319	0.384	0.297	0.276
ha-en	-0.022	-0.016	-0.007	-0.003	-0.003	-0.009	-0.008	-0.007	-0.011	-0.014	-0.011
ja-en	-0.028	-0.021	-0.019	-0.018	-0.015	-0.017	-0.014	-0.015	-0.014	-0.012	-0.015
ru-en	-0.121	-0.124	-0.12	-0.12	-0.117	-0.123	-0.12	-0.119	-0.121	-0.124	-0.122
de-en	-0.155	-0.158	-0.151	-0.148	-0.153	-0.157	-0.155	-0.156	-0.158	-0.155	-0.152
fr-de	0.057	0.058	0.055	0.052	0.044	0.055	0.062	0.057	0.053	0.047	0.051
de-fr	0.054	0.066	0.082	0.079	0.075	0.051	0.057	0.054	0.049	0.055	0.055
bn-hi	-0.006	0.007	0.016	0.026	0.026	0.024	0.024	0.022	0.015	0.023	0.014
hi-bn	0.132	0.119	0.126	0.135	0.139	0.137	0.145	0.15	0.147	0.146	0.14
xh-zu	0.128	0.129	0.139	0.13	0.118	0.104	0.101	0.101	0.09	0.082	0.065
zu-xh	0.169	0.139	0.181	0.181	0.16	0.129	0.122	0.116	0.12	0.118	0.109

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.213	0.218	0.213	0.22	0.198	0.209	0.209	0.196	0.193	0.194	0.197
en-zh	0.042	0.045	0.048	0.048	0.043	0.043	0.043	0.046	0.041	0.04	0.04
en-ha	0.058	0.06	0.061	0.061	0.061	0.06	0.062	0.064	0.068	0.067	0.067
en-ja	0.189	0.187	0.196	0.193	0.202	0.196	0.195	0.19	0.184	0.179	0.179
en-ru	0.088	0.089	0.109	0.107	0.116	0.116	0.109	0.106	0.108	0.104	0.105
en-de	0.172	0.172	0.172	0.172	0.034	0.034	0.034	0.034	0.034	0.034	0.034
cs-en	-0.047	-0.048	-0.045	-0.055	-0.054	-0.06	-0.056	-0.055	-0.053	-0.058	-0.058
zh-en	0.297	0.319	0.265	0.265	0.33	0.286	0.362	0.286	0.297	0.276	0.276
ha-en	-0.014	-0.013	-0.011	-0.014	-0.012	-0.015	-0.014	-0.014	-0.014	-0.014	-0.014
ja-en	-0.016	-0.02	-0.021	-0.02	-0.02	-0.022	-0.022	-0.021	-0.02	-0.021	-0.018
ru-en	-0.121	-0.121	-0.123	-0.123	-0.123	-0.126	-0.124	-0.116	-0.116	-0.12	-0.121
de-en	-0.152	-0.152	-0.152	-0.154	-0.159	-0.154	-0.154	-0.155	-0.156	-0.155	-0.157
fr-de	0.049	0.047	0.043	0.041	0.031	0.034	0.032	0.036	0.032	0.032	0.031
de-fr	0.039	0.04	0.035	0.039	0.049	0.046	0.053	0.046	0.046	0.046	0.052
bn-hi	0.015	0.02	0.022	0.022	0.019	0.02	0.023	0.023	0.027	0.025	0.025
hi-bn	0.123	0.114	0.114	0.113	0.114	0.12	0.12	0.116	0.108	0.091	0.074
xh-zu	0.076	0.083	0.081	0.094	0.096	0.088	0.098	0.089	0.085	0.083	0.07
zu-xh	0.118	0.122	0.14	0.131	0.141	0.138	0.141	0.144	0.137	0.125	0.122

TABLE A.8: Pearson correlations (r) for the WMT22 system dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.484	0.477	0.469	0.458	0.451	0.443	0.442	0.437	0.434	0.434	0.429
en-zh	0.069	0.091	0.115	0.122	0.113	0.118	0.119	0.111	0.11	0.115	0.123
en-hr	0.866	0.862	0.869	0.868	0.875	0.875	0.869	0.871	0.868	0.871	0.87
en-ja	0.911	0.914	0.909	0.901	0.906	0.909	0.911	0.905	0.896	0.893	0.896
en-liv	0.943	0.946	0.941	0.938	0.94	0.943	0.943	0.944	0.945	0.945	0.944
en-ru	0.798	0.787	0.792	0.794	0.798	0.793	0.793	0.797	0.8	0.798	0.793
en-uk	0.816	0.828	0.832	0.825	0.818	0.804	0.803	0.796	0.795	0.789	0.788
en-de	0.598	0.602	0.62	0.641	0.644	0.654	0.651	0.659	0.673	0.672	0.672
liv-en	0.987	0.984	0.978	0.979	0.979	0.976	0.978	0.98	0.982	0.981	0.978
zh-en	0.715	0.777	0.807	0.821	0.83	0.841	0.842	0.851	0.852	0.85	0.848
sah-ru	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
uk-cs	0.857	0.849	0.854	0.851	0.854	0.861	0.861	0.858	0.855	0.856	0.857
cs-uk	0.851	0.846	0.836	0.841	0.84	0.839	0.838	0.836	0.835	0.836	0.834

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.429	0.431	0.436	0.44	0.44	0.44	0.445	0.444	0.439	0.437	0.439
en-zh	0.119	0.123	0.108	0.114	0.108	0.116	0.122	0.136	0.148	0.149	0.148
en-hr	0.871	0.87	0.872	0.873	0.877	0.878	0.879	0.881	0.881	0.879	0.879
en-ja	0.892	0.893	0.881	0.879	0.872	0.877	0.88	0.884	0.885	0.889	0.891
en-liv	0.944	0.946	0.946	0.947	0.947	0.946	0.949	0.947	0.943	0.942	0.942
en-ru	0.795	0.794	0.798	0.795	0.798	0.795	0.796	0.789	0.785	0.782	0.782
en-uk	0.789	0.79	0.793	0.794	0.794	0.793	0.798	0.798	0.796	0.796	0.794
en-de	0.668	0.666	0.665	0.658	0.666	0.664	0.66	0.652	0.648	0.644	0.644
liv-en	0.977	0.977	0.978	0.977	0.978	0.977	0.98	0.978	0.977	0.975	0.975
zh-en	0.845	0.844	0.845	0.842	0.842	0.839	0.84	0.836	0.828	0.826	0.83
sah-ru	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
uk-cs	0.855	0.855	0.855	0.855	0.854	0.853	0.853	0.851	0.848	0.846	0.848
cs-uk	0.832	0.832	0.83	0.831	0.828	0.829	0.83	0.831	0.83	0.83	0.832

TABLE A.9: Kendall correlations (τ) for the WMT22 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.014	0.014	0.023	0.024	0.021	0.024	0.021	0.021	0.015	0.009	0.013
en-zh	-0.041	-0.03	-0.034	-0.039	-0.035	-0.024	-0.026	-0.027	-0.026	-0.029	-0.026
en-hr	0.119	0.115	0.14	0.125	0.123	0.11	0.117	0.118	0.128	0.123	0.122
en-ja	0.097	0.098	0.101	0.091	0.085	0.082	0.081	0.082	0.079	0.08	0.075
en-liv	0.362	0.332	0.35	0.34	0.33	0.33	0.334	0.31	0.292	0.322	0.332
en-ru	0.262	0.227	0.234	0.248	0.23	0.199	0.188	0.227	0.223	0.244	0.234
en-uk	0.081	0.063	0.062	0.062	0.066	0.055	0.044	0.041	0.04	0.048	0.052
en-de	0.4	0.4	0.4	0.8	0.8	0.8	1.0	0.8	0.8	0.8	1.0
liv-en	0.247	0.276	0.311	0.303	0.302	0.302	0.286	0.298	0.291	0.306	0.314
zh-en	0.179	0.217	0.241	0.217	0.195	0.225	0.223	0.213	0.193	0.209	0.203
sah-ru	0.458	0.419	0.484	0.492	0.478	0.45	0.456	0.47	0.461	0.456	0.444
uk-cs	0.145	0.159	0.16	0.153	0.141	0.125	0.132	0.146	0.167	0.156	0.146
cs-uk	0.133	0.149	0.159	0.154	0.154	0.148	0.15	0.151	0.149	0.154	0.151

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.008	0.014	0.014	0.011	0.02	0.018	0.017	0.011	0.011	0.008	0.008
en-zh	-0.03	-0.028	-0.034	-0.033	-0.031	-0.033	-0.033	-0.028	-0.029	-0.027	-0.027
en-hr	0.118	0.11	0.119	0.116	0.109	0.101	0.106	0.118	0.127	0.129	0.128
en-ja	0.081	0.083	0.082	0.083	0.081	0.083	0.084	0.075	0.072	0.074	0.074
en-liv	0.344	0.344	0.346	0.34	0.328	0.328	0.346	0.34	0.34	0.344	0.348
en-ru	0.22	0.202	0.174	0.188	0.16	0.167	0.167	0.202	0.192	0.195	0.206
en-uk	0.049	0.047	0.048	0.048	0.044	0.049	0.042	0.036	0.049	0.038	0.034
en-de	1.0	0.8	0.8	0.8	0.6	0.8	1.0	0.8	1.0	1.0	1.0
liv-en	0.313	0.286	0.261	0.26	0.249	0.258	0.269	0.28	0.272	0.277	0.286
zh-en	0.225	0.219	0.195	0.191	0.179	0.185	0.179	0.183	0.211	0.213	0.209
sah-ru	0.441	0.436	0.45	0.439	0.43	0.427	0.447	0.416	0.408	0.399	0.408
uk-cs	0.144	0.15	0.155	0.153	0.149	0.159	0.158	0.156	0.143	0.137	0.137
cs-uk	0.148	0.152	0.154	0.15	0.14	0.14	0.142	0.141	0.139	0.141	0.141

Appendix B

Publications

B.1 Tokengram_F, a Fast and Accurate Token-based chrF++ Derivative

Tokengram_F, a fast and accurate token-based chrF++ derivative

Sören Dréano

ML-Labs
Dublin City University
soren.dreano2@mail.dcu.ie

Derek Molloy

School of Electronic Engineering
Dublin City University
derek.molloy@dcu.ie

Noel Murphy

School of Electronic Engineering
Dublin City University
noel.murphy@dcu.ie

Abstract

The tokengram_F metric presented in this paper is a novel approach to evaluating machine translation that has been submitted as part of the WMT23 challenge. It offers a new perspective on evaluating machine translation that takes advantage of modern tokenization algorithms to provide a more natural representation of the language in comparison to word n -grams.

Tokengram_F is an F-score-based evaluation metric for Machine Translation that is heavily inspired by chrF++ and can act as a more accurate replacement. By replacing word n -grams with n -grams obtained from tokenization algorithms, tokengram_F captures similarities between words sharing the same semantic roots.

While requiring minimal training based on an open corpus of monolingual datasets, the tokengram_F metric proposed still retains excellent performance that is comparable to more computationally expensive metrics. The tokengram_F metric demonstrates its versatility by showing satisfactory results, even when a tokenizer for a specific language is not available. In such cases, the tokenizer of a related language can be used instead, highlighting the adaptability of the tokengram_F metric to less commonly-used languages.

evaluation metrics that can effectively measure the quality of machine-generated translations.

Popular metrics in MT include BLEU (Papineni et al., 2002), which measures the overlap between sequences of words in the reference and generated texts; chrF (Popović, 2015), which is an F1 score at the character-level; chrF++ (Popović, 2017), which extends chrF with word n -grams; TER (Snover et al., 2006), which counts the erroneously-aligned words between the reference and the generation, and METEOR (Banerjee and Lavie, 2005), which takes into account synonyms and stems.

More recent metrics rely on neural network architectures, such as COMET (Rei et al., 2020) and MS-COMET (Kocmi et al., 2022). By using the similarity of vector representations of the generated translation and the reference translation, they provide state-of-the-art machine evaluation of translations at the cost of being expensive to train and compute.

Since its third instance in 2006, the Workshop on Statistical Machine Translation (WMT) has released an evaluation task to compare metrics each year. Generated translations are usually ranked by humans, and the correlation coefficient between the human-performed ranking and the evaluation metric-performed ranking determines the quality of the metric.

1 Introduction

Machine Translation (MT) is a subdomain of Neural Language Processing (NLP) that is focused on the translation of one natural language to another, with the aim of producing natural-sounding sentences. To evaluate the quality of algorithm-generated translations, evaluation metrics provide quantitative scores to objectively assess the accuracy of the model. Machine-generated translations are compared to human-generated translations in different ways depending on the evaluation metric. In recent years, machine translation has seen a great deal of progress in terms of accuracy and fluency. However, there is still a need for more robust

2 Tokengram_F

2.1 Tokenization

As text cannot be directly processed by machine learning algorithms, it first has to be converted into a numerical representation. Tokenization splits the text into smaller character sequences, including but not limited to phonemes, syllables, letters, words or base pairs, collectively named tokens. The tokenization process also often consists of adding special tokens, such as the unknown <unk> token to represent never-seen characters or the padding

730

<pad> token to pad the sentence to a fixed length. Each token can be converted to and from a unique identifier, which is usually an integer between 0 and the maximum vocabulary size minus one.

2.2 chrF++

An n -gram refers to a consecutive series of n tokens that are extracted from a given corpus of text or speech, with these units of text being defined based on the particular context of the application. A character-gram, or unigram, is a token that contains exactly one character, while a word-gram contains an entire word. chrF++ is an F-score using both word-grams and character-grams to compare the generated translation to the reference translation. The general formula is

$$ngrF\beta = (1 + \beta^2) \frac{(ngrP \times ngrF)}{(\beta^2 \times ngrP + ngrR)}$$

where β determines the weight of the recall as discussed in Section 3.4.3.

2.3 Modern tokenization algorithms

Subword-based tokenization divides words depending on their number of occurrences in the training data. Subwords can be combined to represent less frequent words or even words that were not present in the training data. For instance, in cases where a word such as “decaying” is absent from the vocabulary, an English tokenizer may represent it by combining the “decay” and “ing” tokens.

Byte Pair Encoding (BPE) (Sennrich et al., 2016), had been used for data compression long before it was ever applied in NLP-related tasks. After first counting the frequency of each unique word in the training data, BPE merges frequent occurrences of subword pairs until it reaches the desired vocabulary size.

Instead of starting from a small vocabulary representing the set of unique words, and growing in size from there (as in BPE), Unigram (Kudo, 2018) initialises its base vocabulary to a large number of symbols and then trims it down to the desired size. It is analogous to factor analysis, as at each step it calculates the loss of information that would be induced by removing each token, and then erases the less important ones from its vocabulary.

The sentence “The kingly sovereign governs” becomes:

```
words: "The" "kingly" "sovereign" "governs"
tokens: "The" "king" "ly" "sovereign" "govern" "s"
```

2.4 Replacing word-grams

When using word-grams for scoring, each word is compared regardless of its proximity to other words. For example, the words “say” and “saying” share a common root but this link would be lost when using word-grams.

In this work, the authors claim that modern tokenization algorithms can be used instead of word-grams to split the text in a more natural manner that reflects the structure of each language. Tokengram_F is an evaluation metric derived from chrF++ that replaces the use of word-grams by tokens learned either by Unigram or by BPE.

3 Methodology

3.1 Framework

SentencePiece (Kudo and Richardson, 2018) is a fast data-driven text tokenizer and detokenizer implementing the Unigram algorithm. The vocabulary size (number of individual tokens) needs to be provided before training. The minimum vocabulary size would consist of the number of special tokens and individual characters of the alphabet for each language. A large vocabulary size might lead to overfitting and a reduction in the effectiveness of the model, given that some parameters will be dedicated to rare words.

3.2 Training

The tokengram_F score uses the same 3-letter ISO-639-2 language code as the Tatoeba dataset, while the WMT tasks rely on the 2-letter ISO-639-1 language code. The website of the Library of Congress (Library of Congress, 2017) was used for conversions between the two norms.

The Tatoeba Translation Challenge (Tiedemann, 2020) is an initiative that aims to evaluate the effectiveness of MT systems on a large, diverse, and high-quality parallel corpus. While the main training data relies on OPUS (Tiedemann, 2012), which provides open-source sentence-aligned text corpora to support data-driven NLP, Tatoeba also provides monolingual datasets extracted from CirrusSearch Wikimedia dumps (Foundation, 2023).

Out of the 279 different languages available, 240 had a sufficiently large corpus to be included in the work described in this paper.

3.3 Exceptions

As the tokengram_F metric is dependent on the utilization of the Tatoeba monolingual datasets for tokenizer training, adaptations were necessary to accommodate languages that are not represented within this dataset.

3.3.1 Livonian:

While there is no dataset in the Tatoeba Translation Challenge to train a Livonian tokenizer, the Latvian tokenizer produced satisfactory results and was utilised as a substitute, highlighting the versatility of the tokengram_F metric and its ability to accommodate languages that are less frequently used.

3.3.2 Serbian and Indonesian:

The Tatoeba challenge does not offer monolingual datasets for neither Serbian nor Indonesian. Nevertheless, the Tatoeba Wikimedia data, which are appropriate for tokenizer training and available for both the Serbian and Indonesian languages, were employed as a substitute.

3.4 Tokengram_F parameters

3.4.1 Tokenization algorithm:

While Tokengram_F can be used with any tokenization algorithm, this study examined both BPE and Unigram.

3.4.2 n -gram length:

This parameter determines the number of items in the reference that will be compared with each item in the source sentence, to assess the degree of correspondence of the two sentences.

Previous work (Popović, 2015, 2017) has indicated that for chrF++ there is no necessity to set the maximum word n -gram length beyond $N=6$.

3.4.3 Beta:

In this metric, the relative importance of precision and recall in the evaluation metric is determined by the β parameter. When beta is equal to 1.0, precision and recall have equal importance, while when beta is equal to 3.0, recall is three times more significant than precision. Previous research (Popović, 2015) has evaluated two beta values, 1.0 and 3.0, with the latter being considered “the most promising variant” due to the higher correlations it obtained.

3.4.4 Vocabulary size:

The goal of the present study is to mitigate the effect of infrequent words on the accuracy of the tokengram_F metric. To investigate the influence of vocabulary size on the performance of the metric, three tokenizers were trained for each language using vocabulary sizes of 16,000, 32,000, and 50,000 tokens. As the average vocabulary size tends to decrease from one year to the next (Libovický, 2021), wider vocabularies have not been examined.

3.5 Optimal parameters

3.5.1 Finetuning

The optimal parameters were determined based on achieving the highest average correlation among segments or systems across three datasets: WMT20 (Mathur et al., 2020), WMT21 (Freitag et al., 2021), and WMT22 (Freitag et al., 2022).

Initially, the n -gram length was assessed at values of 3, 6, and 9, while maintaining a vocabulary size of 50,000. Consistent with the findings of the original paper, a n -gram length of 6 demonstrated the strongest correlation as shown in Table 1, and thus it was chosen for subsequent evaluations.

Subsequently, the beta values of 2, 3, and 4 were examined specifically for an n -gram length N of 6. The best overall correlation is obtained with Unigram and $\beta=3.0$.

Table 3 presents the results obtained with a vocabulary size of 32,000. As with the previous vocabulary size, the choice of the tokenization algorithm only slightly affects the results. A β of 3.0 or 4.0 seems to give the best results.

As shown in Table 2, the vocabulary size of 16,000, which was the smallest size examined, exhibits generally weaker correlations compared to larger sizes, thus precluding the exploration of smaller sizes.

3.5.2 Results

Despite the marginal disparity, when the results are not rounded, the optimal parameters for tokengram_F are found to be a vocabulary size of 50,000, unigram tokenization, an n -gram length $N=6$, and a β value of 3.0.

3.6 Source code

The source code of tokengram_F is available at https://github.com/SorenDreano/tokengram_F.

4 Conclusion

Instead of using word n -grams as a basis for comparing a generated translation with a reference translation, tokengram_F utilises contemporary tokenization algorithms to accomplish this task. As a result, words that share common roots are deemed similar, regardless of whether they are exact matches or not.

The results obtained from evaluating the tokengram_F metric on the WMT20, WMT21, and WMT22 datasets indicate that the use of tokens generated through the SentencePiece framework leads to improved performance compared to the use of traditional word-grams in the chrF++ metric. Full results are displayed in Appendix A. The tokengram_F metric is a simple and efficient method for obtaining a reasonable correlation with human rankings, with the added benefit of requiring minimal training time to be applied to new languages.

In the segment-level task of the WMT22 edition, tokengram_F managed to obtain better overall correlations than any other metric that could provide results for all language pairs, including ones that require extensive neural networks to operate. With the exception of two tasks, tokengram_F outperformed both chrF and chrF++ metrics.

In conclusion, the tokengram_F metric is presented as a promising alternative for evaluating the quality of machine translations, as it offers a simple and efficient solution with above-average performance compared to other models. The findings of this study provide strong evidence of the potential of the tokengram_F metric as a valuable evaluation tool for machine translation. Its combination of simplicity, efficiency, and adaptability make it an attractive alternative to existing metrics and a promising direction for future research in the field.

5 Further work

The optimal number of tokens in a tokenizer may vary depending on the language. Subsequent research could concentrate on determining the most suitable vocabulary size per language.

The majority of the tokenizers were trained using the MonoLanguage Datasets from the Tatoeba Challenge, which are based on data from the Wikimedia Foundation. It remains possible that alternate data sources may produce varying results.

6 Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

- Satanjeev Banerjee and Alon Lavie. 2005. [METEOR: An automatic metric for MT evaluation with improved correlation with human judgments](#). In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.
- Wikimedia Foundation. 2023. [Wikimedia downloads](#).
- Markus Freitag, Ricardo Rei, Nitika Mathur, Chi kiu Lo, Craig Stewart, Eleftherios Avramidis, Tom Kocmi, George Foster, Alon Lavie, and André Martins. 2022. [Results of wmt22 metrics shared task: Stop using bleu - neural metrics are better and more robust](#). In *Proceedings of the Seventh Conference on Machine Translation*, pages 46–68, Abu Dhabi.
- Markus Freitag, Ricardo Rei, Nitika Mathur, Chi-kiu Lo, Craig Stewart, George Foster, Alon Lavie, and Ondřej Bojar. 2021. [Results of the WMT21 metrics shared task: Evaluating metrics with expert-based human evaluations on TED and news domain](#). In *Proceedings of the Sixth Conference on Machine Translation*, pages 733–774, Online. Association for Computational Linguistics.
- Tom Kocmi, Hitokazu Matsushita, and Christian Federmann. 2022. [MS-COMET: More and better human judgements improve metric performance](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 541–548, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.
- Taku Kudo. 2018. [Subword regularization: Improving neural network translation models with multiple subword candidates](#). *CoRR*, abs/1804.10959.
- Taku Kudo and John Richardson. 2018. [SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.
- Jindřich Libovický. 2021. [Jindřich's blog – machine translation weekly 86: The wisdom of the wmt crowd](#). Online, Accessed: 07.02.2023.
- Library of Congress. 2017. [Codes for the representation of names of languages](#).
- Nitika Mathur, Johnny Wei, Markus Freitag, Qingsong Ma, and Ondřej Bojar. 2020. [Results of the WMT20 metrics shared task](#). In *Proceedings of the Fifth Conference on Machine Translation*, pages 688–725, Online. Association for Computational Linguistics.

- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. [Bleu: a method for automatic evaluation of machine translation](#). In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.
- Maja Popović. 2015. [chrF: character n-gram F-score for automatic MT evaluation](#). In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.
- Maja Popović. 2017. [chrF++: words helping character n-grams](#). In *Proceedings of the Second Conference on Machine Translation*, pages 612–618, Copenhagen, Denmark. Association for Computational Linguistics.
- Ricardo Rei, Craig Stewart, Ana C. Farinha, and Alon Lavie. 2020. [COMET: A neural framework for MT evaluation](#). *CoRR*, abs/2009.09025.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. [Neural machine translation of rare words with subword units](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.
- Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micculla, and John Makhoul. 2006. [A study of translation edit rate with targeted human annotation](#). In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.
- Jörg Tiedemann. 2012. [Parallel data, tools and interfaces in OPUS](#). In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).
- Jörg Tiedemann. 2020. [The tatoeba translation challenge - realistic data sets for low resource and multilingual MT](#). *CoRR*, abs/2010.06354.

A Appendix. Tables

Table 1: Correlations over all metrics depending of the hyperparameters with a vocabulary size of 50,000

Vocabulary	50000									
Algorithm	BPE					Unigram				
Beta	4	3			2	4	3			2
<i>n</i> -gram	6	9	6	3	6	6	9	6	3	6
System WMT20	0.875	0.871	0.876	0.871	0.877	0.876	0.871	0.876	0.871	0.877
System WMT21	0.715	0.715	0.716	0.713	0.717	0.716	0.715	0.717	0.714	0.718
System WMT22	0.837	0.831	0.834	0.834	0.829	0.836	0.831	0.835	0.834	0.830
Segment WMT20	0.277	0.274	0.277	0.279	0.277	0.276	0.273	0.277	0.278	0.277
Segment WMT21	0.158	0.155	0.158	0.166	0.157	0.160	0.159	0.158	0.170	0.152
Segment WMT22	0.398	0.393	0.398	0.399	0.400	0.398	0.393	0.399	0.397	0.400
Average	0.543	0.540	0.544	0.543	0.543	0.544	0.540	0.544	0.543	0.542

Table 2: Correlations over all metrics depending of the hyperparameters with a vocabulary size of 16,000

Vocabulary size	16000					
Tokenization algorithm	BPE			Unigram		
Beta	4	3	2	4	3	2
<i>n</i> -gram length	6					
System WMT20	0.875	0.876	0.877	0.876	0.877	0.877
System WMT21	0.715	0.716	0.717	0.716	0.717	0.718
System WMT22	0.839	0.837	0.83	0.839	0.837	0.83
Segment WMT20	0.278	0.278	0.277	0.277	0.277	0.277
Segment WMT21	0.152	0.153	0.157	0.146	0.15	0.153
Segment WM22	0.398	0.399	0.4	0.398	0.4	0.402
Average	0.543	0.543	0.543	0.542	0.543	0.543

Table 3: Correlations over all metrics depending of the hyperparameters with a vocabulary size of 32,000

Vocabulary size	32000					
Tokenization algorithm	BPE			Unigram		
Beta	4	3	2	4	3	2
n -gram length	6					
System WMT20	0.875	0.876	0.877	0.876	0.876	0.877
System WMT21	0.715	0.716	0.717	0.716	0.717	0.718
System WMT22	0.837	0.835	0.83	0.837	0.835	0.83
Segment WMT20	0.277	0.277	0.277	0.277	0.277	0.277
Segment WMT21	0.157	0.158	0.157	0.158	0.158	0.153
Segment WM22	0.397	0.399	0.400	0.399	0.401	0.402
Average	0.543	0.544	0.543	0.544	0.544	0.543

Table 4: Tokengram_F results on the WMT20 dataset compared to chrF++

Language pair	tokengram_F system r	chrF++ system r	tokengram_F segment τ	chrF++ segment τ
en-cs	0.865	0.833	0.485	0.478
en-de	0.961	0.958	0.371	0.367
en-ru	0.981	0.952	0.162	0.156
en-ta	0.941	0.956	0.590	0.579
en-zh	0.851	0.983	0.403	0.388
en-ja	0.949	0.328	0.521	0.506
en-pl	0.958	0.315	0.256	0.255
en-iu	0.433	0.338	0.340	0.338
cs-en	0.872	0.844	0.095	0.09
de-en	0.997	0.998	0.440	0.435
pl-en	0.508	0.970	0.032	0.034
ta-en	0.957	0.522	0.184	0.186
km-en	0.984	0.965	0.281	0.275
ps-en	0.894	0.964	0.143	0.145
ja-en	0.972	0.763	0.251	0.245
ru-en	0.921	0.977	0.055	0.054
zh-en	0.960	0.841	0.130	0.130
iu-en	0.765	0.726	0.242	0.246

Table 5: Tokengram_F results on the WMT21 dataset compared to chrF (chrF was used in place of chrF++ as chrF++ results were not reported)

Language pair	tokengram_F system r	chrF system r	tokengram_F segment τ	chrF segment τ
en-cs	0.978	0.970	0.549	0.531
en-zh	0.625	0.549	0.121	0.092
en-ha	0.760	0.748	0.185	0.186
en-ja	0.967	0.966	0.384	0.371
en-ru	0.756	0.943	0.214	0.201
en-de	0.842	0.831	0.448	0.098
cs-en	0.562	0.562	-0.052	-0.053
zh-en	0.269	0.723	0.395	-0.035
ha-en	0.921	0.924	0.021	0.021
ja-en	0.823	0.831	0.006	0.005
ru-en	0.579	0.593	-0.123	-0.126
de-en	0.424	0.357	-0.151	-0.162
fr-de	0.655	0.646	0.049	0.054
de-fr	0.504	0.498	0.111	0.110
bn-hi	0.949	0.941	0.079	0.071
hi-bn	0.877	0.872	0.335	0.327
xh-zu	0.999	0.998	0.306	0.301
zu-xh	0.997	0.999	0.529	0.530

Table 6: Tokengram_F results on the WMT22 dataset compared to chrF++ (chrF was used in place of chrF++ as chrF++ results were not reported)

Language pair	tokengram_F system r	chrF system r	tokengram_F segment τ	chrF segment τ
en-cs	0.602	0.689	0.077	0.147
en-zh	0.248	0.210	-0.044	0.051
en-hr	0.899	0.920	0.274	0.185
en-ja	0.927	0.931	0.241	0.142
en-liv	0.989	0.988	0.370	0.101
en-ru	0.852	0.813	0.659	0.153
en-uk	0.869	0.895	0.178	0.177
en-de	0.799	0.811	1.000	0.085
liv-en	0.985	0.969	0.500	0.184
zh-en	0.787	0.881	0.415	0.071
sah-ruh	1.000	1.000	0.856	0.430
uk-cs	0.971	0.979	0.350	0.171
cs-uk	0.921	0.927	0.311	0.195

B.2 Embed_Llama: Using LLM Embeddings for the Metrics Shared Task

Embed_Llama: using LLM embeddings for the Metrics Shared Task

Sören Dréano

ML-Labs
Dublin City University
soren.dreano2@mail.dcu.ie

Derek Molloy

School of Electronic Engineering
Dublin City University
derek.molloy@dcu.ie

Noel Murphy

School of Electronic Engineering
Dublin City University
noel.murphy@dcu.ie

Abstract

Embed_Llama is an assessment metric for language translation that hinges upon the utilization of the recently introduced Llama 2 Large Language Model (LLM), specifically focusing on its embedding layer, to transform sentences into a vector space that establishes connections between geometric and semantic proximities.

Investigations utilizing previous WMT datasets have revealed that within the Llama 2 architecture, relying solely on the initial embedding layer does not result in the highest degree of correlation when assessing machine translations. The incorporation of additional layers, however, holds the potential to augment the contextual understanding of sentences.

As a contribution to the WMT23 challenge, this study delves into the advantages derived from employing a pre-trained LLM that has not undergone fine-tuning specifically for translation evaluation tasks, to provide a metric conducive to operation on readily accessible consumer-grade hardware. This research digs into the observation that deeper layers within the model do not result in a linear increase in the spatial proximity between sentences within the vector space.

1 Introduction

The assessment of algorithm-generated translations entails the utilization of evaluation metrics that furnish quantitative scores to objectively gauge the precision of the model's output. Various methodologies are employed to juxtapose machine-generated translations with their human-generated counterparts, contingent upon the specific evaluation metric employed. In recent years, the realm of machine translation (MT) has witnessed notable advancements in terms of both translation accuracy and linguistic fluency.

Over time, there has been a significant enhancement in the correlation coefficient between human

assessments and the automated evaluation of sentences generated by machines. Earlier metrics, such as BLEU (Papineni et al., 2002) or chrF++ (Popović, 2017), predominantly relied on the textual overlap between reference translations and the machine-generated counterparts. In contrast, contemporary approaches, exemplified by COMET (Rei et al., 2020), harness recent breakthroughs in Natural Language Processing (NLP) and transformer models, enabling them to consider not only individual words, but also to leverage contextual semantics for a more comprehensive evaluation.

In the domain of Machine Learning (ML) applied to NLP, the embedding layer assumes a pivotal role within neural network architectures, particularly in tasks centered on textual data. Its central objective lies in the transformation of discrete tokens, encompassing entities like words or characters into continuous vector representations. These vector representations, which maintain continuity, are amenable to acquisition and manipulation by neural networks and are commonly referred to as *word embeddings*.

2 Embed_Llama

The initial component in a Natural Language Processing (NLP) model is typically an embedding layer, which serves the purpose of converting the distinct identifiers of tokens within the input sentence into a vectorized representation. In this context, it is essential to emphasize that sentences conveying similar semantic content should exhibit proximity in the vector space, irrespective of the presence of word-level overlap, in contrast to sentences chosen randomly.

Embed_Llama draws inspiration from Word2vec (Mikolov et al., 2013) using Llama 2 (Touvron et al., 2023), a contemporary open-source pre-trained model. Rather than needing to train an extra NLP model for the purpose of assessing translation quality, a viable alternative approach involves uti-

738

lizing a pre-trained, extensive neural network like Llama 2, which has been originally trained for next-token prediction. This approach allows for the investigation of how closely related sentences evolve across the model’s various layers, all without incurring the supplementary expenses associated with fine-tuning or training anew.

2.1 Word2vec

Word2vec is a methodology that utilizes a neural network model to extract word associations from comprehensive textual corpora. Post training, this model possesses the capability to identify synonymous terms and offer word suggestions for unfinished sentences. As the terminology implies, Word2vec symbolizes individual words by employing distinct numerical vectors, systematically engineered to encapsulate both the semantic and syntactic characteristics inherent to the words.

Embed_Llama leverages vectorial space to estimate similarity or dissimilarity. This estimation is accomplished by computing the cosine distance between two sentences.

2.2 Vocabulary size

The lexical repertoire of Llama 2 encompasses 32,000 unique tokens, a figure lower than that of both GPT-2 (Radford et al., 2019) and GPT-NeoX (Black et al., 2022), which both employ 50,000 unique tokens. Regrettably, the specific vocabulary sizes pertaining to GPT-3 (Brown et al., 2020) and GPT-4 (OpenAI, 2023) remain undisclosed.

2.3 Embeddings

The dimensionality of the embedding space represents a hyperparameter subject to adjustment. Embeddings of higher dimensions have the capacity to capture more intricate relationships; however, it is noteworthy that such higher-dimensional embeddings may necessitate increased quantities of data and computational resources for their effective utilization.

2.4 Cosine distance

The cosine similarity metric quantifies the similarity between two n-dimensional vectors by computing the cosine of the angle between them. This scoring measure finds common application in the domain of text mining (Singhal, 2001). The general formula for two vectors A and B is:

$$\cos \theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

To enable efficient computation on consumer-grade hardware, the two sentences slated for comparison are padded to match the maximum token count of the longer sentence in the pair. Consequently, a sentence initially shaped as [length] transforms into the shape [length × 4,096] following its processing through the embedding layer.

3 Hyperparameter

As the objective of this current study revolves around the utilization of a pre-trained network, our sphere of influence is limited to a sole hyperparameter, namely, the number of blocks to retain before computing the cosine distance.

3.1 Block architecture

As shown in Figure 1, each block, denoted as the *LlamaDecoderLayer*, is structured with several components, including an attention layer, two normalization layers, and a multi-layer perceptron. The multi-layer perceptron, in turn, consists of three linear layers along with an associated activation function, while the attention layer also includes a rotary embedding layer.

The Llama 2 model, comprising 7 billion parameters, encompasses an embedding layer, 32 blocks, and a projection layer. To determine the optimal number of blocks, datasets extracted from the WMT challenge editions of 2020, 2021, and 2022 were employed. Due to the limited GPU memory allocation in the current project, it was only feasible to investigate the Llama 2 model up to a depth of 22 blocks, whereas the model has a total of 32 available blocks.

```
LlamaDecoderLayer(
  (self_attn): LlamaAttention(
    (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (mlp): LlamaMLP(
    (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
    (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
    (act_fn): SiLUActivation()
  )
  (input_layernorm): LlamaRMSNorm()
  (post_attention_layernorm): LlamaRMSNorm()
)
```

Figure 1: Architecture of the Llama2 model as displayed in the Huggingface library

3.2 Exploring the depth

It was initially hypothesized that increasing the number of blocks would improve the contextual

representation of sentence meaning. Figure 2 reveals that, in contrast to our initial hypotheses, the augmentation of block quantities does not significantly modify the correlation between the systems and the ground truth, whether by augmentation or reduction. Furthermore, it is noteworthy that this correlation exhibits variability across different datasets. Specifically, the Pearson correlation between the number of layers and algorithm accuracy is 0.34 for the WMT20 dataset, but it decreases to -0.79 for the WMT22 dataset.

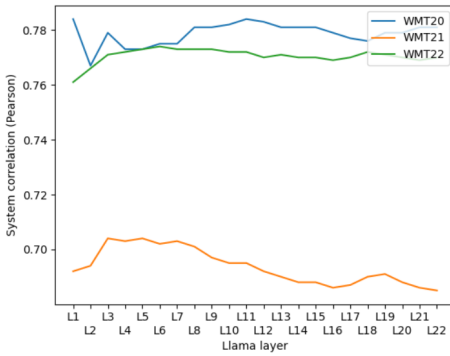


Figure 2: System correlations depending on the Llama layer

As shown in Figure 3, these observations hold with respect to segment correlation as well. The quantity of blocks employed in the Embed_Llama does not consistently enhance the metric’s quality, whether for individual segments or entire systems.

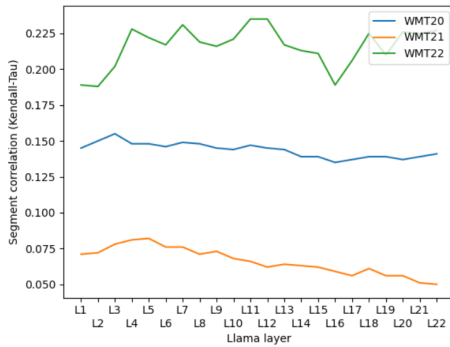


Figure 3: Segment correlations depending on the Llama layer

The highest levels of correlation between human rankings and Embed_Llama rankings were achieved by utilizing a mere two blocks following

the embedding layer, resulting in optimal overall performance across the WMT20, WMT21, and WMT22 datasets. This not only expedited the computational process, but also decreased the GPU memory demands for metric computation.

3.3 Inter-languages variations

As depicted in Figure 4, the associations between metric quality and language pairs exhibit large variations. For instance, when considering the Hungarian-to-English language pair, the Pearson coefficient registers at 0.83, whereas it falls to -0.85 for the Czech-to-Ukrainian pair.

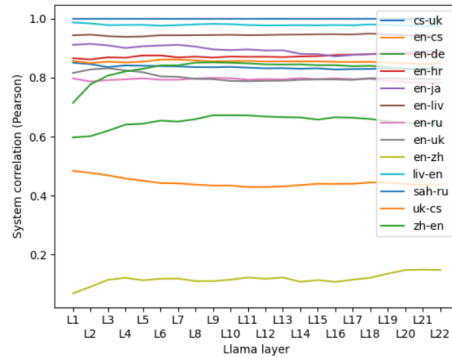


Figure 4: System correlations for each language pair in the WMT2022 dataset depending on the Llama layer

3.4 Intra-languages variations

Considerable variability is observed even within the same language pair across various datasets. For instance, the English-to-Chinese language pair is encompassed within the WMT2020, WMT2021, and WMT2022 datasets. However, as illustrated in Figure 5, no discernible correlations emerge between the number of utilized blocks and the quality of Embed_Llama scores. This is evident in the transformation of the Pearson coefficient, which shifts from -0.28 in the WMT2021 to 0.73 in the WMT2022 dataset.

3.5 Inter-datasets variations

Table 1 presents the average mean values and their corresponding standard deviations, showcasing the relationship between metric accuracy and the number of utilized blocks for language pairs common to all three datasets. It is notable that, apart for the English-to-Japanese pair, there exists a significant degree of variability in the performance of the

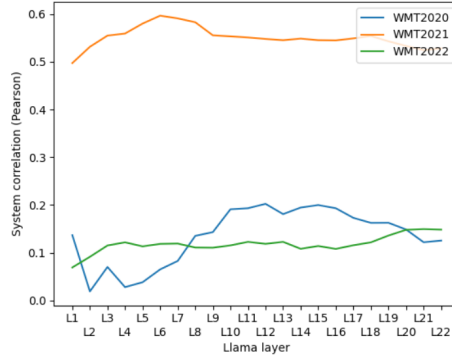


Figure 5: System correlations of the English-to-Chinese language pair depending on the Llama layer and the dataset

Language pair	Mean	Standard deviation
en-cs	-0.34	0.22
en-de	-0.02	0.4
en-ja	-0.89	0.04
en-ru	-0.02	0.54
en-zh	0.35	0.45
zh-en	-0.17	0.46

Table 1: Means and standard deviations of the system correlations depending on the Llama layer when WMT2020, WMT2021 and WMT2022 are merged

same language pairs across different datasets. This variability is underscored by the substantial standard deviations in relation to the absolute values of the means. Given the limited usage of just three datasets, it is essential to acknowledge that the relatively small sample size may hinder the ability to draw conclusive inferences regarding inter-dataset variability.

3.6 Full results

Tables 2, 4 and 6, correspondingly, present the Pearson correlation coefficients for datasets WMT2020, WMT2021 and WMT2022, illustrating the association between the algorithm-assigned scores and the actual rankings of the evaluated systems for individual language pairs.

With regard to segment-level correlations, they are presented in Tables 3, 5 and 7 for WMT2020, WMT2021, and WMT2022, respectively. It is noteworthy that, in contrast to system correlations, these are represented by Kendall coefficients, which are utilized as a measure of ordinal associa-

tion.

It is noteworthy that the observed variations in these correlations are predominantly influenced by the specific language pairs, rather than the depth of the final block employed prior to cosine similarity computation, aligning with our anticipated outcome.

3.7 Source code

The source code of Embed_Llama is available at https://github.com/SorenDreano/embed_llama.

4 Conclusion

Although the authors initially anticipated that Embed_Llama would exhibit suboptimal performance for a majority of language pairs, except for those involving English, due to the apparent constraints posed by a limited vocabulary size and the nature of the dataset Llama 2 was trained on, the actual performance did not exhibit a significant underperformance.

The results from previous iterations of the WMT metrics shared task, presented in Appendix A, indicate that this approach may not meet the contemporary state-of-the-art standards exemplified by METRICX_XXL (unpublished) and COMET-22 (Rei et al., 2022).

The methodology involving the utilization of a non-finetuned, pre-trained Large Language Model (LLM) to assess translation quality through vector space similarity comparisons remains a prospective avenue of inquiry. This prospect gains relevance in light of forthcoming open-source models characterized by expansive vocabularies and training data encompassing diverse languages.

5 Further work

Given the recent proliferation of open-source LLMs, it is likely that another model, either presently or in the near future, may surpass the performance of Llama 2 for translation evaluation without necessitating any fine-tuning.

In the current investigation, the exploration has been confined to the 7 billion parameters model. It remains conceivable that employing a more extensive model with increased parameters may yield a more precise metric, albeit at the trade-off of heightened computational resource demands.

In the present evaluation, an exploration was limited to the initial 22 blocks. Subsequent endeavors

may consider augmenting this number, as doing so could potentially result in further benefits.

Moreover, it is worth noting that the optimal selection of the number of blocks to employ may be contingent upon the specific target language. Consequently, adjusting this hyperparameter based on the language in question could potentially yield enhanced correlation results.

In the scope of the current academic study, solely the cosine distance served as the chosen similarity measure for tensors. Future research endeavors may wish to investigate alternative distance metrics, such as the Euclidean distance or the Manhattan distance, for potential exploration.

6 Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

- Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. [Gpt-neox-20b: An open-source autoregressive language model](#).
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners](#).
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. [Efficient estimation of word representations in vector space](#).
- OpenAI. 2023. [Gpt-4 technical report](#).
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. [Bleu: a method for automatic evaluation of machine translation](#). In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.
- Maja Popović. 2017. [chrF++: words helping character n-grams](#). In *Proceedings of the Second Conference on Machine Translation*, pages 612–618, Copenhagen, Denmark. Association for Computational Linguistics.
- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. [Language models are unsupervised multitask learners](#).
- Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. 2022. [COMET-22: Unbabel-IST 2022 submission for the metrics shared task](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.
- Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. [Unbabel’s participation in the WMT20 metrics shared task](#). In *Proceedings of the Fifth Conference on Machine Translation*, pages 911–920, Online. Association for Computational Linguistics.
- Amit Singhal. 2001. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24:35–43.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. [Llama 2: Open foundation and fine-tuned chat models](#).

A Appendix. Tables

Table 2: Pearson correlations (r) for the WMT20 system dataset depending on the Llama layer. The findings pertaining to the second layer are shown in bold, as it represents the prevailing default layer count within the Embed_Llama

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.855	0.866	0.875	0.877	0.873	0.876	0.872	0.863	0.861	0.864	0.867
en-de	0.923	0.928	0.933	0.929	0.928	0.927	0.927	0.926	0.925	0.925	0.928
en-ru	0.856	0.677	0.769	0.772	0.81	0.829	0.821	0.902	0.912	0.899	0.918
en-ta	0.883	0.901	0.914	0.921	0.926	0.932	0.934	0.936	0.939	0.942	0.945
en-zh	0.137	0.019	0.07	0.028	0.038	0.065	0.083	0.135	0.143	0.191	0.193
en-ja	0.898	0.892	0.879	0.871	0.88	0.881	0.874	0.883	0.88	0.876	0.871
en-pl	0.88	0.874	0.88	0.878	0.875	0.864	0.868	0.873	0.873	0.872	0.868
en-iu	0.252	0.226	0.194	0.156	0.137	0.121	0.119	0.106	0.105	0.096	0.094
cs-en	0.795	0.777	0.752	0.771	0.789	0.788	0.796	0.796	0.796	0.789	0.775
de-en	0.992	0.996	0.996	0.994	0.99	0.99	0.99	0.988	0.988	0.988	0.99
pl-en	0.419	0.403	0.428	0.433	0.401	0.395	0.4	0.394	0.388	0.398	0.407
ta-en	0.876	0.889	0.918	0.922	0.919	0.923	0.921	0.919	0.917	0.919	0.918
km-en	0.954	0.969	0.988	0.988	0.988	0.989	0.99	0.989	0.987	0.986	0.985
ps-en	0.926	0.874	0.877	0.862	0.872	0.877	0.873	0.875	0.875	0.875	0.878
ja-en	0.913	0.938	0.947	0.946	0.946	0.948	0.949	0.948	0.946	0.948	0.949
ru-en	0.949	0.939	0.949	0.948	0.953	0.947	0.948	0.949	0.948	0.944	0.939
zh-en	0.97	0.967	0.963	0.956	0.957	0.955	0.955	0.954	0.954	0.951	0.951
iu-en	0.64	0.676	0.68	0.662	0.638	0.645	0.638	0.625	0.616	0.615	0.633

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.865	0.866	0.86	0.858	0.851	0.856	0.855	0.857	0.866	0.874	0.874
en-de	0.925	0.926	0.922	0.921	0.917	0.918	0.918	0.921	0.923	0.929	0.929
en-ru	0.922	0.904	0.903	0.898	0.902	0.885	0.862	0.902	0.93	0.941	0.938
en-ta	0.942	0.942	0.944	0.946	0.947	0.947	0.948	0.948	0.948	0.946	0.946
en-zh	0.202	0.181	0.195	0.2	0.193	0.173	0.162	0.163	0.149	0.122	0.125
en-ja	0.87	0.867	0.869	0.865	0.865	0.857	0.858	0.848	0.838	0.836	0.836
en-pl	0.873	0.871	0.875	0.872	0.869	0.86	0.865	0.862	0.858	0.857	0.857
en-iu	0.084	0.085	0.082	0.084	0.083	0.078	0.073	0.082	0.091	0.092	0.086
cs-en	0.781	0.787	0.787	0.794	0.802	0.794	0.8	0.79	0.783	0.776	0.785
de-en	0.989	0.989	0.988	0.988	0.986	0.989	0.989	0.991	0.992	0.994	0.994
pl-en	0.408	0.42	0.421	0.418	0.417	0.414	0.413	0.41	0.406	0.415	0.415
ta-en	0.918	0.917	0.914	0.91	0.907	0.912	0.912	0.913	0.913	0.919	0.921
km-en	0.983	0.982	0.975	0.975	0.967	0.964	0.967	0.968	0.97	0.966	0.968
ps-en	0.876	0.881	0.887	0.892	0.891	0.888	0.889	0.889	0.889	0.895	0.897
ja-en	0.947	0.946	0.944	0.941	0.936	0.94	0.942	0.943	0.942	0.941	0.941
ru-en	0.938	0.938	0.938	0.939	0.94	0.935	0.935	0.935	0.933	0.935	0.937
zh-en	0.95	0.951	0.951	0.95	0.952	0.953	0.951	0.954	0.955	0.958	0.958
iu-en	0.619	0.613	0.606	0.6	0.602	0.627	0.63	0.639	0.644	0.662	0.66

Table 3: Kendall correlations (τ) for the WMT20 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.226	0.231	0.244	0.234	0.232	0.226	0.229	0.225	0.225	0.222	0.226
en-de	0.181	0.19	0.207	0.201	0.205	0.198	0.202	0.201	0.199	0.196	0.198
en-ru	0.028	0.034	0.037	0.045	0.038	0.045	0.042	0.044	0.043	0.046	0.05
en-ta	0.355	0.354	0.323	0.284	0.282	0.267	0.279	0.281	0.282	0.26	0.255
en-zh	0.147	0.141	0.164	0.144	0.138	0.141	0.147	0.152	0.152	0.16	0.165
en-ja	0.277	0.281	0.295	0.29	0.285	0.287	0.283	0.285	0.284	0.284	0.282
en-pl	0.097	0.095	0.103	0.101	0.101	0.102	0.098	0.101	0.098	0.094	0.102
en-iu	0.218	0.224	0.205	0.187	0.18	0.173	0.185	0.187	0.186	0.179	0.18
cs-en	0.064	0.065	0.073	0.072	0.077	0.076	0.076	0.073	0.074	0.078	0.083
de-en	0.349	0.374	0.387	0.384	0.39	0.389	0.385	0.376	0.372	0.375	0.381
pl-en	-0.016	-0.016	-0.0	-0.007	-0.0	0.0	-0.003	0.002	-0.002	-0.005	-0.006
ta-en	0.108	0.118	0.125	0.121	0.122	0.119	0.128	0.121	0.106	0.108	0.123
km-en	0.141	0.145	0.144	0.12	0.13	0.122	0.117	0.114	0.097	0.108	0.114
ps-en	0.088	0.081	0.074	0.076	0.058	0.053	0.06	0.061	0.065	0.061	0.08
ja-en	0.109	0.136	0.136	0.144	0.147	0.149	0.151	0.144	0.137	0.139	0.134
ru-en	0.011	0.02	0.019	0.022	0.026	0.022	0.031	0.023	0.02	0.02	0.017
zh-en	0.065	0.067	0.071	0.072	0.073	0.072	0.072	0.071	0.068	0.069	0.069
iu-en	0.16	0.161	0.175	0.178	0.187	0.191	0.195	0.194	0.195	0.198	0.199

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.23	0.229	0.222	0.223	0.215	0.218	0.221	0.222	0.221	0.222	0.221
en-de	0.19	0.193	0.187	0.183	0.177	0.178	0.184	0.184	0.186	0.186	0.19
en-ru	0.043	0.037	0.043	0.042	0.045	0.036	0.039	0.037	0.039	0.043	0.044
en-ta	0.281	0.29	0.283	0.266	0.261	0.267	0.264	0.232	0.204	0.224	0.237
en-zh	0.166	0.168	0.162	0.165	0.16	0.165	0.166	0.163	0.165	0.166	0.164
en-ja	0.282	0.276	0.28	0.281	0.274	0.272	0.271	0.271	0.263	0.266	0.27
en-pl	0.098	0.096	0.091	0.087	0.093	0.093	0.095	0.093	0.09	0.092	0.092
en-iu	0.17	0.167	0.162	0.163	0.154	0.153	0.149	0.146	0.141	0.145	0.143
cs-en	0.081	0.073	0.073	0.076	0.074	0.079	0.081	0.081	0.077	0.073	0.075
de-en	0.377	0.377	0.371	0.372	0.361	0.37	0.37	0.378	0.374	0.383	0.386
pl-en	-0.004	0.0	-0.001	0.005	0.001	-0.001	0.001	-0.004	-0.006	-0.004	-0.003
ta-en	0.118	0.111	0.107	0.101	0.103	0.11	0.112	0.114	0.112	0.106	0.103
km-en	0.109	0.11	0.092	0.097	0.083	0.091	0.1	0.111	0.116	0.124	0.126
ps-en	0.074	0.081	0.067	0.068	0.064	0.056	0.066	0.07	0.07	0.067	0.07
ja-en	0.128	0.125	0.116	0.112	0.102	0.115	0.117	0.125	0.126	0.129	0.129
ru-en	0.015	0.007	0.009	0.008	0.013	0.012	0.015	0.018	0.02	0.025	0.022
zh-en	0.069	0.072	0.07	0.069	0.066	0.069	0.07	0.072	0.07	0.074	0.074
iu-en	0.192	0.177	0.173	0.176	0.173	0.179	0.184	0.183	0.188	0.189	0.189

Table 4: Pearson correlations (r) for the WMT21 system dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.985	0.984	0.985	0.982	0.981	0.981	0.982	0.982	0.979	0.98	0.982
en-zh	0.497	0.531	0.555	0.559	0.58	0.597	0.591	0.583	0.555	0.553	0.551
en-ha	0.534	0.551	0.567	0.567	0.567	0.553	0.554	0.553	0.563	0.56	0.547
en-ja	0.82	0.83	0.838	0.836	0.83	0.84	0.837	0.839	0.829	0.822	0.816
en-ru	0.567	0.621	0.682	0.674	0.674	0.658	0.674	0.614	0.572	0.576	0.573
en-de	0.818	0.793	0.804	0.801	0.794	0.789	0.794	0.799	0.796	0.792	0.79
cs-en	0.542	0.454	0.435	0.428	0.429	0.432	0.426	0.426	0.429	0.42	0.423
zh-en	0.232	0.186	0.155	0.159	0.172	0.179	0.188	0.196	0.198	0.196	0.192
ha-en	0.825	0.855	0.858	0.855	0.867	0.868	0.867	0.867	0.865	0.86	0.855
ja-en	0.728	0.726	0.748	0.753	0.761	0.76	0.759	0.759	0.761	0.761	0.759
ru-en	0.613	0.606	0.535	0.514	0.5	0.496	0.502	0.506	0.506	0.504	0.509
de-en	0.171	0.22	0.237	0.238	0.221	0.23	0.223	0.223	0.225	0.234	0.243
fr-de	0.564	0.556	0.555	0.556	0.556	0.555	0.557	0.555	0.556	0.554	0.552
de-fr	0.477	0.511	0.578	0.579	0.579	0.58	0.579	0.575	0.563	0.564	0.574
bn-hi	0.908	0.943	0.94	0.942	0.937	0.929	0.941	0.95	0.943	0.941	0.942
hi-bn	0.879	0.872	0.913	0.93	0.925	0.915	0.912	0.907	0.911	0.908	0.915
xh-zu	0.952	0.932	0.934	0.931	0.923	0.909	0.904	0.905	0.898	0.899	0.891
zu-xh	0.95	0.937	0.97	0.973	0.975	0.972	0.971	0.972	0.976	0.976	0.976

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.983	0.984	0.983	0.983	0.979	0.981	0.982	0.983	0.983	0.984	0.983
en-zh	0.548	0.545	0.548	0.545	0.545	0.549	0.554	0.543	0.533	0.524	0.528
en-ha	0.541	0.543	0.554	0.553	0.571	0.562	0.572	0.567	0.556	0.542	0.534
en-ja	0.81	0.804	0.795	0.793	0.781	0.783	0.786	0.777	0.772	0.774	0.779
en-ru	0.581	0.582	0.562	0.58	0.526	0.557	0.565	0.603	0.621	0.625	0.637
en-de	0.796	0.803	0.808	0.808	0.814	0.81	0.81	0.8	0.79	0.787	0.791
cs-en	0.394	0.386	0.383	0.387	0.388	0.395	0.41	0.425	0.422	0.433	0.435
zh-en	0.189	0.189	0.185	0.189	0.19	0.173	0.174	0.168	0.157	0.142	0.146
ha-en	0.846	0.842	0.84	0.841	0.841	0.827	0.828	0.829	0.823	0.818	0.814
ja-en	0.76	0.763	0.768	0.771	0.775	0.772	0.772	0.77	0.769	0.762	0.763
ru-en	0.51	0.512	0.513	0.52	0.521	0.52	0.529	0.534	0.535	0.532	0.528
de-en	0.23	0.217	0.212	0.196	0.184	0.207	0.21	0.205	0.203	0.215	0.214
fr-de	0.553	0.554	0.555	0.555	0.553	0.55	0.55	0.547	0.543	0.541	0.542
de-fr	0.566	0.564	0.558	0.558	0.556	0.567	0.564	0.561	0.563	0.574	0.577
bn-hi	0.935	0.932	0.931	0.931	0.93	0.933	0.933	0.938	0.94	0.93	0.922
hi-bn	0.904	0.892	0.876	0.86	0.869	0.863	0.859	0.865	0.864	0.852	0.829
xh-zu	0.9	0.903	0.906	0.903	0.909	0.905	0.908	0.912	0.903	0.898	0.896
zu-xh	0.977	0.979	0.981	0.981	0.981	0.981	0.977	0.977	0.976	0.969	0.962

Table 5: Kendall correlations (τ) for the WMT21 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.197	0.198	0.243	0.243	0.238	0.223	0.221	0.219	0.219	0.214	0.22
en-zh	0.028	0.028	0.048	0.042	0.042	0.044	0.039	0.041	0.039	0.037	0.037
en-ha	0.073	0.077	0.087	0.081	0.077	0.071	0.071	0.068	0.062	0.061	0.062
en-ja	0.181	0.184	0.204	0.21	0.214	0.204	0.197	0.196	0.204	0.196	0.192
en-ru	0.094	0.101	0.099	0.099	0.089	0.09	0.091	0.086	0.081	0.087	0.084
en-de	0.241	0.31	0.172	0.172	0.241	0.31	0.31	0.241	0.241	0.241	0.241
cs-en	-0.042	-0.048	-0.049	-0.045	-0.041	-0.044	-0.042	-0.05	-0.046	-0.045	-0.051
zh-en	0.319	0.286	0.319	0.384	0.395	0.319	0.308	0.319	0.384	0.297	0.276
ha-en	-0.022	-0.016	-0.007	-0.003	-0.003	-0.009	-0.008	-0.007	-0.011	-0.014	-0.011
ja-en	-0.028	-0.021	-0.019	-0.018	-0.015	-0.017	-0.014	-0.015	-0.014	-0.012	-0.015
ru-en	-0.121	-0.124	-0.12	-0.12	-0.117	-0.123	-0.12	-0.119	-0.121	-0.124	-0.122
de-en	-0.155	-0.158	-0.151	-0.148	-0.153	-0.157	-0.155	-0.156	-0.158	-0.155	-0.152
fr-de	0.057	0.058	0.055	0.052	0.044	0.055	0.062	0.057	0.053	0.047	0.051
de-fr	0.054	0.066	0.082	0.079	0.075	0.051	0.057	0.054	0.049	0.055	0.055
bn-hi	-0.006	0.007	0.016	0.026	0.026	0.024	0.024	0.022	0.015	0.023	0.014
hi-bn	0.132	0.119	0.126	0.135	0.139	0.137	0.145	0.15	0.147	0.146	0.14
xh-zu	0.128	0.129	0.139	0.13	0.118	0.104	0.101	0.101	0.09	0.082	0.065
zu-xh	0.169	0.139	0.181	0.181	0.16	0.129	0.122	0.116	0.12	0.118	0.109

Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.213	0.218	0.213	0.22	0.198	0.209	0.209	0.196	0.193	0.194	0.197
en-zh	0.042	0.045	0.048	0.048	0.043	0.043	0.043	0.046	0.041	0.04	0.04
en-ha	0.058	0.06	0.061	0.061	0.061	0.06	0.062	0.064	0.068	0.067	0.067
en-ja	0.189	0.187	0.196	0.193	0.202	0.196	0.195	0.19	0.184	0.179	0.179
en-ru	0.088	0.089	0.109	0.107	0.116	0.116	0.109	0.106	0.108	0.104	0.105
en-de	0.172	0.172	0.172	0.172	0.034	0.034	0.034	0.034	0.034	0.034	0.034
cs-en	-0.047	-0.048	-0.045	-0.055	-0.054	-0.06	-0.056	-0.055	-0.053	-0.058	-0.058
zh-en	0.297	0.319	0.265	0.265	0.33	0.286	0.362	0.286	0.297	0.276	0.276
ha-en	-0.014	-0.013	-0.011	-0.014	-0.012	-0.015	-0.014	-0.014	-0.014	-0.014	-0.014
ja-en	-0.016	-0.02	-0.021	-0.02	-0.02	-0.022	-0.022	-0.021	-0.02	-0.021	-0.018
ru-en	-0.121	-0.121	-0.123	-0.123	-0.123	-0.126	-0.124	-0.116	-0.116	-0.12	-0.121
de-en	-0.152	-0.152	-0.152	-0.154	-0.159	-0.154	-0.155	-0.156	-0.155	-0.155	-0.157
fr-de	0.049	0.047	0.043	0.041	0.031	0.034	0.032	0.036	0.032	0.032	0.031
de-fr	0.039	0.04	0.035	0.039	0.049	0.046	0.053	0.046	0.046	0.046	0.052
bn-hi	0.015	0.02	0.022	0.022	0.019	0.02	0.023	0.023	0.027	0.025	0.025
hi-bn	0.123	0.114	0.114	0.113	0.114	0.12	0.12	0.116	0.108	0.091	0.074
xh-zu	0.076	0.083	0.081	0.094	0.094	0.088	0.098	0.089	0.085	0.083	0.07
zu-xh	0.118	0.122	0.14	0.131	0.141	0.138	0.141	0.144	0.137	0.125	0.122

Table 6: Pearson correlations (r) for the WMT22 system dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.484	0.477	0.469	0.458	0.451	0.443	0.442	0.437	0.434	0.434	0.429
en-zh	0.069	0.091	0.115	0.122	0.113	0.118	0.119	0.111	0.11	0.115	0.123
en-hr	0.866	0.862	0.869	0.868	0.875	0.875	0.869	0.871	0.868	0.871	0.87
en-ja	0.911	0.914	0.909	0.901	0.906	0.909	0.911	0.905	0.896	0.893	0.896
en-liv	0.943	0.946	0.941	0.938	0.94	0.943	0.943	0.944	0.945	0.945	0.944
en-ru	0.798	0.787	0.792	0.794	0.798	0.793	0.793	0.797	0.8	0.798	0.793
en-uk	0.816	0.828	0.832	0.825	0.818	0.804	0.803	0.796	0.795	0.789	0.788
en-de	0.598	0.602	0.62	0.641	0.644	0.654	0.651	0.659	0.673	0.672	0.672
liv-en	0.987	0.984	0.978	0.979	0.979	0.976	0.978	0.98	0.982	0.981	0.978
zh-en	0.715	0.777	0.807	0.821	0.83	0.841	0.842	0.851	0.852	0.85	0.848
sah-ru	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
uk-cs	0.857	0.849	0.854	0.851	0.854	0.861	0.861	0.858	0.855	0.856	0.857
cs-uk	0.851	0.846	0.836	0.841	0.84	0.839	0.838	0.836	0.835	0.836	0.834
Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.429	0.431	0.436	0.44	0.44	0.44	0.445	0.444	0.439	0.437	0.439
en-zh	0.119	0.123	0.108	0.114	0.108	0.116	0.122	0.136	0.148	0.149	0.148
en-hr	0.871	0.87	0.872	0.873	0.877	0.878	0.879	0.881	0.881	0.879	0.879
en-ja	0.892	0.893	0.881	0.879	0.872	0.877	0.88	0.884	0.885	0.889	0.891
en-liv	0.944	0.946	0.946	0.947	0.947	0.946	0.949	0.947	0.943	0.942	0.942
en-ru	0.795	0.794	0.798	0.795	0.798	0.795	0.796	0.789	0.785	0.782	0.782
en-uk	0.789	0.79	0.793	0.794	0.794	0.793	0.798	0.798	0.796	0.796	0.794
en-de	0.668	0.666	0.665	0.658	0.666	0.664	0.66	0.652	0.648	0.644	0.644
liv-en	0.977	0.977	0.978	0.977	0.978	0.977	0.98	0.978	0.977	0.975	0.975
zh-en	0.845	0.844	0.845	0.842	0.842	0.839	0.84	0.836	0.828	0.826	0.83
sah-ru	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
uk-cs	0.855	0.855	0.855	0.855	0.854	0.853	0.853	0.851	0.848	0.846	0.848
cs-uk	0.832	0.832	0.83	0.831	0.828	0.829	0.83	0.831	0.83	0.83	0.832

Table 7: Kendall correlations (τ) for the WMT22 segment dataset depending on the Llama layer

Language pair	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
en-cs	0.014	0.014	0.023	0.024	0.021	0.024	0.021	0.021	0.015	0.009	0.013
en-zh	-0.041	-0.03	-0.034	-0.039	-0.035	-0.024	-0.026	-0.027	-0.026	-0.029	-0.026
en-hr	0.119	0.115	0.14	0.125	0.123	0.11	0.117	0.118	0.128	0.123	0.122
en-ja	0.097	0.098	0.101	0.091	0.085	0.082	0.081	0.082	0.079	0.08	0.075
en-liv	0.362	0.332	0.35	0.34	0.33	0.33	0.334	0.31	0.292	0.322	0.332
en-ru	0.262	0.227	0.234	0.248	0.23	0.199	0.188	0.227	0.223	0.244	0.234
en-uk	0.081	0.063	0.062	0.062	0.066	0.055	0.044	0.041	0.04	0.048	0.052
en-de	0.4	0.4	0.4	0.8	0.8	0.8	1.0	0.8	0.8	0.8	1.0
liv-en	0.247	0.276	0.311	0.303	0.302	0.302	0.286	0.298	0.291	0.306	0.314
zh-en	0.179	0.217	0.241	0.217	0.195	0.225	0.223	0.213	0.193	0.209	0.203
sah-ru	0.458	0.419	0.484	0.492	0.478	0.45	0.456	0.47	0.461	0.456	0.444
uk-cs	0.145	0.159	0.16	0.153	0.141	0.125	0.132	0.146	0.167	0.156	0.146
cs-uk	0.133	0.149	0.159	0.154	0.154	0.148	0.15	0.151	0.149	0.154	0.151
Language pair	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22
en-cs	0.008	0.014	0.014	0.011	0.02	0.018	0.017	0.011	0.011	0.008	0.008
en-zh	-0.03	-0.028	-0.034	-0.033	-0.031	-0.033	-0.033	-0.028	-0.029	-0.027	-0.027
en-hr	0.118	0.11	0.119	0.116	0.109	0.101	0.106	0.118	0.127	0.129	0.128
en-ja	0.081	0.083	0.082	0.083	0.081	0.083	0.084	0.075	0.072	0.074	0.074
en-liv	0.344	0.344	0.346	0.34	0.328	0.328	0.346	0.34	0.34	0.344	0.348
en-ru	0.22	0.202	0.174	0.188	0.16	0.167	0.167	0.202	0.192	0.195	0.206
en-uk	0.049	0.047	0.048	0.048	0.044	0.049	0.042	0.036	0.049	0.038	0.034
en-de	1.0	0.8	0.8	0.8	0.6	0.8	1.0	0.8	1.0	1.0	1.0
liv-en	0.313	0.286	0.261	0.26	0.249	0.258	0.269	0.28	0.272	0.277	0.286
zh-en	0.225	0.219	0.195	0.191	0.179	0.185	0.179	0.183	0.211	0.213	0.209
sah-ru	0.441	0.436	0.45	0.439	0.43	0.427	0.447	0.416	0.408	0.399	0.408
uk-cs	0.144	0.15	0.155	0.153	0.149	0.159	0.158	0.156	0.143	0.137	0.137
cs-uk	0.148	0.152	0.154	0.15	0.14	0.14	0.142	0.141	0.139	0.141	0.141

B.3 CycleGN: A Cycle Consistent Approach for Neural Machine Translation

CycleGN: a Cycle Consistent approach for Neural Machine Translation

Sören Dréano

ML-Labs
Dublin City University
soren.dreano2@mail.dcu.ie

Derek Molloy

School of Electronic Engineering
Dublin City University
derek.molloy@dcu.ie

Noel Murphy

School of Electronic Engineering
Dublin City University
noel.murphy@dcu.ie

Abstract

CycleGN is a Neural Machine Translation framework relying on the Transformer architecture. Its approach is similar to a Discriminator-less CycleGAN, specifically tailored for non-parallel text datasets.

The foundational concept of our research posits that in an ideal scenario, retro-translations of generated translations should revert to the original source sentences. Consequently, a pair of models can be trained using a Cycle Consistency Loss only, with one model translating in one direction and the second model in the opposite direction.

As a contribution to the WMT24 challenge, this study explores the efficacy of the CycleGN architectural framework in learning translation tasks across two language pairs, English-Chinese and German-English, under two distinct non-parallel dataset conditions: permuted and non-intersecting. Our findings demonstrate the robust adaptability of CycleGN in learning translation tasks, irrespective of the language pair.

the performance levels of human professionals in certain contexts.

Neural Machine Translation (NMT) datasets necessitate substantial text corpora, structured as aligned pairs. This alignment implies the requirement for sentences with equivalent meaning to be present in a minimum of two distinct languages, enabling the initiation of model training to forge linguistic linkages. Ongoing initiatives, including OPUS (Tiedemann and Thottingal, 2020) and Tatoeba (Tiedemann, 2012), are committed to facilitating public access to these datasets. Parallel datasets comprise a small subset of the volume of data in monolingual datasets.

Despite the widespread availability of large parallel corpora for numerous language pairs, the capacity to employ solely monolingual datasets would substantially expand the pool of training data. This approach is particularly beneficial for languages with scarce parallel text corpora.

Regardless of the remarkable efficacy exhibited by Large Language Models (LLM) in NMT without the necessity of exclusive training on parallel data (Zhu et al., 2023), their considerable magnitude renders them costly in terms of both training and operation. This economic burden consequently restricts their widespread availability.

Back-translation (Sennrich et al., 2016) is a technique leveraging a trained MT (Machine Translation) model to translate sentences from a monolingual dataset to produce corresponding pairs, thereby synthetically augmenting the training data. Our research is founded on the premise that the process of translating a sentence from a source language to a target language, followed by its retro-translation from the target language back to the source language, allows for the measurement of the disparity between the original and the machine-retro-translated sentences. This disparity serves as a metric to assess the efficacy of the models and facilitates the backpropagation of gradients within

1 Introduction

The introduction of the Transformer architecture (Vaswani et al., 2017) marked a significant advancement in the field of Machine Translation, witnessing widespread adoption since its inception. Although self-attention mechanisms were not novel and had been investigated in prior studies (Bahdanau et al., 2016), the Transformer model demonstrated its formidable capabilities within Natural Language Processing (NLP). Characterized by its parallelized structure, the Transformer architecture facilitated computational efficiency, enabling the incorporation of a larger number of parameters. This enhancement has been exemplified in NLP systems like Charles University Block-Backtranslation-Improved Transformer Translation (cubt) (Popel et al., 2020), which have surpassed

the networks. Notably, this methodology has been previously implemented in the realm of Image-to-Image Translation, as evidenced in the renowned CycleGAN framework from [Zhu et al. \(2017\)](#).

2 Previous work

The TextCycleGAN model ([Lorandi et al., 2023](#)), while not utilizing the Transformer architecture nor operating within the MT field, introduced an innovative strategy for text style transfer. This approach employed a CycleGAN on the Yelp dataset to facilitate the learning of mappings between positive and negative textual styles, notably in the absence of paired examples.

[Shen et al. \(2017\)](#) exemplified the feasibility of training two encoder-decoder networks in an unsupervised manner that enables the sharing of a latent space, thereby permitting style transfer. [Lample et al. \(2018\)](#), adopting a similar technique within the MT context, substantiated that the use of parallel datasets is not a prerequisite for effective translation.

3 Definitions

Machine Translation models are most commonly trained using “parallel” datasets, which are structured collections of text pairs. Each pair comprises a segment of text in a source language and its translation in the target language. By providing direct translations, models learn correspondences between text units to map the source language to the target language.

A non-parallel dataset on the other hand does not consist in pairs of text segments, consequently the source and target sentences do not share any explicit correspondence. Such a dataset can be created by combining any two monolingual datasets of two distinct languages and adjusting for the number of samples. In the context of this research, two sub-categories of non-parallel datasets are introduced.

3.1 Permuted dataset

A “permuted” dataset is defined as a parallel dataset wherein the sentences of one language have been systematically rearranged. Consequently, this results in a non-parallel corpus where it is guaranteed that each sentence has a corresponding translation located at an unspecified index within the dataset. The authors postulate that when employing sufficiently large monolingual datasets, which are not

derived from permuted parallel corpora, it is likely that most sentences will possess an accurate translation “somewhere” within the dataset.

3.2 Non-intersecting dataset

A “non-intersecting” dataset is a non-parallel dataset for which it is guaranteed that no sentence has an exact translation. A non-intersecting dataset is derived from a meticulously curated parallel dataset devoid of duplicate entries. Two unique sets of natural integers are produced, each functioning as an index list of phrases to retain for each respective language.

4 Datasets

The datasets employed in this study are the English-German and Chinese-English language pairs from the WMT23 challenge ([Kocmi et al., 2023](#)). The data released for the WMT23 General MT task can be freely used for research purposes. Due to the current implementation’s high computational demands, the models were not trained for the entirety of an epoch. Specifically, only 10% of the English-German dataset was used, while about half of the Chinese-English dataset in the non-intersecting condition.

Type	English-German	Chinese-English
Permuted	27,801,496	27,801,496
Non-intersecting	27,801,496	17,676,442
Original dataset	295,805,439	35,452,884

Table 1: Number of sentences used during training depending on the dataset type

5 Training

For greater clarity, the mathematical notations from the original CycleGAN work will be employed in the present study. Given two languages $\mathcal{X}$ and $\mathcal{Y}$ with appropriate datasets, the objective is to obtain two NMT models $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$ and $\mathcal{F} : \mathcal{Y} \mapsto \mathcal{X}$ such that if the translations are perfect, $\mathcal{G}(\mathcal{F}(y)) = y$ and $\mathcal{F}(\mathcal{G}(x)) = x$, with $x \in \mathcal{X}$ and for $y \in \mathcal{Y}$.

By using the Cross-Entropy Loss (CEL) ([Zhang and Sabuncu, 2018](#)) in the role of the Cycle Consistency Loss (CCL), we can determine the distance between the original sentence and its double translation in order to compute the gradients.

As in the original CycleGAN work, our current study also implements an Identity Loss (IL), which also relies on the CEL, to help with the training stability. As $\mathcal{G}$ consists in a mapping $\mathcal{X} \mapsto \mathcal{Y}$, if

given an input $y \in \mathcal{Y}$, the input should remain unchanged such that $\mathcal{G}(y) = y$. The same loss is applied to $\mathcal{F}$ between $\mathcal{F}(x)$ and x , as displayed in Figure 1.

5.1 Model architecture

The architecture used for both models, $\mathcal{G}$ and $\mathcal{F}$, is the Marian framework (Junczys-Dowmunt et al., 2018) implemented by Huggingface’s Transformers library (Wolf et al., 2020), which is licensed under the Apache Licence. While most parameters follow the default configuration, Table 2 references the changes that were made in order to reduce the computational cost of the architecture.

Parameter	Huggingface	Current work
Vocabulary size	58,101	32,000
Encoder layers	12	6
Decoder layers	12	6
Encoder attention heads	16	8
Decoder attention heads	16	8
Encoder feed-forward	4096	2048
Decoder feed-forward	4096	2048
Position embeddings	1024	128
Activation function	GELU	ReLU

Table 2: Non-default parameters in the configuration of Marian Transformer models

5.2 Vocabulary organization

NMT models usually employ either a unified tokenizer or two distinct tokenizers. In the case of a single tokenizer, it is trained using sentences from both the source and target distributions, avoiding any duplicates. This approach facilitates the sharing of the encoder and decoder embedding layers, thereby diminishing computational demands and enhancing model accuracy (Press and Wolf, 2017).

Conversely, the alternative approach entails training one tokenizer on the source distribution and another one on the target distribution. While this method restricts the possibility of tying embeddings, it can potentially double the vocabulary size without increasing the dimensions of the embeddings. The overall vocabulary size of the model in this scenario, is the cumulative total of the two individual vocabularies, barring shared tokens like punctuation symbols.

While contemporary Transformer models like Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) and Generative Pre-trained Transformers (GPT) (Radford et al., 2018) typically utilize a single tokenizer, this study

introduces a novel vocabulary methodology that amalgamates the aforementioned approaches. This method involves training two tokenizers, each for a respective language and with half the vocabulary size. Subsequently, the identifiers of one tokenizer are adjusted to prevent overlap, yielding a result analogous to a single tokenizer that includes duplicates across languages. It is important to note that special tokens such as $\langle eos \rangle$ (End of Sentence) and $\langle pad \rangle$ (Padding) are shared and not duplicated. This strategy is designed to simplify model analysis during development, albeit at the expense of a reduced vocabulary.

5.3 Obtaining labels

In the training process of a Transformer model, it is imperative to have prior knowledge of the labels, as the decoder predicts tokens sequentially. Each token prediction, barring the initial one, is contingent upon all preceding predictions. By possessing prior knowledge of the reference translation, it becomes feasible to contrast each predicted token against the ground truth, enabling the calculation of the loss at every step.

Nevertheless, in the case of non-parallel datasets, the labels are by definition not known in advance. It is therefore not possible to calculate the loss after each predicted token. Furthermore, the act of selecting the most probable token for each prediction constitutes a non-differentiable operation, thus precluding the possibility of backpropagation once the sentence is fully generated.

Naturally, in inference mode, Transformers are able to generate sentences without labels. Thus, the first step is to generate the pseudo-labels $\hat{x}$ and $\hat{y}$, where $\hat{x}$ is used as the label of y and $\hat{y}$ as the label of x . Even though this step cannot be used to compute the gradients, it is crucial for the entire process.

$\hat{x}$ is computed from $\mathcal{F}(\hat{y})$ with x as the label, and $\hat{y}$ is computed from $\mathcal{G}(\hat{x})$ with y as the label. The CCL is applied between $\hat{x}$ and x , and between $\hat{y}$ and y to compute the gradients and update the weights of $\mathcal{G}$ and $\mathcal{F}$.

5.4 A Discriminator-less GAN

The CycleGAN methodology, as indicated by its nomenclature, is predicated on the Generative Adversarial Network (GAN) framework, initially introduced in Goodfellow et al. (2014). This paradigm involves the training of a Generator model in conjunction with another model, termed

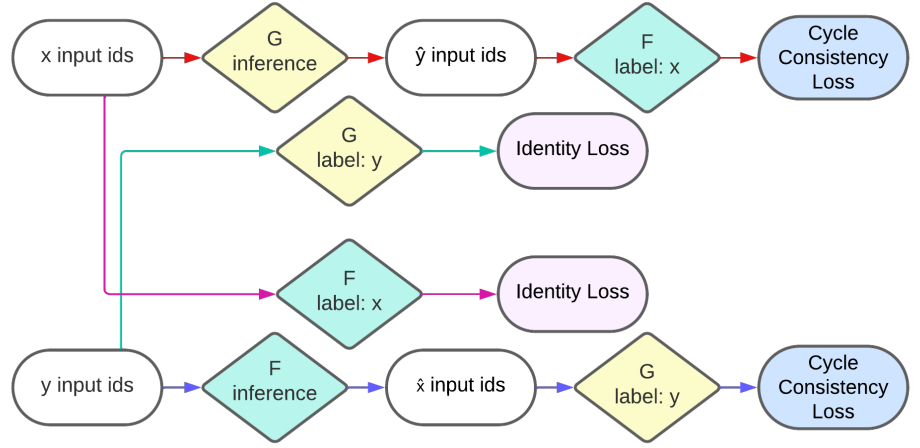


Figure 1: CycleGN training process

the Discriminator. The Discriminator is specifically trained to distinguish between authentic samples drawn from the dataset and synthetic samples produced by the Generator. In the CycleGAN training process, the Discriminators intervene after the generation of $\hat{x}$ and $\hat{y}$, helping the training of the Generators. However, as mentioned in Section 5.3, there can be no gradient computation during the generation of $\hat{x}$ and $\hat{y}$ in a Transformer and as such, Discriminators cannot be used in the present work. This is why CycleGN is not an “Adversarial” approach, hence the name.

6 Pre-training

During the development of CycleGN, a critical issue became clear, which prevented the model’s ability to converge and learn effectively. As described in Section 5.3, the first step of the CycleGN framework is to generate $\hat{x}$ and $\hat{y}$. During the first initialisations, these pseudo-labels will be generated randomly and will depend only on the initialization of the weights of $\mathcal{G}$ and $\mathcal{F}$. However, the models consistently converge towards a trivial solution wherein by merely reproducing the input, they satisfy the loss function criteria without achieving any meaningful learning or transformation of the data.

6.1 Absence of intermediate evaluation

As there is no Discriminator to ensure that $\hat{x}$ belongs to $\mathcal{X}$ and $\hat{y}$ belongs to $\mathcal{Y}$, $\mathcal{G}$ and $\mathcal{F}$ converge towards $x = \hat{y} = \hat{x}$ and $y = \hat{x} = \hat{y}$, as this approach achieves an optimal outcome on the CCL function, registering a value of zero, as schematised in Figure 2.

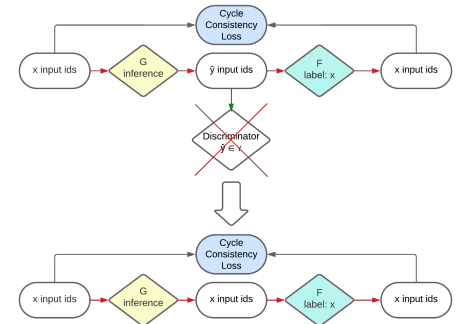


Figure 2: In the absence of a Discriminator $y \in \mathcal{Y}$ and pre-training is not employed, the CycleGN architecture will converge towards a state where no translation happens and still perfectly satisfy the CCL function

6.2 Moving away from the easiest path

Masked Language Modeling (MLM) is a pre-training strategy implemented in BERT, where a specified proportion of the input tokens are sub-

stituted with a unique $\langle \text{mask} \rangle$ token. The objective of the neural network under this paradigm is to accurately reconstruct the original sentence from this degraded input. This process enables the model to discern intricate relationships between words and to develop a profound representation of the language. This pre-training has revealed excellent performances in diverse NLP application such as sentiment analysis (Alaparthi and Mishra, 2021), text classification (Sun et al., 2020), Named Entity Recognition (NER) (Souza et al., 2020) (Chang et al., 2021) (Akhtyamova, 2020) and paraphrase detection (Khairova et al., 2022).

As MLM does not require any labels, as the labels are generated from the dataset, it is perfectly adapted to the CycleGN approach. A single model $\mathcal{H}$ is trained on the non-parallel dataset to reconstruct both languages, with 15% of the input tokens masked. This model $\mathcal{H}$ has the exact same architecture as $\mathcal{G}$ and $\mathcal{F}$. When training the CycleGN, rather than randomly initializing $\mathcal{G}$ and $\mathcal{F}$, the parameters from $\mathcal{H}$ are directly transferred to both $\mathcal{G}$ and $\mathcal{F}$. Indeed, as $\mathcal{H}$ learns to reconstruct both language $\mathcal{X}$ and $\mathcal{Y}$, it can be used to initialize both networks. Figure 3 shows the training process of $\mathcal{H}$.

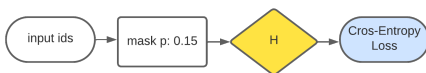


Figure 3: Masked Language Modeling training process

7 Training stability

It is crucial for the CycleGN framework that the two models exhibit approximately equivalent levels of performance. Given the interdependent nature of these models, where the output of one serves as the input for the other, maintaining consistency between them during training is imperative. Without a strategy in place to prevent the performance of the models from diverging, it is possible for one model to gain the “upper hand” over the other.

7.1 Divergence between the Generators

Figure 4 presents the evolution of the CCL of an early prototype of CycleGN and it can clearly be seen that one of the two generators, $\mathcal{F}$, ends up performing much better than its counterpart $\mathcal{G}$, which blocks any future training.

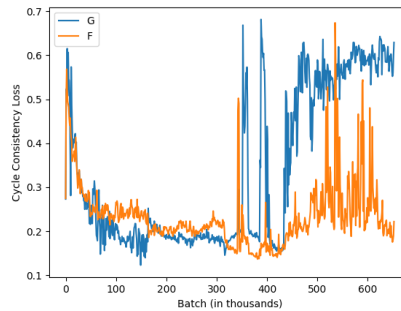


Figure 4: Evolution of the Cross-Entropy Loss during the training of an early prototype on the permuted German-English dataset

7.2 Gradient Clipping

Gradient clipping is a technique utilized in the training of Deep Learning (DL) models, to address the problem of “exploding” gradients. This issue occurs when gradients escalate to excessively high values during training, leading to numerical instability and impeding the model’s convergence to an optimal solution.

Gradient clipping can be implemented through two primary methods: norm clipping and value clipping. Norm clipping involves establishing a threshold on the overall magnitude of the gradient vector. On the other hand, value clipping involves individually adjusting elements of the gradient vector that exceed the specified threshold.

By clipping the gradients by norm, with a threshold of 1.0, as advised by the Huggingface library, the training stabilizes and the divergence between $\mathcal{G}$ and $\mathcal{F}$ disappears.

Figure 5 demonstrates how the addition of gradient clipping helps with training stability during the training of the permuted German-English model.

7.3 Batch size

The original CycleGAN research mentions using a batch size of 1, and while they did not state the reason in the research paper, one of the authors explained it in a GitHub issue (Junyanz, 2017) as a lack of GPU memory.

Rajput et al. (2021) examined the impact of batch size within the CycleGAN architecture, observing a significant decline in performance the more the batch size is increased. This deterioration was evident both through the example images presented in

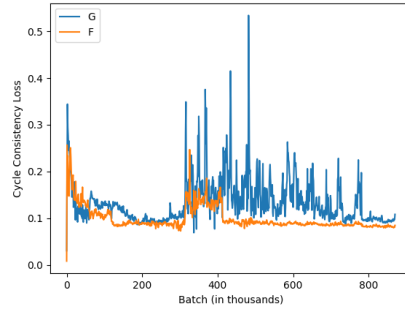


Figure 5: Evolution of the Cross-Entropy Loss during the training of the permuted German-English models

that study and through the calculated cosine dissimilarity, indicating inferior model performance with larger batch sizes. However, quality was achieved at the expense of computational efficiency, as the training duration to achieve 200 epochs was 8 hours with a batch size of 1, but this was reduced to just 2 hours with a batch size of 64.

In the context of this research, however, the trade-off between quality improvement and computing resource, as observed in the aforementioned study, does not hold true. Utilizing a batch size of 1 in the CycleGAN experiments hindered any form of convergence. Consequently, a batch size of 32 was selected, as it represents the maximum capacity that could be accommodated within the available 24GB of GPU memory of the NVidia 4090 used for this work.

7.4 One large epoch or multiple smaller ones?

The CycleGAN framework is recognized for its computational expense due to several inherent factors. Primarily, as CycleGAN operates on the principle of cycle consistency, it necessitates the training of two GANs simultaneously – one for each direction of the transformation. This structure requires substantial computational resources, as each GAN consists of both a Generator and a Discriminator.

The resource-intensiveness of the CycleGAN process, thus limits the size of the dataset that can be used in a reasonable time. This necessitated a decision between training for a single epoch on a large dataset, or training for multiple epochs on a smaller corpus arose.

The CycleGN framework was compared on the

permuted German-English dataset under four different conditions:

1. One epoch containing 1% of the dataset
2. Five epochs containing 0.2% of the dataset
3. One epoch containing 2% of the dataset
4. Five epochs containing 0.4% of the dataset

The Crosslingual Optimised Metric for Evaluation of Translation (COMET) score (Rei et al., 2020) was selected as our comparison criterion, as this metric has proven to be one of the most effective in recent WMT competitions, according to Koemi et al. (2022), due to its strong correlation with human judgment, aligning well with our goal of mirroring human evaluative standards. Multiple COMET models have been made available and the default “wmt22-comet-da” model was chosen. The average scores obtained on 10,000 test sentences that were not part of the model training set are presented in Table 3.

Condition	English->German	German->English
1	0.2727	0.2715
2	0.2411	0.2635
3	0.2741	0.2665
4	0.2258	0.2658

Table 3: COMET scores of CycleGN models depending on the permuted German-English dataset condition

Models exposed to a larger portion of the total dataset demonstrate superior performance compared to those limited to a smaller, repetitive subset, especially when the dataset encompasses over half a million to a million sentences. The authors extrapolate this result to larger datasets and thus chose to train the CycleGN models for a single epoch on the largest dataset possible.

8 Results

Even if tracking the CCL is an inexpensive manner to estimate the progress of the training of the CycleGN architecture, a low loss value can also hide an absence of translation, as mentioned in Section 6.1. This is why an evaluation metric such as COMET is crucial to assess the progression of the CycleGN framework.

8.1 Evolution of COMET score during training

To measure the performances of CycleGN, every 1,000th batch the CCL was averaged and 1,000 sentences from the test set were translated to compute the COMET score.

Figures 6, 7, 8 and 9 demonstrate that the actual quality of translation, as measured by the COMET metric, increases with time. Figures 6 through 9 illustrate a progressive enhancement in the translation quality over time, as quantified by the COMET metric. This enhancement is observed respectively in the permuted and non-intersecting German-English models (Figures 6 and 7), as well as in the permuted and non-intersecting English-Chinese models (Figures 8 and 9). Figures 6 and 7 exhibit a sudden drop in the increase of accuracy, which is acknowledged by the authors. This anomaly will be thoroughly examined and discussed in a subsequent academic study.

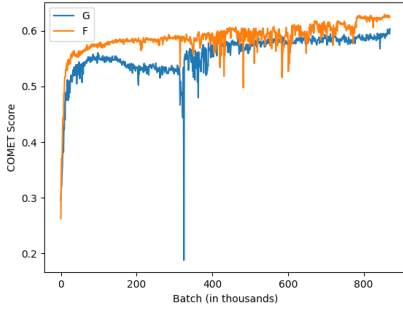


Figure 6: Evolution of the COMET score during the training of the permuted German-English models

8.2 COMET Scores post-training completion

After the end of the training, a test set of 10,000 sentences per language were translated and the COMET scores are displayed in Table 4. In order to give a point of comparison, architecture-matched models using the original parallel datasets were trained. As in the case of the CycleGN training, these parallel models were only trained for a single epoch on the exact same number of sentences as the permuted models were.

The authors expected the COMET score of the CycleGN to be inferior to architecture-matched models trained using parallel corpora, as information is by definition lost during the permutation of

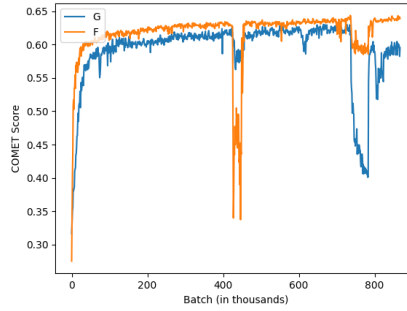


Figure 7: Evolution of the COMET score during the training of the non-intersecting German-English models

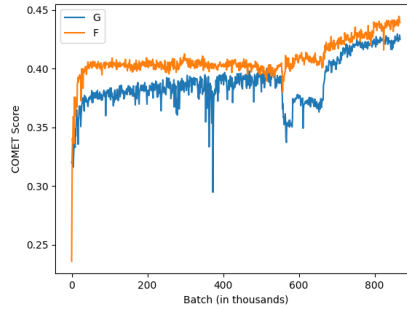


Figure 8: Evolution of the COMET score during the training of the permuted Chinese-English models

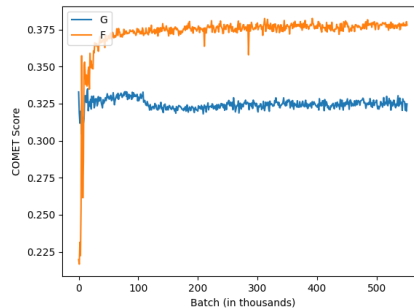


Figure 9: Evolution of the COMET score during the training of the non-intersecting Chinese-English models

the parallel datasets. However, the authors argue that the differences between the scores is likely smaller with larger datasets.

	English → German	German → English
Permuted	0.505	0.537
Non-intersecting	0.556	0.579
Parallel	0.780	0.775

Table 4: COMET score of the German-English models

	English → Chinese	Chinese → English
Permuted	0.425	0.537
Non-intersecting	0.382	0.448
Parallel	0.000	0.749

Table 5: COMET score of the Chinese-English models

9 Future Work

Further investigations will benefit from the incorporation of a more extensive dataset and an exploration of larger model architectures.

9.1 Target dataset

The current work has been trained on a small dataset compared to MT standards. Future work should try to see how convergence progresses with more iterations. Further computational optimizations are probably necessary to shorten the training time required.

9.2 Larger models

The current architecture relies on a total of 158,769,152 parameters, which is only about a third of the size of the default in the Huggingface library. Although Tables 4 and 5 demonstrate that the current number of parameters, when trained using a parallel dataset, is capable of producing better translations than when exposed to permuted and non-intersecting datasets, an increase in both the number of epochs and the size of the dataset should be prioritized, larger models being common in NMT.

10 Source Code

The source code of CycleGN is available at <https://github.com/SorenDreano/CycleGN>.

Limitations

The investigation acknowledges certain inherent limitations which may impact the generalizability and applicability of the findings.

Language diversity

Another issue that arises from the computing cost of CycleGN is the lack in language diversity. Indeed, our current work only used the English-

German and Chinese-English language pairs. Consequently, it cannot be certain that the approach presented can be applied to other languages and all alphabets. This is why CycleGN is taking part in WMT24, to explore the framework’s performance on a wide range of language pairs.

Training limitations

Since training a CycleGN model is particularly costly, there is a trade-off between training models on all language pairs, or choosing a subset of these pairs to train fewer models with more iterations and on a greater number of examples. In order to demonstrate the effectiveness of CycleGN on a wide range of language pairs, the first choice was made, i.e. to train models on all pairs, even if this means obtaining inferior results.

Unused models

Unlike the previous edition (Kocmi, 2023), where most language pairs were bidirectional, i.e. the evaluations were to and from, the 2024 General Translation task is unidirectional. This means that for each language pair, it is sufficient to train a model that translates from the source to the target.

This is not a change that is favourable to CycleGN, since it is a bidirectional training architecture. Indeed, its cyclical nature means that one model must be trained from one language to another, and another model must complete the cycle, i.e. from this second language to the first. In other words, half the time spent training CycleGN is spent training a model which only serves to train the first, but which will never be evaluated in the contest.

This change has been accompanied by an increase in the number of language pairs, from 6 bidirectional and 2 unidirectional in 2023 to 11 unidirectional in 2024.

Monolingual datasets

During the WMT challenge, teams are provided with monolingual datasets. Although this dataset format is perfectly suited to CycleGN training, they have been discarded for two reasons. The first is that for the majority of language pairs, the parallel datasets supplied have been truncated in order to reduce training time. The second is related to the construction of permuted and non-intersecting datasets, since it is preferable to build them from non-parallel datasets, as detailed in Section 3.

Reduced dataset sizes

The datasets were truncated to obtain a maximum of 27,801,496 sentences for training and 100,000 sentences for the development set. The final size of the datasets used and the number of epochs is shown in Table 6 for permuted models and Table 7 for non-intersecting models. While the permuted models have all been trained, this was not the case for the non-intersecting models, due to lack of time.

Training time

To make it possible to train so many models, several machines were used, with different technical characteristics, in particular different GPUs. However, by estimating the training time according to the number of sentences in the dataset and the GPU used, the total training time for all the models trained on the WMT24 datasets represents approximately 3,700 hours on an NVidia 4090.

Ethics Statement

This study, focusing on the training of NMT models using non-parallel datasets, adheres to the highest ethical standards in research. We recognize the critical importance of ethical considerations in computational linguistics and machine learning, especially as they pertain to data sourcing, model development, and potential impacts on various linguistic communities.

Our research utilizes publicly available, non-parallel linguistic datasets. We ensure that all data is sourced following legal and ethical guidelines, respecting intellectual property rights and privacy concerns.

In our commitment to scientific integrity, we maintain transparency in our research methodologies, model development, and findings. We aim to make our results reproducible and accessible to the scientific community, contributing positively to the field of machine translation.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

- Liliya Akhtyamova. 2020. [Named entity recognition in spanish biomedical literature: Short review and bert model](#). In *2020 26th Conference of Open Innovations Association (FRUCT)*, pages 1–7.
- Shivaji Alaparthi and Manit Mishra. 2021. [Bert: a sentiment analysis odyssey](#). *Journal of Marketing Analytics*, 9(2):118–126.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. [Neural machine translation by jointly learning to align and translate](#).
- Yuan Chang, Lei Kong, Kejia Jia, and Qinglei Meng. 2021. [Chinese named entity recognition method based on bert](#). In *2021 IEEE International Conference on Data Science and Computer Application (ICDSCA)*, pages 294–299.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [Bert: Pre-training of deep bidirectional transformers for language understanding](#).
- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. [Generative adversarial networks](#).
- Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. [Marian: Fast neural machine translation in C++](#). In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.
- Junyanz. 2017. [Question: Batch size · issue 27 · junyanz/pytorch-cyclegan-and-pix2pix](#).
- Nina Khairova, Anastasiia Shapovalova, Orken Mamyrbayev, Nataliia Sharonova, and Kuralay. 2022. Using bert model to identify sentences paraphrase in the news corpus. In *CEUR Workshop Proceedings, volume 3171*, pages 38–48.
- Tom Kocmi. 2023. [Shared task: General machine translation](#).
- Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, Barry Haddow, Philipp Koehn, Benjamin Marie, Christof Monz, Makoto Morishita, Kenton Murray, Makoto Nagata, Toshiaki Nakazawa, Martin Popel, Maja Popović, and Mariya Shmatova. 2023. [Findings of the 2023 conference on machine translation \(WMT23\): LLMs are here but not quite there yet](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 1–42, Singapore. Association for Computational Linguistics.

Language pair	Parallel sentences in WMT24 dataset	Sentences used	Number of epochs
Czech-Ukrainian	10,757,756	10,657,756	1
English-Chinese	55,216,751	27,801,496	1
English-Czech	56,288,239	27,801,496	1
English-German	295,805,439	27,801,496	1
English-Hindi	315,070	314,070	10
English-Icelandic	23,434,361	23,334,361	1
English-Japanese	33,875,119	27,801,496	1
English-Russian	75,961,169	27,801,496	1
English-Spanish	626,076,911	27,801,496	1
English-Ukrainian	16,062,359	15,962,359	1
Japanese-Chinese	22,642,571	22,542,571	1

Table 6: Comparison between the number of sentences available in the WMT24 dataset and the number of sentences used to train the permuted models depending on the language pair

Language pair	Parallel sentences in WMT24 dataset	Sentences used	Number of epochs
English-Chinese	55,216,751	17,676,442	1
English-Czech	56,288,239	27,801,496	1
English-German	295,805,439	27,801,496	1
English-Russian	75,961,169	27,801,496	1

Table 7: Comparison between the number of sentences available in the WMT24 dataset and the number of sentences used to train the non-intersecting models depending on the language pair

- Tom Kocmi, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Thammie Gowda, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Rebecca Knowles, Philipp Koehn, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Michal Novák, Martin Popel, and Maja Popović. 2022. [Findings of the 2022 conference on machine translation \(WMT22\)](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.
- Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc’Aurelio Ranzato. 2018. [Unsupervised machine translation using monolingual corpora only](#).
- Michela Lorandi, Maram A.Mohamed, and Kevin McGuinness. 2023. [Adapting the CycleGAN Architecture for Text Style Transfer](#). *Irish Machine Vision and Image Processing Conference*.
- Martin Popel, Marketa Tomkova, Jakub Tomek, Łukasz Kaiser, Jakob Uszkoreit, Ondřej Bojar, and Zdeněk Žabokrtský. 2020. [Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals](#). *Nature Communications*, 11(4381):1–15.
- Ofir Press and Lior Wolf. 2017. [Using the output embedding to improve language models](#).
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. *OpenAI*.
- Pranjal Singh Rajput, Kanya Satis, Sonnya Dellarosa, Wenxuan Huang, and Obinna Agba. 2021. [cgans for cartoon to real-life images](#).
- Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. [COMET: A neural framework for MT evaluation](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. [Improving neural machine translation models with monolingual data](#).
- Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. [Style transfer from non-parallel text by cross-alignment](#).
- Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. 2020. [Portuguese named entity recognition using bert-crf](#).
- Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. [How to fine-tune bert for text classification?](#)
- Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.
- Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and*

Evaluation (LREC'12), Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#).

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. [Huggingface's transformers: State-of-the-art natural language processing](#).

Zhilu Zhang and Mert R. Sabuncu. 2018. [Generalized cross entropy loss for training deep neural networks with noisy labels](#).

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. [Unpaired image-to-image translation using cycle-consistent adversarial networks](#).

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2023. [Multilingual machine translation with large language models: Empirical results and analysis](#).

B.4 Exploration of the CycleGN Framework for Low-Resource Languages

Exploration of the CycleGN Framework for Low-Resource Languages

Sören Dréano

ML-Labs
Dublin City University
soren.dreano2@mail.dcu.ie

Derek Molloy

School of Electronic Engineering
Dublin City University
derek.molloy@dcu.ie

Noel Murphy

School of Electronic Engineering
Dublin City University
noel.murphy@dcu.ie

Abstract

CycleGN is a Neural Machine Translation framework relying on the Transformer architecture. Its approach is similar to a Discriminator-less CycleGAN, specifically tailored for non-parallel text datasets.

The foundational concept of our research posits that in an ideal scenario, retro-translations of generated translations should revert to the original source sentences. Consequently, a pair of models can be trained using a Cycle Consistency Loss only, with one model translating in one direction and the second model in the opposite direction.

One of the main advantages of such an approach is that it makes it possible to learn with non-parallel datasets, which are by definition rare and short for low-resource languages. In order to verify this hypothesis and as a contribution to the WMT24 challenge, CycleGN models were trained for both the “Translation into Low-Resource Languages of Spain” and “Low-Resource Indic Language Translation” tasks. These submissions fall under the “constrained” category, as no pre-trained translation model was used, and the models were trained using the provided datasets.

Given that the CycleGN architecture demonstrated its capacity to learn from non-parallel datasets, the authors anticipated that it would similarly be effective in learning from low-resource languages. However, preliminary results indicate that, for most low-resource language pairs, the models did not exhibit significant learning ability. This study explores this lack of learning.

1 Introduction

The introduction of the Transformer architecture (Vaswani et al., 2017) marked a significant advancement in Machine Translation, rapidly gaining widespread adoption. Its parallelized structure enhanced computational efficiency, allowing for the integration of a larger number of parameters.

Neural Machine Translation (NMT) relies on extensive text corpora, structured as aligned pairs, where sentences of equivalent meaning are available in at least two different languages. This alignment is crucial for initiating model training to establish linguistic connections. Ongoing efforts, such as OPUS (Tiedemann and Thottingal, 2020) and Tatoeba (Tiedemann, 2012), focus on providing public access to these datasets. However, parallel datasets represent only a small fraction of the total data available in monolingual datasets.

While large parallel corpora exist for many language pairs, the ability to utilize monolingual datasets alone would greatly increase the available training data. This approach is particularly advantageous for low-resource languages, with limited parallel text corpora.

Back-translation (Sennrich et al., 2016) is a technique that enhances training data by using a pre-trained machine translation (MT) model to translate sentences from a monolingual dataset, creating synthetic parallel pairs. This method allows for the generation of additional training examples in situations where parallel corpora are scarce.

This research builds on the concept that translating a sentence from a source language to a target language, and then back-translating it to the source language, provides a means to evaluate the effectiveness of the translation models. By comparing the original sentence with the machine-generated back-translation, the discrepancy is then quantified using a Cycle Consistency Loss, which serves as a metric for model performance and guides the backpropagation of gradients within the neural networks. This approach is analogous to techniques used in Image-to-Image Translation, such as the CycleGAN framework proposed by Zhu et al. (2017).

2 Previous work

The TextCycleGAN model (Lorandi et al., 2023), although not based on the Transformer architecture

or focused on Machine Translation (MT), introduced a novel approach for text style transfer. This method applied a CycleGAN to the Yelp dataset, enabling the model to learn mappings between positive and negative textual styles without the need for paired examples.

Shen et al. (2017) demonstrated the potential of training two encoder-decoder networks in an unsupervised manner, allowing for the sharing of a latent space and facilitating style transfer. Similarly, Lample et al. (2018) extended this technique to the MT domain, proving that effective translation can be achieved without relying on parallel datasets.

3 Definitions

Machine Translation models are most commonly trained using “parallel” datasets, which are structured collections of text pairs. Each pair comprises a segment of text in a source language and its translation in the target language. A non-parallel dataset on the other hand does not consist in pairs of text segments, consequently the source and target sentences do not share any explicit correspondence.

In the context of this study, the datasets are “permuted”. A permuted dataset is defined as a parallel dataset wherein the sentences of one language have been systematically rearranged. Consequently, this results in a non-parallel corpus where it is guaranteed that each sentence has a corresponding translation located at an unspecified index within the dataset.

4 Datasets

The PILAR dataset (Galiano-Jiménez et al., 2024) has been used exclusively for the low-resource languages of Spain. Using a parallel curated dataset as a starting point ensures that the dataset is non-parallel by permuting the sentences. For each Iberian language, both a literary and a crawled versions were available in the PILAR datasets and have been merged for training. The development sets of the PILAR dataset are translations of the development sets of the FLORES dataset (NLLB Team et al., 2022), which is an evaluation benchmark for multilingual machine translation.

The Low-Resource Indic Language Translation task was also part of the WMT23 (Pal et al., 2023). The datasets were kept the same between the two editions.

Table 1 references the number of sentences used for each language-pair.

Language Pair	Number of lines	Number of epochs
Spanish-Aragonese	84,703	10
Spanish-Asturian	38,869	10
English-Assamese	2,624,715	1
English-Khasi	182,737	3
English-Manipuri	2,144,897	1
English-Mizo	1,909,823	1

Table 1: Number of sentences for each language pair and number of epochs during training

5 Training

For clarity and consistency, the mathematical notations from the original CycleGAN framework will be adopted in this study. The objective is to develop two Neural Machine Translation (NMT) models for two languages, $\mathcal{X}$ and $\mathcal{Y}$, using their respective datasets. Specifically, we aim to construct models $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$ and $\mathcal{F} : \mathcal{Y} \mapsto \mathcal{X}$ such that, in the ideal scenario of perfect translation, the relationships $\mathcal{G}(\mathcal{F}(y)) = y$ and $\mathcal{F}(\mathcal{G}(x)) = x$, with $x \in \mathcal{X}$ and for $y \in \mathcal{Y}$.

To achieve this, the Cross-Entropy Loss (CEL) (Zhang and Sabuncu, 2018) is utilised as the Cycle Consistency Loss (CCL), which measures the distance between the original sentence and its doubly translated counterpart, thereby guiding the computation of gradients.

Furthermore, similar to the original CycleGAN implementation, our study also incorporates an Identity Loss (IL) to enhance training stability. This loss, also based on CEL, ensures that when the model $\mathcal{G}$, which maps $\mathcal{X} \mapsto \mathcal{Y}$, receives an input $y \in \mathcal{Y}$, the output remains unchanged, i.e., $\mathcal{G}(y) = y$. The same loss function is applied to $\mathcal{F}$, ensuring that $\mathcal{F}(x)$ remains equal to x , as illustrated in Figure 1.

Further details of the training process, including the specific methodologies, vocabulary organization and pretraining, are comprehensively discussed in the CycleGN submission for the WMT24 main translation task. Readers interested in the full technical details are encouraged to refer to that publication for a more complete understanding of the training framework.

5.1 Model architecture

The architecture used for both models, $\mathcal{G}$ and $\mathcal{F}$, is the Marian framework (Junczys-Dowmunt et al., 2018) implemented by Huggingface’s Transformers library (Wolf et al., 2020), which is licensed under the Apache Licence. While most parameters

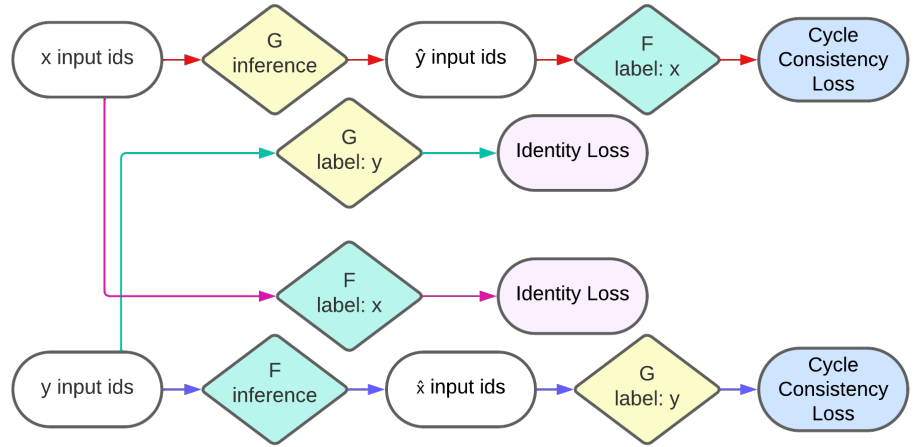


Figure 1: CycleGN training process

follow the default configuration, Table 2 references the changes that were made in order to reduce the computational cost of the architecture.

Parameter	Huggingface	Current work
Vocabulary size	58,101	32,000
Encoder layers	12	6
Decoder layers	12	6
Encoder attention heads	16	8
Decoder attention heads	16	8
Encoder feed-forward	4096	2048
Decoder feed-forward	4096	2048
Position embeddings	1024	128
Activation function	GELU	ReLU

Table 2: Non-default parameters in the configuration of Marian Transformer models

6 Results

Even if tracking the CCL is an inexpensive manner to estimate the progress of the training of the CycleGN architecture, a low loss value can also hide an absence of translation. Indeed, as there is no Discriminator to ensure that $\hat{x}$ belongs to $\mathcal{X}$ and $\hat{y}$ belongs to $\mathcal{Y}$, $\mathcal{G}$ and $\mathcal{F}$ will converge towards $x = \hat{y} = \hat{x}$ and $y = \hat{x} = \hat{y}$, as this approach achieves an optimal outcome on the CCL function, registering a value of zero. This is why an evaluation metric such as COMET is crucial to assess the progression of the CycleGN framework. To measure the performances of CycleGN, every 1,000th batch the CCL was averaged.

6.1 Indic Languages

Tracking the evolution of the CDC clearly shows the absence of learning in the four language pairs examined. The evolution of the CCL is particularly chaotic, which is partly due to an imbalance of class. Table 3 displays the average number of tokens in the Indic datasets depending on the language. In 3 of the 4 cases, the difference is large, i.e. sentences where the difference in the number of tokens is more than 10%.

Language pair	Length of source	Length of target
English-Assamese	33.10	22.81
Encoder layers	24.09	75.27
English-Manipuri	24.09	26.07
English-Mizo	32.05	17.55

Table 3: Average number of tokens in sentences

Figures 2, 3, 4 and 5 display the respective evolution of the Cycle Consistency Loss during the training of the language-pairs English-Assamese, English-Khasi, English-Manipuri and English-Mizo.

Contrary to what the authors had hoped for on the basis of previous results obtained for the main task of the WMT24 challenge, no model followed the expected learning curve, i.e. G and F models with a close and slowly decreasing Cycle Consistency Loss.

To reduce this imbalance of class, it may be necessary to manually adjust the size of the sentences.

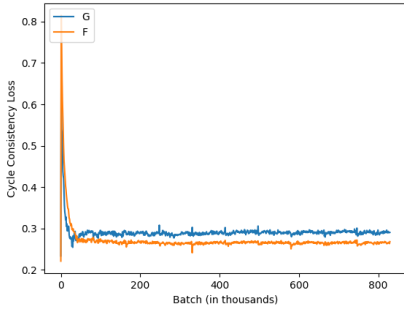


Figure 2: Evolution of the Cycle Consistency Loss during the training of the English-Assamese model

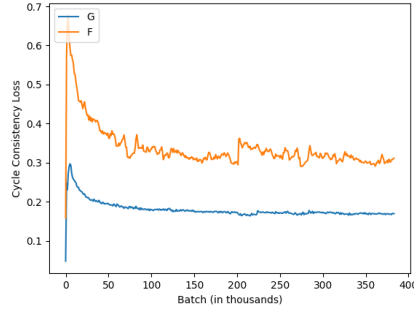


Figure 5: Evolution of the Cycle Consistency Loss during the training of the English-Mizo model

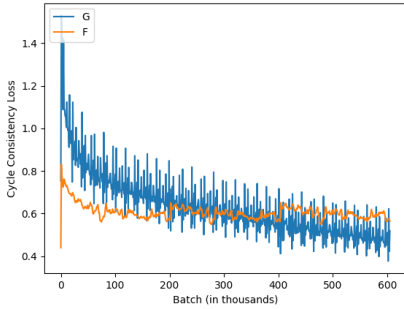


Figure 3: Evolution of the Cycle Consistency Loss during the training of the English-Khasi model

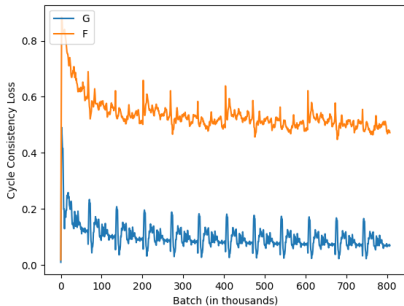


Figure 4: Evolution of the Cycle Consistency Loss during the training of the English-Manipuri model

This can be done by choosing another tokenization method, selectively choosing phrases to keep only those of a similar size, or by trimming sentences to lengthen or shorten them as required.

6.2 Iberian Languages

As with Indic Languages, CycleGN was unable to learn from the datasets provided. However, it was not due to an imbalance of classes in this case, but rather because the classes were too close together, as the Iberian languages are very close to the source language, Spanish.

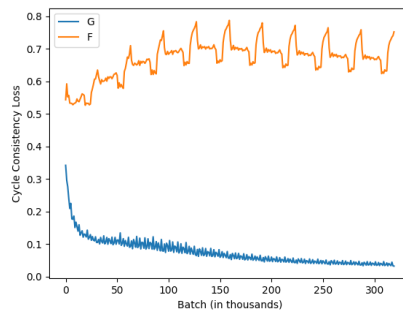


Figure 6: Evolution of the Cycle Consistency Loss during the training of the Spanish-Aragonese model

Rather than translating directly from Spanish into Aranese or Asturian, it is possible that translation can be achieved by using a different intermediate language such as English. Thus, two CycleGN models would have to be trained, the first to translate from Aranese or Asturian into English, and the second from English into Spanish. This would double the training time for an already expensive framework.

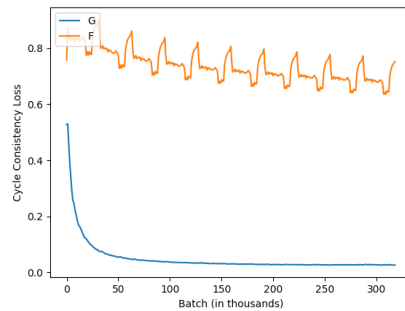


Figure 7: Evolution of the Cycle Consistency Loss during the training of the Spanish-Asturian model

7 Conclusion

In conclusion, while the training process demonstrated significant progress and effective translation capabilities in the main study, the results presented in this paper reveal several challenges that prevented similar success. The issues identified, particularly in relation to both class imbalance and class proximity, indicate that further refinement and investigation are necessary. Future research should focus on addressing these challenges, with the aim of optimizing the training process and overcoming the outlined issues. Resolving these problems is crucial for realizing the full potential of the framework within the context discussed in this paper.

8 Future Work

Further investigations will benefit from the incorporation of a more extensive dataset and an exploration of larger model architectures. Future work also include methods discussed in Section 6 to allow translation training.

8.1 Larget dataset

The current work has been trained on a small dataset compared to MT standards. Future work should try to see how convergence progresses with more iterations. Further computational optimizations are probably necessary to shorten the training time required.

8.2 Larger models

The current architecture relies on a total of 158,769,152 parameters, which is only about a third of the size of the default in the Huggingface library.

9 Source Code

The source code of CycleGN is available at <https://github.com/SorenDreano/CycleGN>.

Limitations

The investigation acknowledges certain inherent limitations which may impact the generalizability and applicability of the findings.

Language diversity

Another issue that arises from the computing cost of CycleGN is the lack in language diversity. Indeed, our current work only used the English-German and Chinese-English language pairs. Consequently, it cannot be certain that the approach presented can be applied to other languages and all alphabets. This is why CycleGN is taking part in WMT24, to explore the framework’s performance on a wide range of language pairs.

Training limitations

Due to time constraints and the fact that CycleGN is a computationally expensive architecture, it was not possible to train the Spanish-Aranese pair. Similarly, the training of all models was stopped early, before reaching performance stagnation.

Ethics Statement

This study, focusing on the training of NMT models using non-parallel datasets, adheres to the highest ethical standards in research. We recognize the critical importance of ethical considerations in computational linguistics and machine learning, especially as they pertain to data sourcing, model development, and potential impacts on various linguistic communities.

Our research utilizes publicly available, non-parallel linguistic datasets. We ensure that all data is sourced following legal and ethical guidelines, respecting intellectual property rights and privacy concerns.

In our commitment to scientific integrity, we maintain transparency in our research methodologies, model development, and findings. We aim to make our results reproducible and accessible to the scientific community, contributing positively to the field of machine translation.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foun-

dation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

- Aarón Galiano-Jiménez, Felipe Sánchez-Martínez, and Juan Antonio Pérez-Ortiz. 2024. [Pilar](#).
- Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. [Marian: Fast neural machine translation in C++](#). In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.
- Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc’Aurelio Ranzato. 2018. [Unsupervised machine translation using monolingual corpora only](#).
- Michela Lorandi, Maram A.Mohamed, and Kevin McGuinness. 2023. [Adapting the CycleGAN Architecture for Text Style Transfer](#). *Irish Machine Vision and Image Processing Conference*.
- NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Hefernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia-Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. [No language left behind: Scaling human-centered machine translation](#).
- Santanu Pal, Partha Pakray, Sahinur Rahman Laskar, Lenin Laitonjam, Vanlalmuansangi Khenglawt, Sunita Warjri, Pankaj Kundan Dadure, and Sandeep Kumar Dash. 2023. [Findings of the WMT 2023 shared task on low-resource Indic language translation](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 682–694, Singapore. Association for Computational Linguistics.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. [Improving neural machine translation models with monolingual data](#).
- Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. [Style transfer from non-parallel text by cross-alignment](#).
- Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.
- Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC’12)*, Istanbul, Turkey. European Language Resources Association (ELRA).
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#).
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. [Huggingface’s transformers: State-of-the-art natural language processing](#).
- Zhilu Zhang and Mert R. Sabuncu. 2018. [Generalized cross entropy loss for training deep neural networks with noisy labels](#).
- Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. [Unpaired image-to-image translation using cycle-consistent adversarial networks](#).

Bibliography

- [1] IEC 60027-2. Letter symbols to be used in electrical technology - part 2: Telecommunications and electronics, 2000. URL <https://webstore.iec.ch/en/publication/30633>.
- [2] Mohammad Abdollahi, Tannaz Khaleghi, and Kai Yang. An integrated feature learning approach using deep learning for travel time prediction. *Expert Systems with Applications*, 139:112864, 2020. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2019.112864>. URL <https://www.sciencedirect.com/science/article/pii/S0957417419305743>.
- [3] Saddam Abdulwahab, Mohammed Jabreel, and Dr Moreno. *Deep Learning Models for Paraphrases Identification*. PhD thesis, University Rovira I Virgili, 09 2017. URL https://webs-deim.urv.cat/~itaka/itaka2/PDF/acabats/MEMORIA%20_TFM_SaddamAbdulwahab.pdf.
- [4] Acheampong Francisca Adoma, Nunoo-Mensah Henry, and Wenyu Chen. Comparative analyses of bert, roberta, distilbert, and xlnet for text-based emotion recognition. In *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 117–121, 2020. doi: 10.1109/ICCWAMTIP51612.2020.9317379.
- [5] Manel Affi and Chiraz Latiri. Be-blc: Bert-elmo-based deep neural network architecture for english named entity recognition task. *Procedia Computer Science*, 192: 168–181, 2021. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2021.08.018>. URL <https://www.sciencedirect.com/science/article/pii/S1877050921015052>. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021.
- [6] Natalia Gimelshein et al. What every user should know about mixed precision training in pytorch. <https://pytorch.org/blog/what-every-user-should-know-about-mixed-precision-training-in-pytorch/>, Jul 2022.
- [7] Rajen Chatterjee et al. Findings of the 2021 conference on machine translation (WMT21). In Marta R. Costa-jussa et al., editors, *Proceedings of the Sixth Conference on Machine Translation*, pages 1–88, Online, November 2021. Association for Computational Linguistics. URL <https://aclanthology.org/2021.wmt-1.1/>.
- [8] Liliya Akhtyamova. Named entity recognition in spanish biomedical literature: Short review and bert model. In *2020 26th Conference of Open Innovations Association (FRUCT)*, pages 1–7, 2020. doi: 10.23919/FRUCT48808.2020.9087359.

- [9] Shivaji Alaparthi and Manit Mishra. Bert: a sentiment analysis odyssey. *Journal of Marketing Analytics*, 9(2):118–126, Jun 2021. ISSN 2050-3326. doi: 10.1057/s41270-021-00109-8. URL <https://doi.org/10.1057/s41270-021-00109-8>.
- [10] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.
- [11] Aziz Alotaibi. Deep generative adversarial networks for image-to-image translation: A review. *Symmetry*, 12(10), 2020. ISSN 2073-8994. doi: 10.3390/sym12101705. URL <https://www.mdpi.com/2073-8994/12/10/1705>.
- [12] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE*, 2(1):1–18, 2015.
- [13] Chittaranjan Andrade. Age as a variable: Continuous or categorical? *Indian J Psychiatry*, 59(4):524–525, October 2017.
- [14] Dmitry Lepikhin et al. Palm 2 technical report, 2023. URL <https://arxiv.org/abs/2305.10403>.
- [15] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/arjovsky17a.html>.
- [16] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In *Proceedings DCC'97. Data Compression Conference*, pages 201–210. IEEE, 1997.
- [17] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. An effective approach to unsupervised machine translation. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 194–203, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1019. URL <https://aclanthology.org/P19-1019/>.
- [18] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL <https://arxiv.org/abs/1607.06450>.
- [19] Will Badr. The secret to improved nlp: An in-depth look at the nn.embedding layer in pytorch. <https://towardsdatascience.com/the-secret-to-improve-d-nlp-an-in-depth-look-at-the-nn-embedding-layer-in-pytorch-6e901e193e16>, Jan 2023.
- [20] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. URL <https://arxiv.org/abs/1409.0473>.

- [21] Kai Dang et al. Qwen technical report, 2023. URL <https://arxiv.org/abs/2309.16609>.
- [22] Komal Bajaj, Dushyant Kumar Singh, and Mohd. Aquib Ansari. Autoencoders based deep learner for image denoising. *Procedia Computer Science*, 171:1535–1541, 2020. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2020.04.164>. URL <https://www.sciencedirect.com/science/article/pii/S1877050920311431>. Third International Conference on Computing and Network Communications (CoCoNet’19).
- [23] Federico Baldassarre. How do i check the number of parameters of a model?, 2018. URL <https://discuss.pytorch.org/t/how-do-i-check-the-number-of-parameters-of-a-model/4325/9>.
- [24] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, AAAI’87, page 279–284. AAAI Press, 1987. ISBN 0934613427.
- [25] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Jade Goldstein, Alon Lavie, Chin-Yew Lin, and Clare Voss, editors, *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL <https://aclanthology.org/W05-0909>.
- [26] Fang Bao, Michael Neumann, and Ngoc Thang Vu. CycleGAN-Based Emotion Style Transfer as Data Augmentation for Speech Emotion Recognition. In *Proc. Interspeech 2019*, pages 2828–2832, 2019. doi: 10.21437/Interspeech.2019-2293.
- [27] Christian Federmann et al. Findings of the 2020 conference on machine translation (WMT20). In Marta R. Costa-jussà et al., editors, *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, Online, November 2020. Association for Computational Linguistics. URL <https://aclanthology.org/2020.wmt-1.1/>.
- [28] Elissavet Batziou, Konstantinos Ioannidis, Ioannis Patras, Stefanos Vrochidis, and Ioannis Kompatsiaris. Artistic neural style transfer using cyclegan and fabemid by adaptive information selection. *Pattern Recognition Letters*, 165:55–62, 2023. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2022.11.026>. URL <https://www.sciencedirect.com/science/article/pii/S0167865522003567>.
- [29] Sefi Bell-Kligler, Assaf Shocher, and Michal Irani. Blind super-resolution kernel estimation using an internal-gan. In E. Fox et al., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/5fd0b37cd7dbbb00f97ba6ce92bf5add-Paper.pdf.
- [30] F Bellard. Lossless data compression with neural networks. <https://bellard.org/nncp/nncp.pdf>, 2019.

- [31] James Bernhard. Alternatives to the scaled dot product for attention in the transformer neural network architecture, 2023. URL <https://arxiv.org/abs/2311.09406>.
- [32] B. Joyce Beula Rani and L. Sumathi M. E. Survey on applying gan for anomaly detection. In *2020 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–5, 2020. doi: 10.1109/ICCCI48352.2020.9104046.
- [33] Leo Gao et al. GPT-NeoX-20B: An open-source autoregressive language model. In Angela Fan, Suzana Ilic, Thomas Wolf, and Matthias Gallé, editors, *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.bigscience-1.9. URL <https://aclanthology.org/2022.bigscience-1.9/>.
- [34] Simran Arora et al. On the opportunities and risks of foundation models, 2022. URL <https://arxiv.org/abs/2108.07258>.
- [35] Rafal Jozefowicz et al. Generating sentences from a continuous space. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1002. URL <https://aclanthology.org/K16-1002/>.
- [36] Eleftheria Briakou, Di Lu, Ke Zhang, and Joel Tetreault. Olá, bonjour, salve! XFORMAL: A benchmark for multilingual formality style transfer. In Iz Beltagy et al., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3199–3216, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.256. URL <https://aclanthology.org/2021.naacl-main.256>.
- [37] J Briskilal and C.N. Subalalitha. An ensemble model for classifying idioms and literal texts using bert and roberta. *Information Processing and Management*, 59(1):102756, 2022. ISSN 0306-4573. doi: <https://doi.org/10.1016/j.ipm.2021.102756>. URL <https://www.sciencedirect.com/science/article/pii/S0306457321002375>.
- [38] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1151. URL <https://aclanthology.org/D17-1151/>.
- [39] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019. URL <https://arxiv.org/abs/1809.11096>.

- [40] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. URL <https://aclanthology.org/J93-2003>.
- [41] Berfu Büyüköz, Ali Hürriyetoglu, and Arzucan Özgür. Analyzing ELMo and DistilBERT on socio-political news classification. In Ali Hürriyetoglu, Erdem Yörük, Vanni Zavarella, and Hristo Tanev, editors, *Proceedings of the Workshop on Automated Extraction of Socio-political Events from News 2020*, pages 9–18, Marseille, France, May 2020. European Language Resources Association (ELRA). ISBN 979-10-95546-50-4. URL <https://aclanthology.org/2020.aespen-1.4/>.
- [42] Qiang Cai, Mengxu Ma, Chen Wang, and Haisheng Li. Image neural style transfer: A review. *Computers and Electrical Engineering*, 108:108723, 2023. ISSN 0045-7906. doi: <https://doi.org/10.1016/j.compeleceng.2023.108723>. URL <https://www.sciencedirect.com/science/article/pii/S0045790623001477>.
- [43] George Cardona. Indo-aryan languages, May 2024. URL <https://www.britannica.com/topic/Indo-Aryan-languages>.
- [44] Alisson Carvalho and Rodolfo Azevedo. Towards a transprecision polymorphic floating-point unit for mixed-precision computing. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 56–63, 2019. doi: 10.1109/SBAC-PAD.2019.00022.
- [45] Rosario Catelli, Serena Pelosi, and Massimo Esposito. Lexicon-based vs. bert-based sentiment analysis: A comparative study in italian. *Electronics*, 11(3), 2022. ISSN 2079-9292. doi: 10.3390/electronics11030374. URL <https://www.mdpi.com/2079-9292/11/3/374>.
- [46] Aman Chadha. Transformers. *Distilled AI*, 2020. <https://aman.ai>.
- [47] Joshi Chaitanya. Transformers are graph neural networks. *NTU Graph Deep Learning Lab*, Feb 2020. <https://graphdeeplearning.github.io/post/transformers-are-gnns/>.
- [48] ML Reproducibility Challenge. Ml reproducibility challenge, 2018. URL <https://reproml.org>.
- [49] Yuan Chang, Lei Kong, Kejia Jia, and Qinglei Meng. Chinese named entity recognition method based on bert. In *2021 IEEE International Conference on Data Science and Computer Application (ICDSCA)*, pages 294–299, 2021. doi: 10.1109/ICDSCA53499.2021.9650256.
- [50] David Chen and William Dolan. Collecting highly parallel data for paraphrase evaluation. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 190–200, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL <https://aclanthology.org/P11-1020>.

- [51] Hongqian Chen, Mengxi Guan, and Hui Li. Arcyclegan: Improved cyclegan for style transferring of fruit images. *IEEE Access*, 9:46776–46787, 2021. doi: 10.1109/ACCESS.2021.3068094.
- [52] Jiyou Chen, Gaobo Yang, Huihuang Zhao, and Manimaran Ramasamy. Audio style transfer using shallow convolutional networks and random filters. *Multimedia Tools and Applications*, 79(21):15043–15057, Jun 2020. ISSN 1573-7721. doi: 10.1007/s11042-020-08798-6. URL <https://doi.org/10.1007/s11042-020-08798-6>.
- [53] Zhe Gan et al. Adversarial text generation via feature-mover’s distance. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 4671–4682, Red Hook, NY, USA, 2018. Curran Associates Inc.
- [54] Tianlong Chen, Yu Cheng, Zhe Gan, Jingjing Liu, and Zhangyang Wang. Data-efficient gan training beyond (just) augmentations: A lottery ticket perspective. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20941–20955. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/af4f00ca48321fb026865c5a1772dafd-Paper.pdf.
- [55] Xifeng Yan et al. How large a vocabulary does text classification need? a variational approach to vocabulary selection. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3487–3497, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1352. URL <https://aclanthology.org/N19-1352/>.
- [56] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*, pages 1–5, 2018. doi: 10.1109/WTS.2018.8363930.
- [57] David Chiang. A hierarchical phrase-based model for statistical machine translation. In Kevin Knight, Hwee Tou Ng, and Kemal Oflazer, editors, *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, pages 263–270, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219873. URL <https://aclanthology.org/P05-1033>.
- [58] Fethi Bougares et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL <https://aclanthology.org/D14-1179>.

- [59] Sanghyun Choo and Wonjoon Kim. A study on the evaluation of tokenizer performance in natural language processing. *Applied Artificial Intelligence*, 37(1):2175112, 2023. doi: 10.1080/08839514.2023.2175112. URL <https://doi.org/10.1080/08839514.2023.2175112>.
- [60] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018, 2019. doi: 10.1109/ICCVW.2019.00363.
- [61] Casey Chu, Andrey Zhmoginov, and Mark Sandler. Cyclegan, a master of steganography, 2017. URL <https://arxiv.org/abs/1712.02950>.
- [62] Chunduri. Normalization in machine learning and deep learning. *Transparent Deep Learning*, Oct 2018. <https://medium.com/transparent-deep-learning/normalization-in-machine-learning-and-deep-learning-56eb33eb0165>.
- [63] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.
- [64] Papers With Code. Gyafc dataset | papers with code, 2020. URL <https://paperswithcode.com/dataset/gyafc>.
- [65] Koray Kavukcuoglu et al. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12(null):2493–2537, November 2011. ISSN 1532-4435.
- [66] Pierre Colombo, Pablo Piantanida, and Chloé Clavel. A novel estimator of mutual information for learning to disentangle textual representations. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6539–6550, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.511. URL <https://aclanthology.org/2021.acl-long.511/>.
- [67] Common Crawl. Common crawl. <https://commoncrawl.org>, 2007.
- [68] Saloni Potdar et al. Towards cross-cultural machine translation with retrieval-augmented generation from multilingual knowledge graphs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16343–16360, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.914. URL <https://aclanthology.org/2024.emnlp-main.914/>.
- [69] Alexis Conneau and Guillaume Lample. *Cross-lingual language model pretraining*. Curran Associates Inc., Red Hook, NY, USA, 2019.

- [70] Aparna S. Varde et al. A framework for german-english machine translation with gru rnn. *CEUR Workshop Proceedings*, 3135, 2022. ISSN 1613-0073. Publisher Copyright: © 2022 Copyright for this paper by its authors.; 2022 Workshops of the EDBT/ICDT Joint Conference, EDBT/ICDT-WS 2022 ; Conference date: 29-03-2022.
- [71] Diogo Cortiz. Exploring transformers models for emotion recognition: a comparison of bert, distilbert, roberta, xlnet and electra. In *Proceedings of the 2022 3rd International Conference on Control, Robotics and Intelligent System, CCRIS '22*, page 230–234, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396851. doi: 10.1145/3562007.3562051. URL <https://doi.org/10.1145/3562007.3562051>.
- [72] Russ Cox and Filippo Valsorda. Transformers. *Go Blog*, May 2024. <https://go.dev/blog/chacha8rand>.
- [73] Menglong Cui, Jiangcun Du, Shaolin Zhu, and Deyi Xiong. Efficiently exploring large language models for document-level machine translation with in-context learning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10885–10897, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.646. URL <https://aclanthology.org/2024.findings-acl.646/>.
- [74] Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. Style transformer: Unpaired text style transfer without disentangled latent representation. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5997–6007, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1601. URL <https://aclanthology.org/P19-1601>.
- [75] Scott N. Dean and Scott A. Walper. Variational autoencoder for generation of antimicrobial peptides. *ACS Omega*, 5(33):20746–20754, Aug 2020. doi: 10.1021/acsomega.0c00442. URL <https://doi.org/10.1021/acsomega.0c00442>.
- [76] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. Image anomaly detection with generative adversarial networks. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 3–17, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10925-7.
- [77] S Deorowicz. Silesia compression corpus. <https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia>, 2009.
- [78] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

- [79] L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, May 1996. URL <https://www.rfc-editor.org/info/rfc1951>.
- [80] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423/>.
- [81] Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 545–554, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-24337-0.
- [82] Sören Dreano, Derek Molloy, and Noel Murphy. Embed_Llama: Using LLM embeddings for the metrics shared task. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 738–745, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.60. URL <https://aclanthology.org/2023.wmt-1.60>.
- [83] Sören Dreano, Derek Molloy, and Noel Murphy. Tokengram_F, a fast and accurate token-based chrF++ derivative. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 730–737, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.59. URL <https://aclanthology.org/2023.wmt-1.59>.
- [84] Sören Dreano, Derek Molloy, and Noel Murphy. CycleGN: A cycle consistent approach for neural machine translation. In Barry Haddow, Tom Kocmi, Philipp Koehn, and Christof Monz, editors, *Proceedings of the Ninth Conference on Machine Translation*, pages 165–175, Miami, Florida, USA, November 2024. Association for Computational Linguistics. URL <https://aclanthology.org/2024.wmt-1.10>.
- [85] Dmitry Lepikhin et al. Glam: Efficient scaling of language models with mixture-of-experts, 2022. URL <https://arxiv.org/abs/2112.06905>.
- [86] Ahmad Al-Dahle et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- [87] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL <http://jmlr.org/papers/v12/duchi11a.html>.

- [88] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1045. URL <https://aclanthology.org/D18-1045>.
- [89] Toqeer Ehsan, Amina Tehseen, Kengatharaiyer Sarveswaran, and Amjad Ali. AlphaBrains@DravidianLangTech: Sentiment analysis of code-mixed Tamil and Tulu by training contextualized ELMo word representations. In Bharathi R. Chakravarthi, Ruba Priyadharshini, Anand Kumar M, Sajeetha Thavareesan, and Elizabeth Sherly, editors, *Proceedings of the Third Workshop on Speech and Language Technologies for Dravidian Languages*, pages 152–159, Varna, Bulgaria, September 2023. INCOMA Ltd., Shoumen, Bulgaria. URL <https://aclanthology.org/2023.dravidianlangtech-1.20/>.
- [90] Stefan Elfving, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017. URL <https://arxiv.org/abs/1702.03118>.
- [91] Mete Eminagaoglu. A new similarity measure for vector space models in text classification and information retrieval. *Journal of Information Science*, 48(4): 463–476, 2022. doi: 10.1177/0165551520968055. URL <https://doi.org/10.1177/0165551520968055>.
- [92] Deniz Engin, Anil Genc, and Hazim Kemal Ekenel. Cycle-dehaze: Enhanced cylegan for single image dehazing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.
- [93] Maxim Enis and Mark Hopkins. From llm to nmt: Advancing low-resource machine translation with claude, 2024. URL <https://arxiv.org/abs/2404.13813>.
- [94] Ahmed El-Kishky et al. Beyond english-centric multilingual machine translation. *J. Mach. Learn. Res.*, 22(1), January 2021. ISSN 1532-4435.
- [95] André Martins et al. The devil is in the errors: Leveraging large language models for fine-grained machine translation evaluation. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 1066–1083, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.100. URL <https://aclanthology.org/2023.wmt-1.100/>.
- [96] Organization for Transformative Works. Archive of our own. <https://archiveofourown.org>, 2007.
- [97] Wikimedia Foundation. Wikimedia downloads, 2023. URL <https://dumps.wikimedia.org>.
- [98] Alexander Fraser. Shared task: Unsupervised mt and very low resource supervised mt, 2020. URL https://www.statmt.org/wmt20/unsup_and_very_low_res/.

- [99] Craig Stewart et al. Results of WMT22 metrics shared task: Stop using BLEU – neural metrics are better and more robust. In Rajen Chatterjee et al., editors, *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 46–68, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.wmt-1.2>.
- [100] Ricardo Rei et al. Results of WMT23 metrics shared task: Metrics might be guilty but references are not innocent. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 578–628, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.51. URL <https://aclanthology.org/2023.wmt-1.51>.
- [101] Xuhui Fu. Digital image art style transfer algorithm based on cyclegan. *Computational Intelligence and Neuroscience*, 2022:6075398, Jan 2022. ISSN 1687-5265. doi: 10.1155/2022/6075398. URL <https://doi.org/10.1155/2022/6075398>.
- [102] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11330. URL <https://ojs.aaai.org/index.php/AAAI/article/view/11330>.
- [103] Pascale Fung, Pierre Zweigenbaum, and Reinhard Rapp, editors. *Proceedings of the 2nd Workshop on Building and Using Comparable Corpora: from Parallel to Non-parallel Corpora (BUCC)*, Singapore, August 2009. Association for Computational Linguistics. URL <https://aclanthology.org/W09-3100/>.
- [104] Travis Hoppe et al. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL <https://arxiv.org/abs/2101.00027>.
- [105] María García-Ordás, José Benítez-Andrades, Isaías García, Carmen Benavides, and Hector Alaiz Moreton. Detecting respiratory pathologies using convolutional neural networks and variational autoencoders for unbalancing data. *Sensors*, 20, 02 2020. doi: 10.3390/s20041214.
- [106] Leon Gatys, Alexander Ecker, and Matthias Bethge. A neural algorithm of artistic style. *Journal of Vision*, 16(12):326–326, Sep 2016. ISSN 1534-7362. doi: 10.1167/16.12.326. URL <https://doi.org/10.1167/16.12.326>.
- [107] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016. doi: 10.1109/CVPR.2016.265.
- [108] Felix Gers, Nicol Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*, 3:115–143, 01 2002. doi: 10.1162/153244303768966139.
- [109] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterton, editors,

- Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL <https://proceedings.mlr.press/v9/glorot10a.html>.
- [110] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, mar 1991. ISSN 0360-0300. doi: 10.1145/103162.103163. URL <https://doi.org/10.1145/103162.103163>.
- [111] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246, 2016. doi: 10.1109/ICDMW.2016.0041.
- [112] David Warde-Farley et al. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020. ISSN 0001-0782. doi: 10.1145/3422622. URL <https://doi.org/10.1145/3422622>.
- [113] Google, 2023. URL <https://developers.google.com/machine-learning/gan/training>.
- [114] Google. Word embeddings. *TensorFlow Documentation*, 2023. https://www.tensorflow.org/text/guide/word_embeddings.
- [115] Thamme Gowda and Jonathan May. Finding the optimal vocabulary size for neural machine translation. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.352. URL <https://aclanthology.org/2020.findings-emnlp.352>.
- [116] Eric Grinstein, Ngoc Q. K. Duong, Alexey Ozerov, and Patrick Perez. Audio style transfer. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, April 2018. doi: 10.1109/icassp.2018.8461711. URL <http://dx.doi.org/10.1109/ICASSP.2018.8461711>.
- [117] Pierre Colombo et al. xcomet: Transparent machine translation evaluation through fine-grained error detection. *Transactions of the Association for Computational Linguistics*, 12:979–995, 2024. doi: 10.1162/tacl_a_00683. URL <https://aclanthology.org/2024.tacl-1.54/>.
- [118] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In R. Fergus et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf.
- [119] Yong Yu et al. Long text generation via adversarial training with leaked information. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference*

- and *Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.
- [120] Apoorv Gupta, Aman Bansal, and Vidhi Khanduja. Modern lossless compression techniques: Review, comparison and analysis. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–8, 2017. doi: 10.1109/ICECCT.2017.8117850.
 - [121] Geetika Gupta. What’s the difference between single-, double-, multi- and mixed-precision computing? <https://blogs.nvidia.com/blog/whats-the-difference-between-single-double-multi-and-mixed-precision-computing/>, Nov 2019.
 - [122] Gaëtan Hadjeres, Frank Nielsen, and Francois Pachet. GlSr-vae: Geodesic latent space regularization for variational autoencoder architectures. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017. doi: 10.1109/SSCI.2017.8280895.
 - [123] Mike Hamburg, Paul Kocher, and Mark E Marson. Analysis of intel’s ivy bridge digital random number generator. *Online: http://www.cryptography.com/public/pdf/Intel_TRN_G_Report_20120312.pdf*, 2012.
 - [124] Xiwu Han, Hanzhang Li, and Tiejun Zhao. Train the machine with what it can Learn—Corpus selection for SMT. In Pascale Fung, Pierre Zweigenbaum, and Reinhard Rapp, editors, *Proceedings of the 2nd Workshop on Building and Using Comparable Corpora: from Parallel to Non-parallel Corpora (BUCC)*, pages 27–33, Singapore, August 2009. Association for Computational Linguistics. URL <https://aclanthology.org/W09-3106/>.
 - [125] M Hart. Project gutenber. <https://www.gutenberg.org>, 1971.
 - [126] Dan He, Minh-Quang Pham, Thanh-Le Ha, and Marco Turchi. Gradient-based gradual pruning for language-specific multilingual neural machine translation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 654–670, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.43. URL <https://aclanthology.org/2023.emnlp-main.43/>.
 - [127] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1181. URL <https://aclanthology.org/D15-1181>.
 - [128] Junxian He, Xinyi Wang, Graham Neubig, and Taylor Berg-Kirkpatrick. A probabilistic formulation of unsupervised text style transfer, 2020. URL <https://arxiv.org/abs/2002.03912>.

- [129] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.
- [130] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.
- [131] Matthew Mulholland et al. Predicting grammaticality on an ordinal scale. In Kristina Toutanova and Hua Wu, editors, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 174–180, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-2029. URL <https://aclanthology.org/P14-2029>.
- [132] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL <https://arxiv.org/abs/1606.08415>.
- [133] Heewoo Jun et al. Deep learning scaling is predictable, empirically, 2017. URL <https://arxiv.org/abs/1712.00409>.
- [134] Mickel Hoang, Oskar Alija Bihorac, and Jacobo Rouses. Aspect-based sentiment analysis using BERT. In Mareike Hartmann and Barbara Plank, editors, *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 187–196, Turku, Finland, September–October 2019. Linköping University Electronic Press. URL <https://aclanthology.org/W19-6120>.
- [135] Marius Hobbhahn. Trends in machine learning hardware, Nov 2023. URL <https://epochai.org/blog/trends-in-machine-learning-hardware>.
- [136] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [137] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8): 2554–2558, 1982. doi: 10.1073/pnas.79.8.2554. URL <https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554>.
- [138] Takaaki Hori, Jaejin Cho, and Shinji Watanabe. End-to-end speech recognition with word-based rnn language models. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 389–396, 2018. doi: 10.1109/SLT.2018.8639693.
- [139] Weijun Wang et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. URL <https://arxiv.org/abs/1704.04861>.
- [140] Haifeng Hu, Liang Zhang, and Jianshen Wu. Hamming distance based approximate similarity text search algorithm. In *2015 Seventh International Conference on Advanced Computational Intelligence (ICACI)*, pages 1–6, 2015. doi: 10.1109/ICACI.2015.7184772.

- [141] Xinyu Hu and Xiaojun Wan. Exploring discourse structure in document-level machine translation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13889–13902, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.857. URL <https://aclanthology.org/2023.emnlp-main.857>.
- [142] Zhiqiang Hu, Roy Ka-Wei Lee, Charu C. Aggarwal, and Aston Zhang. Text style transfer: A review and experimental evaluation. *SIGKDD Explor. Newsl.*, 24(1):14–45, jun 2022. ISSN 1931-0145. doi: 10.1145/3544903.3544906. URL <https://doi.org/10.1145/3544903.3544906>.
- [143] Zhiqiang Hu, Roy Ka-Wei Lee, Charu C. Aggarwal, and Aston Zhang. Text style transfer: A review and experimental evaluation, 2023. URL <https://arxiv.org/abs/210.12742>.
- [144] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Toward controlled generation of text. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/hu17e.html>.
- [145] Anna Huang and others. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008)*, Christchurch, New Zealand, volume 4, pages 9–56, 2008.
- [146] Langlin Huang, Shuhao Gu, Zhang Zhuocheng, and Yang Feng. Enhancing neural machine translation with semantic units. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2264–2277, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.149. URL <https://aclanthology.org/2023.findings-emnlp.149/>.
- [147] Mengdi Huang, Chetan Tekur, and Michael Carilli. Introducing native pytorch automatic mixed precision for faster training on nvidia gpus. <https://pytorch.org/blog/accelerating-training-on-nvidia-gpus-with-pytorch-automatic-mixed-precision/>, Jul 2020.
- [148] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi: 10.1109/JRPROC.1952.273898.
- [149] Huggingface. Unigram tokenization. <https://huggingface.co/learn/nlp-course/en/chapter6/7>, 2024.
- [150] Huggingface. Wordpiece tokenization. <https://huggingface.co/learn/nlp-course/en/chapter6/6>, 2024.
- [151] Muhammad Huzaifah bin Md Shahrin and Lonce Wyse. Applying visual domain style transfer and texture synthesis techniques to audio: insights and challenges.

- Neural Computing and Applications*, 32(4):1051–1065, Feb 2020. ISSN 1433-3058. doi: 10.1007/s00521-019-04053-8. URL <https://doi.org/10.1007/s00521-019-04053-8>.
- [152] IBM, 2021. URL <https://www.ibm.com/topics/natural-language-processing>.
- [153] IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. doi: 10.1109/IEEESTD.2008.4610935.
- [154] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, page 448–456. JMLR.org, 2015.
- [155] Matthew Tang et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. doi: 10.1109/CVPR.2018.00286.
- [156] Harsh Jhamtani, Varun Gangal, Eduard Hovy, and Eric Nyberg. Shakespearizing modern language using copy-enriched sequence to sequence models. In Julian Brooke, Thamar Solorio, and Moshe Koppel, editors, *Proceedings of the Workshop on Stylistic Variation*, pages 10–19, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4902. URL <https://aclanthology.org/W17-4902>.
- [157] Y. Ji, B. Gong, M. Langguth, A. Mozaffari, and X. Zhi. Clgan: a generative adversarial network (gan)-based video prediction model for precipitation nowcasting. *Geoscientific Model Development*, 16(10):2737–2752, 2023. doi: 10.5194/gmd-16-2737-2023. URL <https://gmd.copernicus.org/articles/16/2737/2023/>.
- [158] Devendra Singh Chaplot et al. Mistral 7b, 2023. URL <https://arxiv.org/abs/2310.06825>.
- [159] Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. Deep Learning for Text Style Transfer: A Survey. *Computational Linguistics*, 48(1): 155–205, 04 2022. ISSN 0891-2017. doi: 10.1162/coli_a_00426. URL https://doi.org/10.1162/coli_a_00426.
- [160] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2323–2332. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/jin18a.html>.
- [161] Yizhou Yu et al. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3365–3385, 2020. doi: 10.1109/TVCG.2019.2921336.

- [162] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for non-parallel text style transfer. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 424–434, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1041. URL <https://aclanthology.org/P19-1041>.
- [163] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, Mar 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0192-5. URL <https://doi.org/10.1186/s40537-019-0192-5>.
- [164] Karen Sparck Jones. *Natural Language Processing: A Historical Review*, pages 3–16. Springer Netherlands, Dordrecht, 1994. ISBN 978-0-585-35958-8. doi: 10.1007/978-0-585-35958-8_1. URL https://doi.org/10.1007/978-0-585-35958-8_1.
- [165] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In Mirella Lapata, Phil Blunsom, and Alexander Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics. URL <https://aclanthology.org/E17-2068/>.
- [166] Kenneth Heafield et al. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P18-4020>.
- [167] Kunal Banerjee et al. A study of bfloat16 for deep learning training, 2019. URL <https://arxiv.org/abs/1905.12322>.
- [168] Swathi Edem Kanchan M. Tarwani. Survey on recurrent neural network in natural language processing. In *International Journal of Engineering Trends and Technology (IJETT)*, pages 301–304, 2017.
- [169] Ledell Wu et al. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL <https://aclanthology.org/2020.emnlp-main.550/>.
- [170] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018. URL <https://arxiv.org/abs/1710.10196>.
- [171] Lisa Kawai, Philippe Esling, and Tatsuya Harada. Attributes-aware deep music transformation. In *ISMIR*, pages 670–677, 2020.

- [172] Nina Khairova, Anastasiia Shapovalova, Orken Mamyrbayev, Nataliia Sharonova, and Kuralay. Using bert model to identify sentences paraphrase in the news corpus. In *CEUR Workshop Proceedings, volume 3171*, pages 38–48, 2022.
- [173] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5506–5518. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/kim21d.html>.
- [174] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL <https://aclanthology.org/D14-1181/>.
- [175] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL <https://arxiv.org/abs/1412.6980>.
- [176] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In Mohit Bansal and Heng Ji, editors, *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL <https://aclanthology.org/P17-4012>.
- [177] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184 vol.1, 1995. doi: 10.1109/ICASSP.1995.479394.
- [178] Tom Kocmi. Shared task: General machine translation, 2023. URL <http://www2.statmt.org/wmt23/translation-task.html>.
- [179] Tom Kocmi and Christian Federmann. Large language models are state-of-the-art evaluators of translation quality. In Mikhail Mikhailov et al., editors, *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 193–203, Tampere, Finland, June 2023. European Association for Machine Translation. URL <https://aclanthology.org/2023.eamt-1.19/>.
- [180] Christian Federmann et al. Findings of the 2022 conference on machine translation (WMT22). In Rajen Chatterjee et al., editors, *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.wmt-1.1/>.
- [181] Anton Dvorkovich et al. Findings of the 2023 conference on machine translation (WMT23): LLMs are here but not quite there yet. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 1–42, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.1. URL <https://aclanthology.org/2023.wmt-1.1>.

- [182] Anton Dvorkovich et al. Findings of the WMT24 general machine translation shared task: The LLM era is here but MT is not solved yet. In Barry Haddow, Tom Kocmi, Philipp Koehn, and Christof Monz, editors, *Proceedings of the Ninth Conference on Machine Translation*, pages 1–46, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.wmt-1.1. URL <https://aclanthology.org/2024.wmt-1.1/>.
- [183] Philipp Koehn, Franz J. Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003. URL <https://aclanthology.org/N03-1017>.
- [184] Aran Komatsuzaki. One epoch is all you need, 2019. URL <https://arxiv.org/abs/1909.06669>.
- [185] Kyogu Lee et al. Music mixing style transfer: A contrastive learning approach to disentangle audio effects. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. doi: 10.1109/ICASSP49357.2023.10096458.
- [186] M. V. Koroteev. Bert: A review of applications in natural language processing and understanding, 2021. URL <https://arxiv.org/abs/2103.11943>.
- [187] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018. URL <https://arxiv.org/abs/1806.08342>.
- [188] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL <https://aclanthology.org/P18-1007/>.
- [189] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL <https://aclanthology.org/D18-2012>.
- [190] Ria Kulshrestha. Nlp 101: Word2vec — skip-gram and cbow, Nov 2019. URL <https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314>.
- [191] Chih-Te Lai, Yi-Te Hong, Hong-You Chen, Chi-Jen Lu, and Shou-De Lin. Multiple text style transfer by using word-level conditional generative adversarial network with two-phase training. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3579–3584, Hong

- Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1366. URL <https://aclanthology.org/D19-1366>.
- [192] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI’15, page 2267–2273. AAAI Press, 2015. ISBN 0262511290.
- [193] Chelsea Finn et al. Stochastic adversarial video prediction, 2018. URL <https://arxiv.org/abs/1804.01523>.
- [194] Abdelrahman Mohamed et al. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL <https://aclanthology.org/2020.acl-main.703/>.
- [195] Gurvan L’Hostis, David Grangier, and Michael Auli. Vocabulary selection strategies for neural machine translation, 2016. URL <https://arxiv.org/abs/1610.0072>.
- [196] Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1169. URL <https://aclanthology.org/N18-1169>.
- [197] Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1169. URL <https://aclanthology.org/N18-1169/>.
- [198] Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1169. URL <https://aclanthology.org/N18-1169/>.
- [199] Qin Li, Xiaofan Zhang, JinJun Xiong, Wen-mei Hwu, and Deming Chen. Implementing neural machine translation with bi-directional gru and attention mechanism on fpgas using hls. In *Proceedings of the 24th Asia and South Pacific*

- Design Automation Conference, ASPDAC '19*, page 693–698, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450360074. doi: 10.1145/3287624.3287717. URL <https://doi.org/10.1145/3287624.3287717>.
- [200] Yu Cheng et al. Storygan: A sequential conditional gan for story visualization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [201] Jintao Li et al. Asymmetric gan for unpaired image-to-image translation. *IEEE Transactions on Image Processing*, 28(12):5881–5896, 2019. doi: 10.1109/TIP.2019.2922854.
- [202] Wai Lam et al. QuaSE: Sequence editing under quantifiable guidance. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3855–3864, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1420. URL <https://aclanthology.org/D18-1420>.
- [203] Elizabeth D. Liddy. Natural language processing. In *Encyclopedia of Library and Information Science*. Marcel Dekker, 2nd edition, 2001. ISBN 0-8247-2075-X. URL <https://surface.syr.edu/istpub/63/>.
- [204] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL <https://aclanthology.org/W04-1013>.
- [205] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Springer Cham, 2021. ISBN 978-3-031-02181-7. doi: 10.1007/978-3-031-02181-7. URL <https://doi.org/10.1007/978-3-031-02181-7>.
- [206] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 3:111–132, 2022. ISSN 2666-6510. doi: <https://doi.org/10.1016/j.aiopen.2022.10.001>. URL <https://www.sciencedirect.com/science/article/pii/S2666651022000146>.
- [207] Pietro Perona et al. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.
- [208] Lorelei Lingard, Sayra Cristancho, Eva Kathrin Hennel, Christina St-Onge, and Marije van Braak. When english clashes with other languages: Insights and cautions from the writer’s craft series. *Perspect. Med. Educ.*, 10(6):347–351, December 2021.
- [209] Dayiheng Liu, Jie Fu, Yidan Zhang, Chris Pal, and Jiancheng Lv. Revision in continuous space: Unsupervised text style transfer without adversarial learning.

- Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8376–8383, Apr. 2020. doi: 10.1609/aaai.v34i05.6355. URL <https://ojs.aaai.org/index.php/AAAI/article/view/6355>.
- [210] Xiaodan Liang et al. Composable text controls in latent space with ODEs. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16543–16570, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.1030. URL <https://aclanthology.org/2023.emnlp-main.1030/>.
- [211] Jinsong Su et al. Continual learning for multilingual neural machine translation via dual importance-based model division. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12011–12027, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.736. URL <https://aclanthology.org/2023.emnlp-main.736/>.
- [212] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. In Kristina Toutanova and Hua Wu, editors, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1491–1500, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1140. URL <https://aclanthology.org/P14-1140>.
- [213] Yuanyuan Yang et al. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, September 2023. ISSN 2950-1628. doi: 10.1016/j.metrad.2023.100017. URL <http://dx.doi.org/10.1016/j.metrad.2023.100017>.
- [214] Mandar Joshi et al. Roberta: A robustly optimized bert pretraining approach, 2019. URL <https://arxiv.org/abs/1907.11692>.
- [215] Zhiyue Liu, Jiahai Wang, and Zhiwei Liang. Catgan: Category-aware generative adversarial networks with hierarchical evolutionary learning for category text generation, 2019. URL <https://arxiv.org/abs/1911.06641>.
- [216] Liu, Wenbin, Wen, Bojian, Gao, Shang, Zheng, Jiesheng, and Zheng, Yinlong. A multi-label text classification model based on elmo and attention. *MATEC Web Conf.*, 309:03015, 2020. doi: 10.1051/mateconf/202030903015. URL <https://doi.org/10.1051/mateconf/202030903015>.
- [217] Lajanugen Logeswaran, Honglak Lee, and Samy Bengio. Content preserving text generation with attribute controls. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 5108–5118, Red Hook, NY, USA, 2018. Curran Associates Inc.
- [218] Michela Lorandi, A Mohamed, Kevin McGuinness, and others. Adapting the cylegan architecture for text style transfer. *IMVIP*, 2023.

- [219] Yongyi Lu, Yu-Wing Tai, and Chi-Keung Tang. Attribute-guided face generation using conditional cyclegan. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.
- [220] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [221] Simone A. Ludwig. Comparison of time series approaches applied to greenhouse gas analysis: Anfis, rnn, and lstm. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2019. doi: 10.1109/FUZZ-IEEE.2019.8859013.
- [222] Baobao Chang et al. A dual reinforcement learning framework for unsupervised text style transfer. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5116–5122. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/711. URL <https://doi.org/10.24963/ijcai.2019/711>.
- [223] Matouš Macháček and Ondřej Bojar. Results of the WMT14 metrics shared task. In Philipp Koehn et al., editors, *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 293–301, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3336. URL <https://aclanthology.org/W14-3336>.
- [224] M Mahoney. Large text compression benchmark. <http://mattmahoney.net/dc/text.html>, 2006.
- [225] Angshul Majumdar. Blind denoising autoencoder. *IEEE Transactions on Neural Networks and Learning Systems*, 30(1):312–317, 2019. doi: 10.1109/TNNLS.2018.2838679.
- [226] Samat Mamitimin and Min Hou. Chinese-Uyghur sentence alignment: An approach based on anchor sentences. In Pascale Fung, Pierre Zweigenbaum, and Reinhard Rapp, editors, *Proceedings of the 2nd Workshop on Building and Using Comparable Corpora: from Parallel to Non-parallel Corpora (BUCC)*, pages 38–45, Singapore, August 2009. Association for Computational Linguistics. URL <https://aclanthology.org/W09-3108/>.
- [227] Momina Masood, Marriam Nawaz, Khalid Malik, Ali Javed, and Aun Irtaza. Deep-fakes generation and detection: state-of-the-art, open challenges, countermeasures, and way forward, 02 2021.
- [228] David McCandless, Tom Evans, and Paul Barton. The rise and rise of a.i. large language models (llms) and their associated bots like chatgpt, March 2024. URL <https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/>.
- [229] David D. McDonald and James D. Pustejovsky. A computational theory of prose style for natural language generation. In Maghi King, editor, *Second Conference of*

- the European Chapter of the Association for Computational Linguistics, Geneva, Switzerland, March 1985. Association for Computational Linguistics. URL <https://aclanthology.org/E85-1027>.
- [230] Bharat Kaul et al. Ternary neural networks with fine-grained quantization, 2017. URL <https://arxiv.org/abs/1705.01462>.
- [231] Hanımur Mercan, Yaşar Akgün, and Mehmet Cem Odacıoğlu. The evolution of machine translation: A review study. *International Journal of Language and Translation Studies*, 4(1):104–116, 2024.
- [232] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.
- [233] Yajie Miao, Mohammad Gowayyed, and Florian Metze. Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 167–174, 2015. doi: 10.1109/ASRU.2015.7404790.
- [234] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL <https://arxiv.org/abs/1301.3781>.
- [235] Remi Mir, Bjarke Felbo, Nick Obradovich, and Iyad Rahwan. Evaluating style transfer for text. In Jill Burstein, Christy Doran, and Tamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 495–504, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1049. URL <https://aclanthology.org/N19-1049>.
- [236] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014. URL <https://arxiv.org/abs/1411.1784>.
- [237] Vidushi Mishra, SMT AGARWAL, and NEHA PURI. Comprehensive and comparative analysis of neural network. *INTERNATIONAL JOURNAL OF COMPUTER APPLICATION*, 2, 01 2018. doi: 10.26808/rs.ca.i8v2.15.
- [238] MIT. Direct to open: A bold, innovative model for open access to scholarship and knowledge, 2022. URL <https://direct.mit.edu/books/pages/direct-to-open>.
- [239] Ali Moussawi, Pierre Ibrahim, Bilal Said, and Khaleel Mershad. *Using Machine Learning to Enhance "Students' Assessment" Moodle Application*. PhD thesis, Arts, Sciences and Technology University in Lebanon, 05 2020.
- [240] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 09 2011. ISSN 1067-5027. doi: 10.1136/amiajnl-2011-000464. URL <https://doi.org/10.1136/amiajnl-2011-000464>.

- [241] N. Unnikrishnan Nair, P.G. Sankaran, and N. Balakrishnan. Chapter 3 - discrete lifetime models. In N. Unnikrishnan Nair, P.G. Sankaran, and N. Balakrishnan, editors, *Reliability Modelling and Analysis in Discrete Time*, pages 107–173. Academic Press, Boston, 2018. ISBN 978-0-12-801913-9. doi: <https://doi.org/10.1016/B978-0-12-801913-9.00003-8>. URL <https://www.sciencedirect.com/science/article/pii/B9780128019139000038>.
- [242] Mostofa Patwary et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476209. URL <https://doi.org/10.1145/3458817.3476209>.
- [243] E Nemerson. Squash compression benchmark. <https://quixdb.github.io/squash-benchmark/>, 2015.
- [244] Xuan-Phi Nguyen, Shafiq Joty, Thanh-Tung Nguyen, Wu Kui, and Ai Ti Aw. Cross-model back-translated distillation for unsupervised machine translation, 2021. URL <https://arxiv.org/abs/2006.02163>.
- [245] Ivan Niven. A simple proof that π is irrational. *Pi: A Source Book*, pages 276–276, 1947.
- [246] NVidia. Automatic mixed precision for deep learning, 2019. URL <https://developer.nvidia.com/automatic-mixed-precision>.
- [247] NVidia. Train with mixed precision, Mar 2020. URL <https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html>.
- [248] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928. doi: 10.1109/T-AIEE.1928.5055024.
- [249] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/odena17a.html>.
- [250] The Institute of Electrical and Inc Electronics Engineers. Ieee standard for floating-point arithmetic. Standard IEEE Std 754-2008, IEEE Computer Society, New York, NY, USA, August 2008. URL <https://web.archive.org/web/20160806053349/http://www.csee.umbc.edu/~tsimo1/CMSC455/IEEE-754-2008.pdf>.
- [251] University of Helsinki. Language technology research group at the university of helsinki. <https://huggingface.co/Helsinki-NLP>, 2024.
- [252] OpenAI. Embeddings. *OpenAI Documentation*, 2023. <https://platform.openai.com/docs/guides/embeddings>.

- [253] Lama Ahmad et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.
- [254] Andrew Or, Jerry Zhang, Evan Smothers, Khandelwal Kartikay, and Supriya Rao. Quantization-aware training for large language models with pytorch, Jul 2024. URL <https://pytorch.org/blog/quantization-aware-training/>.
- [255] Oracle. Opus codec transcoding support, 2022. URL https://docs.oracle.com/cd/E95618_01/html/sbc_scz810_acliconfiguration/GUID-9BC9897C-96DA-4F7C-8326-734280F85522.htm.
- [256] ORCID. Orcid, open researcher and contributor id, 2012. URL <https://orcid.org/>.
- [257] Phil Sidney Ostheimer, Mayank Kumar Nagda, Marius Kloft, and Sophie Fellenz. Text style transfer evaluation using large language models. In Sakriani Sakti et al., editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 15802–15822, Torino, Italia, May 2024. ELRA and ICCL. URL <https://aclanthology.org/2024.lrec-main.1373/>.
- [258] Kartikey Pant, Yash Verma, and Radhika Mamidi. Sentiinc: Incorporating sentiment information into sentiment transfer without parallel data. In Nicola Ferro et al., editors, *Advances in Information Retrieval*, pages 312–319, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45442-5.
- [259] Toni Pape. *The Aesthetics of Stealth: Digital Culture, Video Games, and the Politics of Perception*. The MIT Press, 10 2024. ISBN 9780262380768. doi: 10.7551/mitpress/15539.001.0001. URL <https://doi.org/10.7551/mitpress/15539.001.0001>.
- [260] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL <https://aclanthology.org/P02-1040>.
- [261] JoonKyu Park, Sanghyun Son, and Kyoung Mu Lee. Content-aware local gan for photo-realistic super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10585–10594, October 2023.
- [262] et al. Parker, Robert. English gigaword fifth edition, 2011. URL <https://catalog.ldc.upenn.edu/LDC2011T07>.
- [263] D.L. Parnas. Software aging. In *Proceedings of 16th International Conference on Software Engineering*, pages 279–287, 1994. doi: 10.1109/ICSE.1994.296790.
- [264] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors,

- Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL <https://proceedings.mlr.press/v28/pascanu13.html>.
- [265] James Bradbury et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [266] Ellie Pavlick and Joel Tetreault. An empirical analysis of formality in online communication. *Transactions of the Association for Computational Linguistics*, 4: 61–74, 2016. doi: 10.1162/tacl_a_00083. URL <https://aclanthology.org/Q16-1005>.
- [267] I Pavlov. 7-zip compression software. <https://www.7-zip.org>, 1999.
- [268] Jiaxin Wang et al. Contour-enhanced cyclegan framework for style transfer from scenery photos to chinese landscape paintings. *Neural Computing and Applications*, 34(20):18075–18096, Oct 2022. ISSN 1433-3058. doi: 10.1007/s00521-022-07432-w. URL <https://doi.org/10.1007/s00521-022-07432-w>.
- [269] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- [270] Christopher Clark et al. Deep contextualized word representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL <https://aclanthology.org/N18-1202/>.
- [271] Alina Beygelzimer et al. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *J. Mach. Learn. Res.*, 22(1), January 2021. ISSN 1532-4435.
- [272] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, April 2018. ISSN 1804-0462. doi: 10.2478/pralin-2018-0002. URL <http://dx.doi.org/10.2478/pralin-2018-0002>.
- [273] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In Chris Hokamp et al., editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL <https://aclanthology.org/W15-3049>.

- [274] Maja Popović. chrF++: words helping character n-grams. In Yvette Graham et al., editors, *Proceedings of the Second Conference on Machine Translation*, pages 612–618, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4770. URL <https://aclanthology.org/W17-4770>.
- [275] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. Style transfer through back-translation. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 866–876, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1080. URL <https://aclanthology.org/P18-1080>.
- [276] Shashank Prasanna. What’s new in mojo 24.3: Community contributions, pythonic collections and core language enhancements. <https://www.modular.com/blog/whats-new-in-mojo-24-3-community-contributions-pythonic-collections-and-core-language-enhancements>, May 2024.
- [277] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In Mirella Lapata, Phil Blunsom, and Alexander Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics. URL <https://aclanthology.org/E17-2025/>.
- [278] Lorenzo Puppi Vecchi, Eliane C. F. Maffezzoli, and Emerson Cabrera Paraiso. Transferring multiple text styles using cyclegan with supervised style latent space. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2022. doi: 10.1109/IJCNN55064.2022.9892978.
- [279] PyTorch. Gelu. <https://pytorch.org/docs/stable/generated/torch.nn.GELU.html>, 2020.
- [280] PyTorch. Relu. <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>, 2020.
- [281] PyTorch. Relu. <https://pytorch.org/docs/stable/generated/torch.nn.SiLU.html>, 2020.
- [282] PyTorch. Uniform. <https://pytorch.org/docs/stable/distributions.html#uniform>, 2024.
- [283] Wisam A. Qader, Musa M. Ameen, and Bilal I. Ahmed. An overview of bag of words;importance, implementation, applications, and challenges. In *2019 International Engineering Conference (IEC)*, pages 200–204, 2019. doi: 10.1109/IEC47844.2019.8950616.
- [284] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 0893-6080. doi: <https://doi.org/10>

- .1016/S0893-6080(98)00116-6. URL <https://www.sciencedirect.com/science/article/pii/S0893608098001166>.
- [285] Yongfeng Zhang et al. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1133–1136, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331341. URL <https://doi.org/10.1145/3331184.3331341>.
- [286] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and others. Improving language understanding by generative pre-training. *OpenAI*, 2018.
- [287] Dario Amodei et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [288] Sharan Narang et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), January 2020. ISSN 1532-4435.
- [289] Pranjal Singh Rajput, Kanya Satis, Sonnya Dellarosa, Wenxuan Huang, and Obinna Agba. cgans for cartoon to real-life images, 2021. URL <https://arxiv.org/abs/2101.09793>.
- [290] Alexander Coraspe Valdez Ramses. Understanding similarity measures for text analysis. <https://wittline.github.io/>, 2021. URL <https://wittline.github.io/distance-metrics/>.
- [291] Sudha Rao and Joel Tetreault. Dear sir or madam, may I introduce the GYAFC dataset: Corpus, benchmarks and metrics for formality style transfer. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 129–140, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1012. URL <https://aclanthology.org/N18-1012/>.
- [292] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection . In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Los Alamitos, CA, USA, June 2016. IEEE Computer Society. doi: 10.1109/CVPR.2016.91. URL <https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.91>.
- [293] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. <http://is.muni.cz/publication/884893/en>.
- [294] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. COMET: A neural framework for MT evaluation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online, November

2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.213. URL <https://aclanthology.org/2020.emnlp-main.213>.
- [295] Ana C Farinha et al. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In Rajen Chatterjee et al., editors, *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.wmt-1.52>.
- [296] Ana C Farinha et al. CometKiwi: IST-unbabel 2022 submission for the quality estimation shared task. In Rajen Chatterjee et al., editors, *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 634–645, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.wmt-1.60>.
- [297] Yadao Wang et al. Pangu-sigma: Towards trillion parameter language model with sparse heterogeneous computing, 2023. URL <https://arxiv.org/abs/2303.10845>.
- [298] Philip Resnik. Mining the web for bilingual text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 527–534, College Park, Maryland, USA, June 1999. Association for Computational Linguistics. doi: 10.3115/1034678.1034757. URL <https://aclanthology.org/P99-1068>.
- [299] Syed Z. Rizvi, Farshid Abbasi, and Javad Mohammadpour Velni. Model reduction in linear parameter-varying models using autoencoder neural networks. In *2018 Annual American Control Conference (ACC)*, pages 6415–6420, 2018. doi: 10.23919/ACC.2018.8431912.
- [300] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4364–4373. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/roberts18a.html>.
- [301] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. doi: 10.1162/tac1_a_00349. URL <https://aclanthology.org/2020.tac1-1.54/>.
- [302] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960. ISSN 0010-4620. doi: 10.1093/comjnl/3.3.175. URL <https://doi.org/10.1093/comjnl/3.3.175>.
- [303] Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280, December 2020. ISSN 2307-387X. doi: 10.1162/tac1_a_00313. URL http://dx.doi.org/10.1162/tac1_a_00313.

- [304] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. URL <https://arxiv.org/abs/1609.04747>.
- [305] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. "MIT Press", 1987.
- [306] Elizabeth Salesky, Andrew Runge, Alex Coda, Jan Niehues, and Graham Neubig. Optimizing segmentation granularity for neural machine translation. *Machine Translation*, 34(1):41–59, Apr 2020. ISSN 1573-0573. doi: 10.1007/s10590-019-09243-8. URL <https://doi.org/10.1007/s10590-019-09243-8>.
- [307] Shakirat A Salihu, Ifeoma P Onyekwere, Modinat A Mabayoje, and Hammed A Mojeed. Performance evaluation of manhattan and euclidean distance measures for clustering based automatic text summarization. *Journal of Engineering and Technology*, 4(1):135–139, 2019.
- [308] V Sanh. Roberta-base-finetuned-yelp-polarity, 2020. URL <https://huggingface.co/VictorSanh/roberta-base-finetuned-yelp-polarity>.
- [309] George Saon, Zoltán Tüske, Daniel Bolanos, and Brian Kingsbury. Advancing rnn transducer technology for speech recognition. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5654–5658, 2021. doi: 10.1109/ICASSP39728.2021.9414716.
- [310] Scarpoile. Fsd fsd. <https://archiveofourown.org/works/56337412/chapters/143138362>, 2024.
- [311] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015. ISSN 0893-6080. doi: 10.1016/j.neunet.2014.09.003. URL <http://dx.doi.org/10.1016/j.neunet.2014.09.003>.
- [312] Armand Joulin et al. CCMatrix: Mining billions of high-quality parallel sentences on the web. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6490–6500, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.507. URL <https://aclanthology.org/2021.acl-long.507>.
- [313] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. Bleurt: Learning robust metrics for text generation. In *Proceedings of ACL*, 2020.
- [314] S. Selva Birunda and R. Kanniga Devi. A review on word embedding techniques for text classification. In Jennifer S. Raj, Abdullah M. Iliyasu, Robert Bestak, and Zubair A. Baig, editors, *Innovative Data Communication Technologies and Application*, pages 267–281, Singapore, 2021. Springer Singapore. ISBN 978-981-15-9651-3.
- [315] Sreelekshmy Selvin, R Vinayakumar, E. A Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. Stock price prediction using lstm, rnn and cnn-sliding

- window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1643–1647, 2017. doi: 10.1109/ICACCI.2017.8126078.
- [316] Stanislaw Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 627–637, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1066. URL <https://aclanthology.org/D17-1066/>.
- [317] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL <https://aclanthology.org/P16-1162>.
- [318] Joelle Pineau et al. A hierarchical latent variable encoder-decoder model for generating dialogues. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.10983. URL <https://ojs.aaai.org/index.php/AAAI/article/view/10983>.
- [319] J Seward. bzip2: A high-quality data compressor. <http://www.bzip.org>, 1996.
- [320] Proff. Sadhana Shalu and M. Rifaya Meera. Neural machine translation for english to hindi using gru. *SSRN Electronic Journal*, 2021. doi: 10.2139/ssrn.3851323. URL <https://dx.doi.org/10.2139/ssrn.3851323>.
- [321] P Shapira. Delving into “delve”. <https://pshapira.net/2024/03/31/delving-into-delve/>, 2024.
- [322] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, page 6833–6844, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.
- [323] Apeksha Shewalkar, Deepika Nyavanandi, and Simone A. Ludwig. Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, 9(4):235–245, 2019. doi: doi:10.2478/jaiscr-2019-0006. URL <https://doi.org/10.2478/jaiscr-2019-0006>.
- [324] Shaykh Siddique, Tahmid Ahmed, Md. Rifayet Azam Talukder, and Md. Mohsin Uddin. English to bangla machine translation using recurrent neural network. *International Journal of Future Computer and Communication*, page 46–51, jun 2020. ISSN 2010-3751. doi: 10.18178/ijfcc.2020.9.2.564. URL <http://dx.doi.org/10.18178/ijfcc.2020.9.2.564>.
- [325] Thalles Santos Silva. A short introduction to generative adversarial networks. <https://sthalles.github.io>, 2017. URL <https://sthalles.github.io/intro-to-gans/>.

- [326] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR 2015)*, pages 1–14, 2015. doi: abs/1409.1556.
- [327] Anshika Rastogi et al. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167, 2017. doi: 10.1109/COMPTELIX.2017.8003957.
- [328] Kaleb E Smith and Anthony O Smith. Conditional gan for timeseries generation, 2020. URL <https://arxiv.org/abs/2006.16477>.
- [329] Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA, August 8-12 2006. Association for Machine Translation in the Americas. URL <https://aclanthology.org/2006.amta-papers.25>.
- [330] Matthew Snover, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. Fluency, adequacy, or HTER? Exploring different human judgments with a tunable MT metric. In Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder, editors, *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 259–268, Athens, Greece, March 2009. Association for Computational Linguistics. URL <https://aclanthology.org/W09-0441>.
- [331] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. In *International Conference on Machine Learning*, pages 5926–5936, 2019.
- [332] Eraldo Rezende Fernandes et al. Bert for stock market sentiment analysis. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1597–1601, 2019. doi: 10.1109/ICTAI.2019.00231.
- [333] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Portuguese named entity recognition using bert-crf, 2020. URL <https://arxiv.org/abs/1909.10649>.
- [334] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>.
- [335] Ieva Staliūnaitė and Ignacio Iacobacci. Compositional and lexical semantics in RoBERTa, BERT and DistilBERT: A case study on CoQA. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7046–7056, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.573. URL <https://aclanthology.org/2020.emnlp-main.573/>.

- [336] Harald Steck, Chaitanya Ekanadham, and Nathan Kallus. Is cosine-similarity of embeddings really about similarity? In *Companion Proceedings of the ACM Web Conference 2024*, volume 201 of *WWW '24*, page 887–890. ACM, May 2024. doi: 10.1145/3589335.3651526. URL <http://dx.doi.org/10.1145/3589335.3651526>.
- [337] Dusan Stosic and Paulius Micikevicius. What is a transformer model?, Jan 2021. URL <https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/>.
- [338] Wen Bo et al. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C), February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063. URL <https://doi.org/10.1016/j.neucom.2023.127063>.
- [339] Marc’Aurelio Ranzato et al. Multiple-attribute text style transfer, 2019. URL <https://arxiv.org/abs/1811.00552>.
- [340] Marc’Aurelio Ranzato et al. Multiple-attribute text style transfer, 2019. URL <https://arxiv.org/abs/1811.00552>.
- [341] Akhilesh Sudhakar, Bhargav Upadhyay, and Arjun Maheswaran. “transforming” delete, retrieve, generate approach for controlled text style transfer. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3269–3279, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1322. URL <https://aclanthology.org/D19-1322/>.
- [342] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2020. URL <https://arxiv.org/abs/1905.05583>.
- [343] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.
- [344] Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In Young Jin Kim et al., editors, *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustainLP)*, pages 78–90, Toronto, Canada (Hybrid), July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.sustainlp-1.5. URL <https://aclanthology.org/2023.sustainlp-1.5/>.
- [345] Lin Zhu et al. Revisiting color-event based tracking: A unified network, dataset, and metric, 2024. URL <https://arxiv.org/abs/2211.11010>.
- [346] Maha Elbayad et al. No language left behind: Scaling human-centered machine translation, 2022. URL <https://arxiv.org/abs/2207.04672>.

- [347] Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.
- [348] Jörg Tiedemann. The tatoeba translation challenge – realistic data sets for low resource and multilingual MT. In Marta R. Costa-jussà et al., editors, *Proceedings of the Fifth Conference on Machine Translation*, pages 1174–1182, Online, November 2020. Association for Computational Linguistics. URL <https://aclanthology.org/2020.wmt-1.139/>.
- [349] Jörg Tiedemann and Santhosh Thottingal. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal, 2020.
- [350] Alexey Tikhonov, Viacheslav Shibaev, Aleksander Nagaev, Aigul Nugmanova, and Ivan P. Yamshchikov. Style transfer for texts: Retrain, report errors, compare with rewrites. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/d19-1406. URL <http://dx.doi.org/10.18653/v1/D19-1406>.
- [351] Oleksandra Tmenova, Rémi Martin, and Luc Duong. CycleGAN for style transfer in x-ray angiography. *International Journal of Computer Assisted Radiology and Surgery*, 14(10):1785–1794, Oct 2019. ISSN 1861-6429. doi: 10.1007/s11548-019-02022-z. URL <https://doi.org/10.1007/s11548-019-02022-z>.
- [352] Martina Toshevskaja and Sonja Gievska. A review of text style transfer using deep learning. *IEEE Transactions on Artificial Intelligence*, 3(5):669–684, 2022. doi: 10.1109/TAI.2021.3115992.
- [353] Marie-Anne Lachaux et al. Llama: Open and efficient foundation language models, 2023. URL <https://arxiv.org/abs/2302.13971>.
- [354] Amjad Almahairi et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL <https://arxiv.org/abs/2307.09288>.
- [355] Barak Turovsky. Found in translation: More accurate, fluent sentences in google translate, Nov 2016. URL <https://blog.google/products/translate/found-in-translation-more-accurate-fluent-sentences-google-translate/>.
- [356] Masao Utiyama and Hitoshi Isahara. Reliable measures for aligning Japanese-English news articles and sentences. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 72–79, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075106. URL <https://aclanthology.org/P03-1010>.

- [357] Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. Llmzip: Lossless text compression using large language models, 2023. URL <https://arxiv.org/abs/2306.04050>.
- [358] Vanroy, Bram and Tezcan, Arda and Macken, Lieve. MATEO: MACHine Translation Evaluation Online. In Nurminen, Mary and Brenner, Judith and Koponen, Maarit and Latomaa, Sirkku and Mikhailov, Mikhail and Schierl, Frederike and Ranasinghe, Tharindu and Vanmassenhove, Eva and Alvarez Vidal, Sergi and Aranberri, Nora and Nunziatini, Mara and Parra Escartín, Carla and Forcada, Mikel and Popovic, Maja and Scarton, Carolina and Moniz, Helena, editor, *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 499–500. European Association for Machine Translation (EAMT), 2023. ISBN 978-952-03-2947-1. URL <https://lt3.ugent.be/mateo/>.
- [359] Llion Jones et al. Attention is all you need. In R. Fergus et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [360] M. K. Vathsala and Ganga Holi. Rnn based machine translation and transliteration for twitter data. *International Journal of Speech Technology*, 23(3):499–504, Sep 2020. ISSN 1572-8110. doi: 10.1007/s10772-020-09724-9. URL <https://doi.org/10.1007/s10772-020-09724-9>.
- [361] Prateek Verma and Julius O. Smith. Neural style transfer for audio spectrograms, 2018. URL <https://arxiv.org/abs/1801.01589>.
- [362] Karel Veselý, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Proc. Interspeech 2013*, pages 2345–2349, 2013. doi: 10.21437/Interspeech.2013-548.
- [363] MK Vijaymeena and K Kavitha. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28, 2016.
- [364] W3Techs. Usage statistics of compression for websites. <https://w3techs.com/technologies/details/ce-compression>, 2024.
- [365] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. <https://github.com/kingoflolz/mesh-transformer-jax>, May 2021.
- [366] Haifeng Wang, Hua Wu, Zhongjun He, Liang Huang, and Kenneth Ward Church. Progress in machine translation. *Engineering*, 18:143–153, 2022. ISSN 2095-8099. doi: <https://doi.org/10.1016/j.eng.2021.03.023>. URL <https://www.sciencedirect.com/science/article/pii/S2095809921002745>.
- [367] Jamie Wang. *Reimagining the More-Than-Human City: Stories from Singapore*. The MIT Press, 10 2024. ISBN 9780262381406. doi: 10.7551/mitpress/15189.001.0001. URL <https://doi.org/10.7551/mitpress/15189.001.0001>.

- [368] Jiapeng Wang and Yihong Dong. Measurement of text similarity: A survey. *Information*, 11(9), 2020. ISSN 2078-2489. doi: 10.3390/info11090421. URL <https://www.mdpi.com/2078-2489/11/9/421>.
- [369] Jing Wang, Haibo He, and Danil V. Prokhorov. A folded neural network autoencoder for dimensionality reduction. *Procedia Computer Science*, 13:120–127, 2012. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2012.09.120>. URL <https://www.sciencedirect.com/science/article/pii/S1877050912007272>. Proceedings of the International Neural Network Society Winter Conference (INNS-WC2012).
- [370] Jingyang Wang, Xiaolei Li, Jiazheng Li, Qiuhong Sun, and Haiyao Wang. Ngcu: A new rnn model for time-series data prediction. *Big Data Research*, 27:100296, 2022. ISSN 2214-5796. doi: <https://doi.org/10.1016/j.bdr.2021.100296>. URL <https://www.sciencedirect.com/science/article/pii/S2214579621001131>.
- [371] Li Wang, Lidan Wang, and Shubai Chen. Esa-cycleGAN: Edge feature and self-attention based cycle-consistent generative adversarial network for style transfer. *IET Image Processing*, 16(1):176–190, 2022.
- [372] Maolin Wang, Seyedramin Rasoulinezhad, Philip H. W. Leong, and Hayden K.-H. So. Niti: Training integer neural networks using integer-only arithmetic. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3249–3261, 2022. doi: 10.1109/TPDS.2022.3149787.
- [373] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.
- [374] Guoyin Wang et al. Topic-guided variational auto-encoder for text generation. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 166–177, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1015. URL <https://aclanthology.org/N19-1015>.
- [375] Xuan Wang, Lijun Sun, Abdellah Chehri, and Yongchao Song. A review of gan-based super-resolution reconstruction for optical remote sensing images. *Remote Sensing*, 15(20), 2023. ISSN 2072-4292. doi: 10.3390/rs15205062. URL <https://www.mdpi.com/2072-4292/15/20/5062>.
- [376] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2015.08.104>. URL <https://www.sciencedirect.com/science/article/pii/S0925231215017671>. RoLoD: Robust Local Descriptors for Computer Vision 2014.

- [377] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5878–5882, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1599. URL <https://aclanthology.org/D19-1599/>.
- [378] Noam Wies, Yoav Levine, Daniel Jannai, and Amnon Shashua. Which transformer architecture fits my data? a vocabulary bottleneck in self-attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11170–11181. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/wies21a.html>.
- [379] Wikipedia. 44,100 Hz — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/44,100_Hz, 2024. [Online; accessed 12-September-2024].
- [380] Wikipedia. Aragonese language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Aragonese%20language&oldid=1227287327>, 2024.
- [381] Wikipedia. Aranese dialect — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Aranese%20dialect&oldid=1225929158>, 2024.
- [382] Wikipedia. Assamese language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Assamese%20language&oldid=1227807788>, 2024.
- [383] Wikipedia. Asturian language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Asturian%20language&oldid=1226321926>, 2024. [Online; accessed 12-June-2024].
- [384] Wikipedia. BERT (language model) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=BERT%20\(language%20model\)&oldid=1212031987](http://en.wikipedia.org/w/index.php?title=BERT%20(language%20model)&oldid=1212031987), 2024. [Online; accessed 26-March-2024].
- [385] Wikipedia. Fonction de Rosenbrock — Wikipedia, the free encyclopedia. <http://fr.wikipedia.org/w/index.php?title=Fonction%20de%20Rosenbrock&oldid=206160278>, 2024.
- [386] Wikipedia. Khasi language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Khasi%20language&oldid=1228524180>, 2024.
- [387] Wikipedia. Meitei language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Meitei%20language&oldid=1227431826>, 2024.

- [388] Wikipedia. Mizo language — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Mizo%20language&oldid=1225127897>, 2024.
- [389] Wikipedia. Pareto principle — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pareto_principle&oldid=1240517401, 2024. [Online; accessed 21-September-2024].
- [390] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.
- [391] Clement Delangue et al. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL <https://aclanthology.org/2020.emnlp-demos.6/>.
- [392] Fei Wu. Overview of cyclegan architecture and training., Oct 2020. URL <https://towardsdatascience.com/overview-of-cyclegan-architecture-and-training-afee31612a2f>.
- [393] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation, 2020. URL <https://arxiv.org/abs/2004.09602>.
- [394] Minghao Wu, Thuy-Trang Vu, Lizhen Qu, George Foster, and Gholamreza Haffari. Adapting large language models for document-level machine translation, 2024. URL <https://arxiv.org/abs/2401.06468>.
- [395] Mohammad Norouzi et al. Google’s neural machine translation system: Bridging the gap between human and machine translation, 2016. URL <https://arxiv.org/abs/1609.08144>.
- [396] Lin Ma et al. Gan-based anomaly detection: A review. *Neurocomputing*, 493:497–535, 2022. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2021.12.093>. URL <https://www.sciencedirect.com/science/article/pii/S0925231221019482>.
- [397] Jianqiong Xiao and Zhiyong Zhou. Research progress of rnn language model. In *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 1285–1288, 2020. doi: 10.1109/ICAICA50127.2020.9182390.
- [398] Haoran Xu, Sixing Lu, Zhongkai Sun, Chengyuan Ma, and Chenlei Guo. VAE based text style transfer with pivot words enhancement learning. In Sivaji Bandyopadhyay, Sobha Lalitha Devi, and Pushpak Bhattacharyya, editors, *Proceedings of the 18th International Conference on Natural Language Processing (ICON)*, pages 162–172, National Institute of Technology Silchar, Silchar, India, December

2021. NLP Association of India (NLP AI). URL <https://aclanthology.org/2021.icon-main.20/>.
- [399] Xuancheng Ren et al. Unpaired sentiment-to-sentiment translation: A cycled reinforcement learning approach. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 979–988, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1090. URL <https://aclanthology.org/P18-1090>.
- [400] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. *Understanding and improving layer normalization*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [401] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. In E. Fox et al., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/254ed7d2de3b23ab10936522dd547b78-Paper.pdf.
- [402] Peng Xu, Jackie Chi Kit Cheung, and Yanshuai Cao. On variational learning of controllable representations for text without supervision. In *Proceedings of the 37th International Conference on Machine Learning, ICML’20*. JMLR.org, 2020.
- [403] Wenju Xu, Shawn Keshmiri, and Guanghui Wang. Adversarially approximated autoencoder for image generation and manipulation. *IEEE Transactions on Multimedia*, 21(9):2387–2396, 2019. doi: 10.1109/TMM.2019.2898777.
- [404] Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis. In M. Hardt et al., editors, *Advances in Neural Information Processing Systems*, volume 36, pages 59304–59322. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/b9e472cd579c83e2f6aa3459f46aac28-Paper-Conference.pdf.
- [405] Rami Al-Rfou et al. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL <https://aclanthology.org/2021.naacl-main.41>.
- [406] Yahoo! Yahoo! answers, 2011. URL <https://webscope.sandbox.yahoo.com/catalog.php?datatype=1>.
- [407] Yahoo! L6 - yahoo! answers comprehensive questions and answers version 1.0, 2011. URL <https://webscope.sandbox.yahoo.com/catalog.php?datatype=1>.
- [408] Eugene Yan. Some intuition on attention and the transformer, Aug 2023. URL <https://eugeneyan.com/writing/attention/>.

- [409] Shuqi Yan, Shaorong He, Xue Lei, Guanhua Ye, and Zhifeng Xie. Video face swap based on autoencoder generation network. In *2018 International Conference on Audio, Language and Image Processing (ICALIP)*, pages 103–108, 2018. doi: 10.1109/ICALIP.2018.8455775.
- [410] Wei Wu et al. SGM: Sequence generation model for multi-label classification. In Emily M. Bender, Leon Derczynski, and Pierre Isabelle, editors, *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3915–3926, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL <https://aclanthology.org/C18-1330>.
- [411] Yuan Xie et al. Training high-performance and large-scale deep neural networks with full 8-bit integers. *Neural Networks*, 125:70–82, 2020. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2019.12.027>. URL <https://www.sciencedirect.com/science/article/pii/S0893608019304290>.
- [412] Zichao Yang, Zhiting Hu, Chris Dyer, Eric P Xing, and Taylor Berg-Kirkpatrick. Unsupervised text style transfer using language models as discriminators. In N. Cesa-Bianchi et al., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/398475c83b47075e8897a083e97eb9f0-Paper.pdf.
- [413] Yelp. Yelp dataset challenge, 2015. URL http://www.yelp.com/dataset_challenge.
- [414] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI’17, page 2852–2858. AAAI Press, 2017.
- [415] Wennian Yu, Il Yong Kim, and Chris Mechefske. Analysis of different rnn autoencoder variants for time series classification and machine prognostics. *Mechanical Systems and Signal Processing*, 149:107322, 2021. ISSN 0888-3270. doi: <https://doi.org/10.1016/j.ymssp.2020.107322>. URL <https://www.sciencedirect.com/science/article/pii/S0888327020307081>.
- [416] Wei E. Zhang et al. Bert-coqac: Bert-based conversational question answering in context. In Li Ning, Vincent Chau, and Francis Lau, editors, *Parallel Architectures, Algorithms and Programming*, pages 47–57, Singapore, 2021. Springer Singapore. ISBN 978-981-16-0010-4.
- [417] Nasser Zalmout and Nizar Habash. Optimizing tokenization choice for machine translation across multiple target languages. *The Prague Bulletin of Mathematical Linguistics*, 108, 06 2017. doi: 10.1515/pralin-2017-0025.
- [418] Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. Autoencoder and its various variants. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419, 2018. doi: 10.1109/SMC.2018.00080.

- [419] Biao Zhang and Rico Sennrich. *Root mean square layer normalization*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [420] Biao Zhang, Deyi Xiong, Jun Xie, and Jinsong Su. Neural machine translation with gru-gated attention model. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4688–4698, 2020. doi: 10.1109/TNNLS.2019.2957276.
- [421] Dan Zhang and Anna Khoreva. Progressive augmentation of gans. In E. Fox et al., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/2e2c080d5490760af59d0baf5acbb84e-Paper.pdf.
- [422] JiaJun Zhang and ChengQing Zong. Neural machine translation: Challenges, progress and future. *Science China Technological Sciences*, 63(10):2028–2050, Oct 2020. ISSN 1869-1900. doi: 10.1007/s11431-020-1632-x. URL <https://doi.org/10.1007/s11431-020-1632-x>.
- [423] Linlin Zhang, Zhirui Zhang, Boxing Chen, Weihua Luo, and Luo Si. Context-adaptive document-level neural machine translation. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6232–6236, 2022. doi: 10.1109/ICASSP43922.2022.9746245.
- [424] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020. URL <https://arxiv.org/abs/1904.09675>.
- [425] Xiang Zhang and Yann LeCun. Text understanding from scratch, 2016. URL <https://arxiv.org/abs/1502.01710>.
- [426] Yizhe Zhang, Zhe Gan, and Lawrence Carin. Generating text via adversarial training. In *NIPS workshop on Adversarial Training*, volume 21, pages 21–32. Academia. edu, 2016.
- [427] Ricardo Henao et al. Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17, page 4006–4015. JMLR.org, 2017.
- [428] Hongtao Lu et al. Spatio-temporal autoencoder for video anomaly detection. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM ’17, page 1933–1941, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349062. doi: 10.1145/3123266.3123451. URL <https://doi.org/10.1145/3123266.3123451>.
- [429] AoJun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights, 2017. URL <https://arxiv.org/abs/1702.03044>.
- [430] Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, page 665–674, New York,

- NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098052. URL <https://doi.org/10.1145/3097983.3098052>.
- [431] Kaitlyn Zhou, Kawin Ethayarajh, Dallas Card, and Dan Jurafsky. Problems with cosine as a measure of embedding similarity for high frequency words. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 401–423, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.45. URL <https://aclanthology.org/2022.acl-short.45/>.
- [432] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017. doi: 10.1109/ICCV.2017.244.
- [433] Shujian Huang et al. Multilingual machine translation with large language models: Empirical results and analysis. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.176. URL <https://aclanthology.org/2024.findings-naacl.176/>.
- [434] Zhiyu Zhu, Junhui Hou, and Dapeng Oliver Wu. Cross-modal Orthogonal High-rank Augmentation for RGB-Event Transformer-trackers . In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 21988–21998, Los Alamitos, CA, USA, October 2023. IEEE Computer Society. doi: 10.1109/ICCV51070.2023.02015. URL <https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.02015>.
- [435] Aniketh Reddy Zihao Ding and Aparna Joshi. 5 – reproducibility. <https://blog.ml.cmu.edu/2020/08/31/5-reproducibility/>, 2020.
- [436] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi: 10.1109/TIT.1977.1055714.
- [437] Zhuofan Zong, Guanglu Song, and Yu Liu. DETRs with Collaborative Hybrid Assignments Training . In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6725–6735, Los Alamitos, CA, USA, October 2023. IEEE Computer Society. doi: 10.1109/ICCV51070.2023.00621. URL <https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.00621>.