Vrije Universiteit Amsterdam

Bachelor Thesis

Enhanced Sequence-to-Sequence Machine Text Translation: Investigating the Benefits of Attention mechanism and Attention mechanism with GRUs

Author: D.A. Milikina

July 3, 2023

1st supervisor: Peter Bloem
2nd reader: Ruud van Bakel

Abstract

This paper investigates the use of Recurrent Neural Network (RNN), where the baseline is the attention mechanism in sequence-to-sequence machine text translation from the source language English to the target language French. Two transformers consisting of an encoder-decoder architecture are proposed where one transformer incorporates a pure attention mechanism, and another consists of an attention mechanism combined with Gated Recurrent Units (GRUs). The transformer that uses pure attention draws inspiration from the paper "Attention Is All You Need" (Vaswani et al.). The main difference between this paper and "Attention Is All You Need" (Vaswani et al.) is that in the latter, teacher forcing techniques are applied. Furthermore, an evaluation of translation performance using the BiLingual Evaluation Understudy (BLEU) score is performed. The experiments reveal that both transformers exhibit improved performance with a relatively low learning rate, specifically 0.0001, and deeper encoder and decoder blocks. The depth of an encoder or decoder block refers to the number of times the block is repeated within the overall architecture of a transformer. Moreover, the estimated BLEU score suggests that for the experiment duration, the model could not generate meaningful French translations based on the provided English sentences, no matter that the accuracy was around 40%. We concluded that the amount of padding highly influences the accuracy of the sentences rather than the words. Furthermore, we performed statistical tests, and we rejected the hypothesis that there is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism combined with GRUs after training for 100 epochs. In addition, the analysis did not provide enough evidence to reject the hypothesis that there is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism with GRUs when deeper encoder-decoder blocks are implemented.

Keywords: transformer, encoder, decoder, text translation, sequence-to-sequence, attention, BLEU score

1 Introduction

Text translation is a fundamental task in natural language processing (NLP), enabling effective communication across linguistic barriers. With the advent of deep learning and neural network architectures, sequence-to-sequence encoder-decoder architecture has emerged as a powerful mechanism for machine translation tasks. These models have demonstrated remarkable success in capturing complex linguistic patterns and generating high-quality translations (Vaswani et al.; Luong et al.; Jozefowicz et al.).

Central to the sequence-to-sequence encoder-decoder architecture is incorporating a transformer that relies mostly on an attention mechanism to draw dependencies. Attention mechanisms address this challenge by allowing the model to focus on relevant parts of the input during the encoding and decoding phases(Vaswani et al.). Therefore, the model can capture relationships between source and target words by attention to different parts of the input sequence. In this encoder-decoder architecture, the encoder processes the input consisting of the source language, English sentences, while the decoder utilises this representation to generate the corresponding target language, French translation (Y. Wu et al.). Combined, the two components facilitate text translation from one language to another, enabling the model to translate English sentences into meaningful and accurate French translations.

Undertaking a machine text translation project offers both academic and practical benefits. It allows for intellectual exploration in natural language processing and computational linguistics while addressing the practical need for effective communication across languages in an interconnected world. Furthermore, machine text translation could have societal implications, facilitating cultural exchange, inclusivity, and cross-domain applications such as international business and diplomacy.

This paper aims to investigate the performance and effectiveness of two different approaches for text translation: pure attention mechanism and attention mechanism combined with Gated Recurrent Units (GRUs) (Cho et al.; Chung et al.).

The main difference between a pure attention mechanism and an attention mechanism with GRUs lies in how they handle sequential data's temporal aspect. In a pure attention mechanism, attention is applied directly to the input sequence or sequence of hidden states without additional recurrent layers. It independently computes attention weights for each element in the sequence based on their relevance to the current context. These attention weights are then used to compute a weighted sum of the input sequence, functioning as a context vector for further processing.

The GRU is a type of Recurrent Neural Network (RNN) that introduces a gating mechanism to control the flow of information through the recurrent connections. The attention mechanism is applied to the hidden states of the GRU, allowing the model to focus on different parts of the input sequence while considering the temporal dependencies. Temporal dependencies refer to the ability of the model to capture and understand the order of words or tokens in a sentence or document. By incorporating an attention mechanism to model temporal dependencies, transformers have the potential to effectively address word order and context within a sentence, possibly leading to enhanced translation output quality. Moreover, GRU layers were employed in models or architectures before the introduction of the attention mechanism. It implies that GRUs were used as the primary recurrent layer for capturing sequential information and modelling dependencies in the data before incorporating attention mechanisms.

By exploring these two approaches and their respective strengths and weaknesses, the study can provide insights into the effectiveness of different models for text translation tasks. We performed extensive experimentation on well-established translation dataset, employing quantitative metrics such as the BLEU score to evaluate the models' performance. Additionally, we performed qualitative analysis by examining sample translations to gain insights into the translation quality and the behaviour of attention mechanisms.

Furthermore, we discuss the computational complexity and training dynamics of each approach. While attention mechanisms enhance translation quality, they also introduce additional computational requirements during training and inference. These considerations are examined,

providing insights into the trade-offs between translation performance and computational resources.
Moreover, the comparative performance between both models is evaluated for a training of 100 epochs of approximately 2 hours.
The contributions of this paper include a comprehensive evaluation of both transformers. Ultimately, the study enhances the comprehension and application of attention mechanisms and GRUs in the domain of sequence-to-sequence machine text translation.

1.1 Research Questions

What is the comparative performance of a transformer with a pure attention mechanism versus a transformer with an attention mechanism combined with Gated Recurrent Units (GRUs) in text translation tasks when trained for 100 epochs?

H0: There is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism combined with GRUs in the text translation task after training for 100 epochs.
HA: There is a significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism combined with GRUs in the text translation task after training for 100 epochs.

Do deeper encoder-decoder blocks affect the performance of the transformers?

The depth of an encoder or decoder block refers to the number of times the block is repeated within the overall architecture of a transformer.

H0: There is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism with GRUs when deeper encoder-decoder blocks are implemented.
HA: There is a significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism with GRUs when deeper encoder-decoder blocks are implemented.

2 Model Architecture

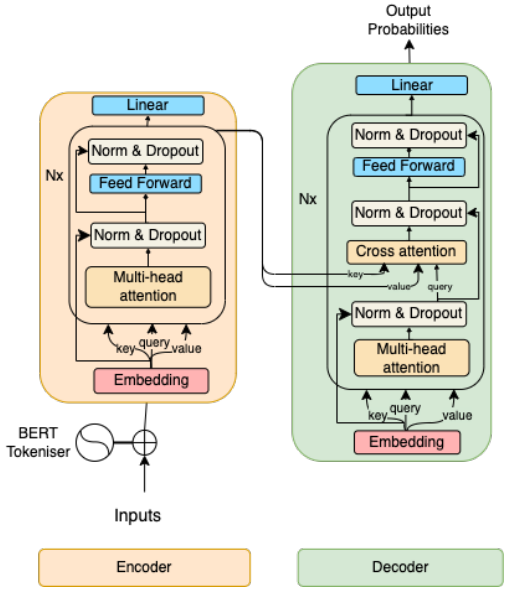

Figure 1: Transformer: Pure attention

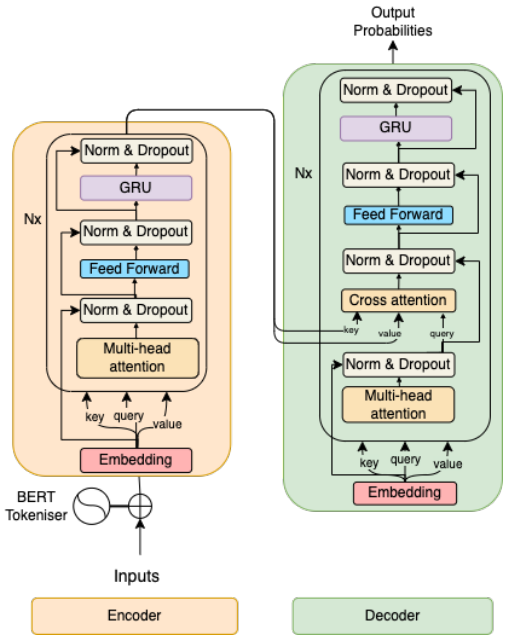

Figure 2: Transformer: Attention combined with GRUs

2.1 Pre-processing

The HuggingFace pre-trained Bidirectional Encoder Representation from Transformers (BERT) (Devlin et al.; HuggingFace, Bert Tokenizer) tokeniser is used to convert strings to a sequence of IDs (integers) both in the source language English and the target language French, which is an essential step in preparing text data for input to the translation model. The BERT tokeniser breaks down the raw strings into units called tokens, and the tokens are mapped to an embedding vector which is fed into the model.
Moreover, unique token IDs 101 and 102, are added to mark each sentence's start and end, respectively. The English-French pairs are ordered ascendingly based on the number of tokens in the English part of the sequence. Custom padding is applied based on the longest sequence in the batch, taking into consideration both languages. The applied padding ensures that all input sequences are of equal length in the current batch, which is necessary for efficient batch processing.
The BERT base uncased tokeniser (BERT Base Uncased), pre-trained on massive amounts of text data, converts the strings to a sequence of IDs in the English sequences. In contrast, the BERT base multilingual uncased pre-trained tokeniser (Bert Base Multilingual) converts the French sequences. Since they are uncased, the pre-trained tokenisers will not make a distinction between, for example, 'english' and 'English'.
The IDs of the English text are then passed through a translation model, which is trained to generate the corresponding French translation. Two BERT tokenisers from two BERT pre-trained tokenisers were used since we observed that French linguistic nuances were lost when the BERT base uncased tokeniser was used to convert the strings to a sequence of IDs in both languages. This resulted in identical token IDs that led to false positives causing higher accuracy on both training and validation accuracy. We speculate that this was caused by the fact that the BERT tokeniser was pre-trained on English data.

2.2 Attention mechanism

An attention function is a mechanism that takes a query and a set of key-value pairs as input and produces an output. In this mechanism, the query, keys, values, and output are all represented as vectors. The output is computed by calculating a weighted sum of the values, where the weight assigned to each value is determined by a compatibility function that compares the query with the corresponding key.

2.3 Multi-head attention layer

The multi-head attention layer employs parallel attention heads (queries, keys, values), each responsible for attending to different aspects of the input representation. It allows the model to jointly attend to information from different representation subspaces at various positions. Moreover, the model can capture diverse semantic and contextual information from the English input by utilising three different attention heads, enhancing its ability to generate accurate and comprehensive French translations.

2.4 Normalisation

A normalisation layer (J. L. Ba et al.) helps to normalise and standardise the activations of a neural network layer along the feature dimension since the outputs are scaled and shifted to have consistent mean and variance properties. This helps stabilise the training process and allows different parts of the model to interact effectively. Moreover, it improves the gradient flow during backpropagation and facilitates faster convergence of the model. By reducing internal covariate shifts, normalisation can also make the training process more robust and allow for higher learning rates. The normalisation layer after the multi-head attention layer ensures that the representations produced are consistent.

2.5 Feed forward network

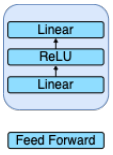

Figure 3: Feed forward architecture

The feed forward network in the encoder-decoder architecture consists of two linear layers and one Rectified Linear Unit (ReLU) activation function in between. This architecture facilitates transforming and processing input data to

generate meaningful representations.
In the encoder, the network starts with a linear layer that performs a transformation on the input data. This layer applies a set of weights to each input feature and produces a new set of transformed features. The output of the linear layer is then passed through a Rectified Linear Unit (ReLU) activation function, which introduces non-linearity to the network (Agarap). The ReLU activation function sets negative values to zero and keeps positive values unchanged. After the ReLU activation, another linear layer is applied, further transforming the features and preparing them for subsequent processing.
In the decoder, the network follows a similar process where the final linear layer is applied to produce the output, generating the desired decoded representation.
By combining linear and ReLU activation in the encoder and decoder, the feed forward network can effectively process and transform the input data, extracting and mapping relevant information to generate meaningful representations and facilitate accurate decoding in the decoder. In both the encoder and decoder, the structure follows the same order.
The feed forward network can be represented as:

$$FFN(x) = ReLU(xW_1 + b_1) \cdot W_2 + b_2$$

where x represents the input, W_1 is a weight matrix of size $(embedding\,dimension) \times (embedding\,dimension \times 4)$, b_1 is a bias vector of size $(embedding\,dimension \times 4)$, $ReLU$ denotes the rectified linear unit activation function applied element-wise, W_2 is a weight matrix of size $(embedding\,dimension) \times (embedding\,dimension \times 4)$, and b_2 is a bias vector of size $(embedding\,dimension)$. This formula represents the sequential operations in the feed forward network, where the input x is linearly transformed by W_1 and b_1, passed through the ReLU activation function, and then linearly transformed again by W_2 and b_2 to produce the network's final output.

2.6 Optimiser

2.6.1 Adam

The Adam optimiser (Kingma and J. Ba) is used to optimise the parameters of both the encoder and decoder components by minimising the loss between the predicted and ground truth French translations.
Adam's optimiser key idea is to enhance the efficiency of converging to the minimum cost function by combining momentum and adaptive learning rates. Throughout the training process, Adam dynamically adjusts the learning rate based on the first and second moments of the gradients. The first moment refers to the gradient mean, while the second moment represents the gradient variance. Moreover, Adam keeps track of exponentially decaying averages of the past gradients and their squares, utilising them to update the parameters in each iteration. This approach enables Adam to achieve faster and more efficient convergence to the cost function's minimum compared to traditional gradient descent methods. Additionally, Adam incorporates a bias correction mechanism to ensure that the initial moment estimates are close to zero.
The primary hyperparameters of Adam include the learning rate, beta1 (the exponential decay rate for the first moment estimate), and beta2 (the exponential decay rate for the second moment estimate).
By setting the learning rate to 0.001, we control the step size at each iteration during gradient descent.

2.7 Regularisation

2.7.1 Residual dropout

Dropout (Hinton et al.) is a regularisation technique widely used in neural networks. It helps prevent overfitting by randomly "dropping out" (setting to zero) a fraction of the neurons during training. When applied after an attention layer, dropout masks certain attention weights and encourages the model to rely on a diverse set of attention patterns rather than a few dominant ones.
Dropout can be particularly effective in models with a large number of parameters, such as transformers.
For all the models, the probability of an element to be zeroed is *0.1*

2.8 Transformer: Encoder and Decoder architecture

The encoder-decoder architecture in this paper generates each token without relying on teacher forcing. In contrast, the encoder-decoder architecture in the paper (Vaswani et al.) operates in an auto-regressive manner, incorporating the previously generated output as additional input at each step when generating the next symbol.

2.9 Encoder

The primary objective of an encoder is to capture and encode the essential information contained in the input sequence, in this case, the English sentences, preserving its semantic and syntactic context.
An encoder can be represented as a function that takes a variable-length input sequence $(x_1, x_2, \ldots, x_n)$ where n is the length of the sequence and maps it to a sequence of continuous representations $(h_1, h_2, \ldots, h_m)$ where m is the dimensionality of the encoded representation (Graves).
During the encoding process, the encoder model processes the input sequence and transforms it into fixed-length vectors. Moreover, the encoder learns to extract relevant features from the input sequence, in this case, the English sentences, taking into consideration the context and relationships between the tokens.
The encoded representation generated by the encoder is then passed on to the decoder component, specifically to the cross-attention layer in the decoder model (Graves). The decoder utilises this representation to generate the corresponding French translation.

2.9.1 Encoder: Pure attention mechanism

The encoder architecture incorporates an embedding layer that maps input tokens to continuous representations. It is followed by a block incorporating various layers. Moreover, each block can be repeated N times. Each block is comprised of a multi-head attention mechanism, normalisation and dropout layers and a feed forward network. In the multi-head attention layer, all of the keys, values and queries come from the previous layer, in this case, the embedding layer.
Furthermore, this attention mechanism allows each position in the encoder to attend to all positions in the previous layer of the encoder. This way, the encoder can capture contextual relationships and dependencies between tokens.
The feed forward network applies non-linear transformations to the outputs of the attention mechanism, enabling the model to capture higher-level features and interactions between tokens (For more information about the network, see section Feed forward network). Finally, the encoded representation is passed through a final linear layer, which maps it to the desired output dimensions and ensures compatibility with the decoder.
The design of the encoder allows it to effectively capture contextual information, create stable representations and transform the encoded representation to a suitable format for subsequent processing in the decoder.
The architecture of the encoder is shown in Figure Transformer: Pure attention

2.9.2 Encoder: Attention mechanism with GRUs

This encoder architecture is based on the encoder with pure attention; however, in the block of stacked layers, after the feed forward construction (For more information about the network, see section Feed forward network), it incorporates one GRU layer, whose output is fed to a final normalisation and dropout layer.
The GRU layer, first introduced in (Cho et al.), a type of RNN, is used to model temporal dependencies and capture sequential information within the encoded representations (Chung et al.). This layer consists of recurrent units that maintain hidden states, which store information from previous time steps. Including the GRU layer in the encoder architecture allows for modelling sequential information and capturing dependencies over time. This enhances the encoder's ability to process and encode input sequences effectively, which we expect to lead to improved performance in the machine text translation task.
Finally, the encoded representation from the GRU layer is passed through a final linear layer, which maps it to the desired output dimensions and ensures compatibility with the decoder.
The architecture of the encoder is shown in Figure

Transformer: Attention combined with GRUs

2.10 Decoder

The decoder model generates French sentences based on the encoded representation passed from the encoder. The decoder can be defined as a function that takes the encoded representation $(h_1, h_2, \ldots, h_m)$ and maps it to an output of French sentences $(y_1, y_2, \ldots, y_k)$ where k is the length of the sentence.

During the decoding process, the decoder utilises the decoded representation and its internal state to generate each token of the output French sentence. It considers the dependencies and relationships captured in the encoded representation to produce a coherent and accurate translation.

The output of the decoder is a translated French sentence, which is expected to accurately represent the meaning and structure of the original English sentence.

2.10.1 Decoder: Pure attention mechanism

The decoder starts with an embedding layer that maps the input tokens to continuous representations.

Following the embedding layer, the decoder comprises a block with stacked layers. Each block consists of several layers arranged in a specific order, and the block can be repeated N times. One block consists of a multi-head attention layer, normalisation and dropout layers, a cross-attention layer, additional normalisation and dropout layers, and a feed forward network (For more information about the network, see section Feed forward network).

The multi-head attention layer is responsible for capturing dependencies within the input sequence. It allows each position in the decoder to attend to all positions in the previous layer, utilising keys, values, and queries obtained from the embedding layer. The decoder can extract relevant information and improve the representation by attending to different parts of the input sequence. The output of the attention layer is then added to the original input, enhancing the representation with the relevant information.

After the attention layer, normalisation (For more information about the layer, see section Normalisation) and dropout layers are applied (For more information about the layer, see section Residual dropout).

The cross-attention layer in the decoder is responsible for introducing information from the encoder, specifically, the keys and values derived from the English sequence input. This cross-attention mechanism lets the decoder align the generated output with the input sequence. It allows each position in the decoder to attend to all positions in the input sequence, facilitating the incorporation of relevant information during the decoding process. Similar to previous layers, normalisation and dropout layers are applied after the cross-attention layer.

Following cross-attention, the decoder employs a feed forward network and the decoded representation is passed through a final linear layer, which maps it to the desired output dimensions.

The overall architecture of the decoder enables it to attend to the input sequence, align the output with the input through cross-attention, and capture higher-level features through the feed forward network.

By combining these components in a stacked manner, the decoder is expected to generate accurate and contextually relevant output sequences based on the encoded representations from the encoder.

The architecture of the decoder is shown in Figure Transformer: Pure attention

2.10.2 Decoder: Attention mechanism with GRUs

This decoder architecture is based on the decoder with pure attention. In addition to the pure attention mechanism, it incorporates one GRU layer and one linear layer. The GRU layer receives the output from the feed forward network and processes it sequentially, capturing temporal dependencies and allowing the model to consider the order of the input sequence. Normalisation and dropout are applied after the GRU layer. Finally, the decoded representation is passed through a final linear layer, which maps it to the desired output dimensions.

The addition of the GRU layer in the blocks introduces sequential processing, allowing the decoder to capture temporal dependencies in the input sequence and considers the order of

the input sequence. This sequential processing enables the decoder to better model the dependencies between tokens in the sequence and capture the context of each token in relation to its neighbouring tokens. In contrast, in the decoder with a pure attention mechanism, the decoder operates purely in a self-attention manner, attending to different positions within its own input without considering the order of the input sequence. By incorporating the GRU layer, the overall architecture of the decoder could become more capable of capturing and generating sequences that exhibit sequential patterns and temporal coherence.
The architecture of the decoder is shown in Figure Transformer: Attention combined with GRUs

3 Experimental Setup

3.1 Data

We use a dataset explicitly curated for text translation tasks comprising 8.41 GB of English-French sentence pairs obtained from Kaggle (DAVE). However, only 801 sentence pairs of the entire dataset were used since there were memory constraints.
The project is implemented in Python using PyTorch(PyTorch Foundation), and the training of the transformers was performed on an NVIDIA GeForce RTX 2060.

3.2 Setup

In this experimental setup, the modified parameters were the learning rate and the depth of the encoder and decoder blocks, whereas all others were kept constants.
On the one hand, using various learning rates allows for an exploration of different convergence rates and speed at which the model learns. Moreover, using a high learning rate may lead to faster convergence, but there is a risk of overshooting a potential optimal solution. In contrast, a low learning rate may result in slow convergence or getting stuck in suboptimal solutions. By experimenting with different learning rates, the optimal balance between convergence speed and accuracy can be investigated; therefore, for the experiments in this paper, two values for learning rates were used, namely, 0.001 and 0.0001.
On the other hand, modifying the depth of the encoder and decoder blocks allows for controlling the model's capacity and complexity. A deeper model can potentially capture more intricate patterns and representations but might be prone to overfitting if the dataset is too small. Furthermore, a more shallow model may be simpler and more computationally efficient but might struggle to learn complex relationships, language patterns and nuances. An investigation of the trade-off between model complexity and generalisation performance is conducted by adjusting the depth.
Two ratios between encoder and decoder depth were examined in this experimental setup. The ratio of encoder:decoder 1:1 and the ratio of encoder:decoder 2:4.
It was decided to have a deeper decoder than an encoder to create a model with more layers to unfold and expand the encoded representation, allowing it to potentially capture and retain more fine-grained information during the decoding process. Another reason was that a deeper decoder increases the model's expressive power, enabling it to capture more complex relationships and dependencies between words in the target language (in this case, French).
Moreover, during decoding, errors or inaccuracies in the generated output can accumulate and propagate from one time step to the next (L. Wu et al.; Hassan et al.; Zhang et al.). Implementing a deeper decoder provides more error correction and refinement opportunities as the model generates subsequent words in the translated sentence. This can help mitigate error propagation and improve overall translation quality.
By manipulating these parameters, their effects can be isolated, and their contributions to the model's performance and overall learning process can be examined.

Both transformers for the machine text translation task used a baseline model configuration.
Training ratio: 0.6
Validation ratio: 0.2
Testing ratio: 0.2
Number of epochs: 100
Batch size: 32
Number of heads: 4
Embedding dimensions for all layers: 128

Dropout probability: 0.1
Total number of sentences used: 801

In the encoders, the initial tensor is of size *(A, B)*, where **A** is the batch size and **B** is the length of longest sequence per batch. It is fed to the embedding layer resulting in a tensor of size *(A, B, C)*, where **C** is the embedding dimension. All layers keep the tensor size *(A, B, C)*, which is returned at the end of each iteration.
For example, the initial tensor is of size (32, 19) transformed to (32, 19, 128) in the embedding layer, and the exact tensor sizes are kept throughout the layers.
In the decoders, the initial tensor is a zero tensor of size equal to the size of the encoder output *(A, B)*. It is fed to the embedding layer resulting in a tensor of size *(A, B, C)*. All layers keep the tensor size *(A, B, C)*. However, at the end of each iteration, the decoder output is transformed to *(A, B, D)*, where **D** is the vocabulary size of BERT base multilingual uncased + 3. The three parameters are added to represent the added padding, start and end of the sentence tokens.

Note: Because of resource constraints, the experiments with regard to the depth of the encoders and decoders and the total number of sentences were limited.

4 Evaluation

4.1 BLEU score

The BiLingual Evaluation Understudy (BLEU) is a commonly used metric in machine translation tasks to evaluate the quality of generated or translated text. It measures the similarity between the generated output and one or more reference translations based on n-gram precision. The BLEU score calculates precision by comparing n-gram sequences (contiguous sequences of n words) between the generated output and the reference translations.
The score ranges from 0 to 1, with 1 indicating a perfect match with the reference translations.
However, since exact matches are rare in translation tasks, a higher BLEU score usually signifies better translation quality. Nevertheless, it is essential to interpret the BLEU score cautiously, as it primarily measures the lexical overlap and does not capture semantic or grammatical aspects of translation quality. The BLEU score is widely adopted due to its simplicity and computational efficiency (Cho et al.). However, it has certain limitations, such as being sensitive to sentence length and not fully capturing the nuances of translation quality (Liu et al.).

4.2 Results for 100 epochs

Transformer Type	LR	Depth encoder:decoder	T Accuracy	V Accuracy	T Loss	V Loss
Pure Attention	0.001	2:4	0.388	0.360	0.002	0.006
		1:1	0.385	0.354	0.002	0.006
	0.0001	**2:4**	0.4	0.37	0.003	0.005
		1:1	0.386	0.366	0.003	0.005
Attention with GRU layers	0.001	2:4	0.385	0.367	0.002	0.006
		1:1	0.386	0.364	0.002	0.006
	0.0001	**2:4**	0.394	0.368	0.003	0.005
		1:1	0.389	0.368	0.0027	0.0047

Table 1: Results using various hyperparameters

The training and validation accuracy for the transformers with pure attention mechanism and the transformer with attention mechanism combined with GRU layers is around 40% for the machine text translation task. This accuracy could be considered relatively low, indicating that the models cannot generate robust translations. Based on the observations, both transformers exhibit signs of overfitting, using a learning rate of 0.001 and 100 epochs. This is evident from the lower training loss compared to the validation loss, regardless of the depth of the encoder and decoder blocks.
On the other hand, when the models employ a learning rate of 0.0001, they show relatively better convergence and performance. The

transformers with a learning rate of 0.0001, demonstrate improved performance compared to the counterpart using a learning rate of 0.001. However, the insignificant difference between training loss and validation loss indicates the possibility of overfitting.
Furthermore, the transformer with pure attention with a learning rate of 0.0001 and depth enc:dec of 1:1 and the transformer with pure attention and GRUs with a learning rate of 0.0001 and depth of enc:dec 1:1 and 2:4 seem to plateau around the 50th epoch (See the Figures in the Appendix). This could point out that the models get stuck or they cannot escape local minima which is a sign that fine-tuning of the learning rate and potentially other parameters have to be performed.
Moreover, when the model has a depth of 2 for the encoder and 4 for the decoder, along with a learning rate of 0.0001, the training and validation accuracy improves, while the training and validation losses remain the same.
However, the transformer with pure attention, learning rate of 0.0001 and depth of enc:dec of 2:4 was not powerful enough to correctly predict the true translations, which is shown in the Figures in the Appendix.
In summary, using a lower learning rate and adjusting the depth of the model's encoder and decoder blocks can mitigate overfitting and enhance performance. Moreover, a learning rate that is too high can lead to unstable training or cause the model to overshoot a possible optimal solution. This may result in the model failing to converge or bounce around an optimal point without effectively improving performance. Conversely, a learning rate that is too low can result in slow convergence or the model getting stuck in a suboptimal solution. Training may take longer, and the model may struggle to reach its full potential. However, balancing model complexity and available resources is vital to ensure efficient and effective training.
For better visualisation of the accuracy and loss of the transformer with pure attention mechanism, see the Appendix, the transformer with attention mechanism and GRU layers, see the Appendix, encoded and decoded the sentences from the dataset can be depicted in the Figures in the Appendix).

The BLEU score in this paper is measured per batch using the corpus_bleu function from the NLTK library (NLTK). It calculates a single corpus-level BLEU score for the generated translation sentences and their ground true translations from the dataset.
The way the BLEU score is calculated is not by averaging the sentence level score but by using the original BLEU metric (Papineni et al.), which accounts for the micro-average precision(i.e. summing the numerators and denominators for each predicted-true sentence pairs before the division).
The BLEU score observed for all experiments was extremely low, which was reported as a 0 in Weights and Biases (Weights and Biases) used for experiment tracking. This score suggests that the generated text has no similarity or overlap with the ground true French sentences from the dataset.
For better visualisation of the BLEU score of the transformer with pure attention, see the Appendix, transformer with attention combined with GRU layers, see the Appendix

To sum up, when looking at the accuracy of 40% together with a BLEU score of 0, the conclusion is that the accuracy metric is highly influenced by the padding tokens rather than the words. This could be because the padding is added based on the length of the longest sequence, which could lead to a sentence with, for example, 2 words and 30 padding tokens. To enhance the model's performance, further experiments can be performed by adjusting the sizes of batches to prevent having sentences with many padding tokens. Extending the model's complexity, which may involve adding more layers or modifying the architecture. This allows for examining different configurations to find the optimal setup for the task.
In addition to adjusting the model's complexity, hyperparameter tuning should be considered. Hyperparameters, such as learning rate, batch size, corpus size, or regularisation techniques, are crucial in determining the model's performance. Fine-tuning these hyperparameters through systematic experimentation can lead to improved results.
By iteratively refining the model's architecture, exploring various hyperparameter settings, and assessing the impact on performance, it is possible to find the best configuration that maximises

the model's potential and enhances its overall performance.

4.3 Hypothesis testing

What is the comparative performance of a transformer with a pure attention mechanism versus a transformer with an attention mechanism combined with Gated Recurrent Units (GRUs) in text translation tasks when trained for 100 epochs?
In order to determine if the null hypothesis can be rejected, a two-tailed t-test was performed based on the validation accuracy. Furthermore, the standard deviation and independent samples t-test were performed using the 95% confidence level.
It was concluded that since the calculated t-value falls outside the critical t-value, we reject the ***H0:*** *There is no significant difference in the performance of a transformer with pure attention mechanism and a transformer with attention mechanism combined with GRUs in the text translation task after training for 100 epochs.*
(See the calculations in the Appendix)

Do deeper encoder-decoder blocks affect the performance of the transformers?
The validation accuracies of both types of depth of enc:dec architecture were taken into account. A two-tailed t-test with a 95% confidence level was performed, pointing out that based on the results, we fail to reject the ***H0:*** *There is no significant difference in the performance of a transformer with pure attention mechanism and a transformer with attention mechanism with GRUs when deeper encoder-decoder blocks are implemented.* (See the calculations in the Appendix)

5 Conclusion and Limitations

This paper examined two transformer models for machine text translation and for evaluation of their performance accuracy, and BLEU scores were used.
The findings show that a trade-off between accuracy and model complexity has to be made depending on the available resources. The results from the experiments showed that for both architectures, using a lower learning rate leads to better convergence since the learning rate plays a crucial role in controlling the speed and stability of the model's learning process.
Using statistical methods, it can be concluded that we reject the null hypothesis, which states that there there is no significant difference between the performance of the reviewed transformers for a machine text translation task for 100 epochs. Furthermore, the results indicate that we fail to reject that there is no significant difference between having only one block in the encoder and one block in the decoder and two blocks in the encoder and four blocks in the decoder.
We encountered many limitations due to memory constraints. They were caused by insufficient Random-access memory (RAM) capacity on the machine during training which impacted the batch size and the complexity of the transformers. Another limitation was the amount of padding applied per sequence since it was based on the longest sentence in the batch. This also had an impact on memory usage during training.
The code implementation for this paper can be accessed on GitHub at the following link: GitHub repository

6 Future research

It could be beneficial to explore a combination of deeper encoder and decoder blocks with extended training cycles. In addition, hyperparameter tuning could increase the model's expressive power, improving translation quality.
Another area that can be explored is whether increasing the number of heads in an encoder-decoder architecture improves the attention mechanism. Adding more heads would generally enable the model to expand the attention scope further and help the model generalise better by capturing a wide range of input patterns.
Also, fine-tuning the hyperparameters and introducing teacher forcing can be beneficial. Teacher forcing is an algorithm for training the weights of recurrent neural networks (RNNs). Its approach involves providing the observed sequence values (also known as ground-truth samples) as an input to the RNN at each step, thereby forcing the RNN to stay as close as possible to the ground-truth sequence. We can feed the ground-truth output drawn from the training dataset as input during training, whereas the true output would be unknown during testing.

Therefore, we can approximate the correct output with the model's output and feed the output back to the model.
Without the teacher forcing the output of the previous time step acts as an input during RNN training. The RNN learns to generate the next output in a sequence based on the previous value. This approach is intended to help the model learn the correct sequence of output tokens more efficiently and stabilise the training process.

Teacher forcing was not implemented due to the time constraints of the thesis.

6.1 Acknowledgements

I would like to express my deep gratitude to my supervisor, *Peter Bloem*, for his guidance, inspiration and unwavering support throughout the thesis. Furthermore, I would like to thank *Ruud van Bakel* for reviewing my thesis as a second reader.

References

Agarap, Abien Fred. Deep Learning using Rectified Linear Units (ReLU). 2019. *arXiv*, arxiv.org/abs/1803.08375.

Ba, Jimmy Lei, et al. Layer Normalization. 2016. *arXiv*, arxiv.org/abs/1607.06450.

Cho, Kyunghyun, et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 2014. *arXiv*, arxiv.org/abs/1406.1078.

Cho, Kyunghyun, et al. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 2014. *arXiv*, arxiv.org/abs/1409.1259.

Chung, Junyoung, et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014. *arXiv*, arxiv.org/abs/1412.3555.

DAVE, DHRUVIL. "English-French Translation Dataset", www.kaggle.com/datasets/dhruvildave/en-fr-translation-dataset?resource=download.

Devlin, Jacob, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. *arXiv*, arxiv.org/abs/1810.04805.

Graves, Alex. Generating Sequences With Recurrent Neural Networks. 2014. *arXiv*, arxiv.org/abs/1308.0850.

Hassan, Hany, et al. Achieving Human Parity on Automatic Chinese to English News Translation. 2018. *arXiv*, arxiv.org/abs/1803.05567.

Hinton, Geoffrey E., et al. Improving neural networks by preventing co-adaptation of feature detectors. 2012. *arXiv*, arxiv.org/abs/1207.0580.

HuggingFace. "Bert Base Multilingual", huggingface.co/bert-base-multilingual-uncased.

———. "BERT Base Uncased", huggingface.co/bert-base-uncased.

———. "Bert Tokenizer", huggingface.co/transformers/v4.9.2/model_doc/bert.html#berttokenizer.

Jozefowicz, Rafal, et al. Exploring the Limits of Language Modeling. 2016. *arXiv*, arxiv.org/abs/1602.02410.

Kingma, Diederik P., and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2017. *arXiv*, arxiv.org/abs/1412.6980.

Liu, Chang, et al. "Better Evaluation Metrics Lead to Better Machine Translation". *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, July 2011, pp. 375–84, aclanthology.org/D11-1035.

Luong, Minh-Thang, et al. Effective Approaches to Attention-based Neural Machine Translation. 2015. *arXiv*, arxiv.org/abs/1508.04025.

NLTK. "Bleu score corpus", www.nltk.org/api/nltk.translate.bleu_score.html#nltk.translate.bleu_score.corpus_bleu.

Papineni, Kishore, et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, July 2002, pp. 311–18, https://doi.org/10.3115/1073083.1073135.

"PyTorch Foundation", pytorch.org/.

Vaswani, Ashish, et al. Attention Is All You Need. 2017. *arXiv*, arxiv.org/abs/1706.03762.

"Weights and Biases", wandb.ai/site.

Wu, Lijun, et al. "Beyond Error Propagation in Neural Machine Translation: Characteristics of Language Also Matter". *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Oct. 2018, pp. 3602–11, https://doi.org/10.18653/v1/D18-1396.

Wu, Yonghui, et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. 2016. *arXiv*, arxiv.org/abs/1609.08144.

Zhang, Xiangwen, et al. Asynchronous Bidirectional Decoding for Neural Machine Translation. 2018. *arXiv*, arxiv.org/abs/1801.05122.

Appendices

A Transformers: Accuracy and Loss

A.1 Transformer with pure attention mechanism

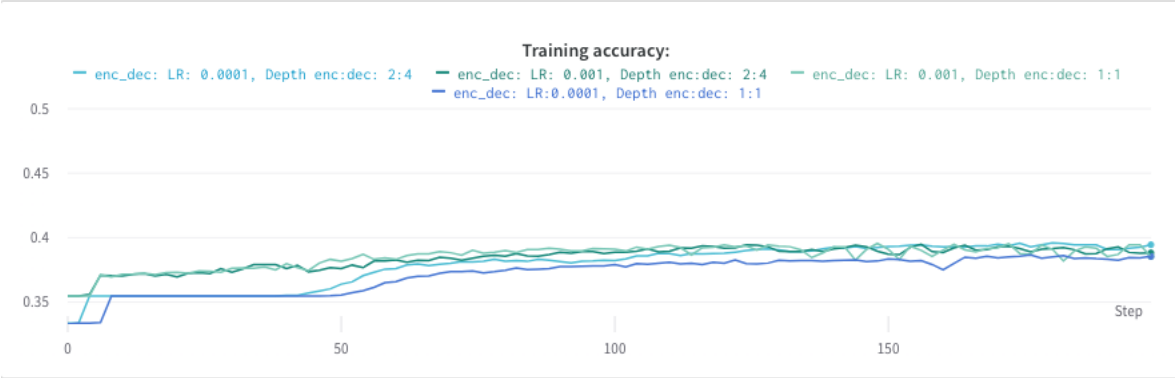

Figure 4: Training accuracy

Figure 5: Training loss

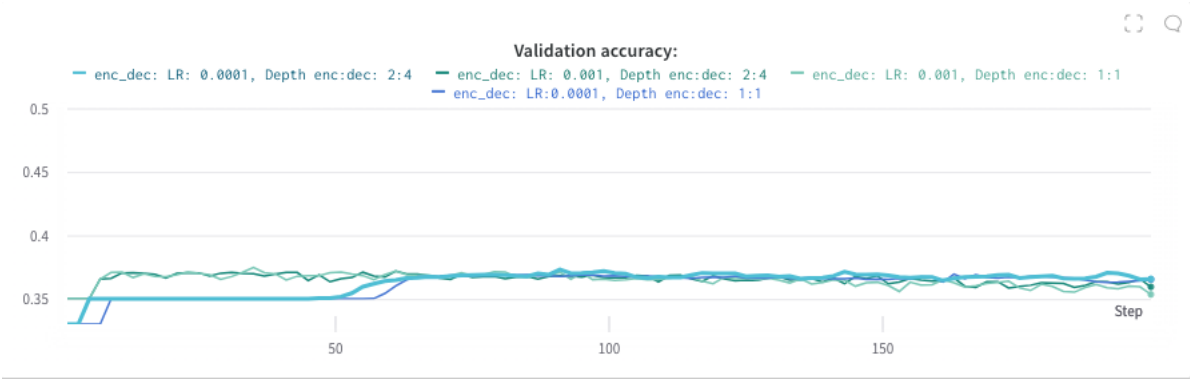

Figure 6: Validation accuracy

Figure 7: Validation loss

A.2 Transformer with attention mechanism combined with GRU layers

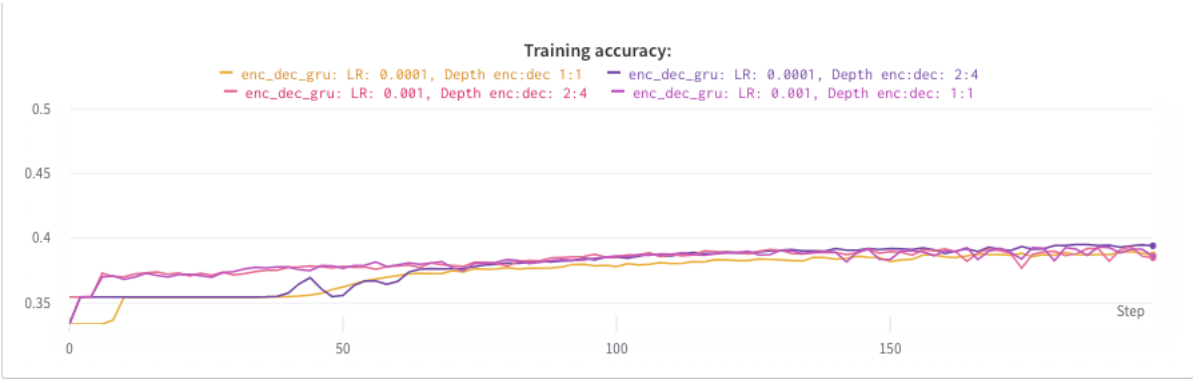

Figure 8: Training accuracy

Figure 9: Training loss

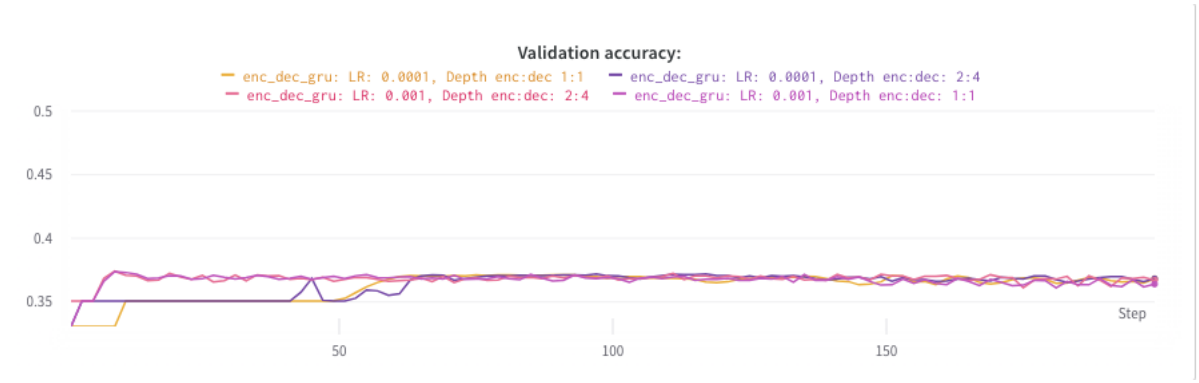

Figure 10: Validation accuracy

Figure 11: Validation loss

B BLEU scores

B.1 Transformer with pure attention mechanism

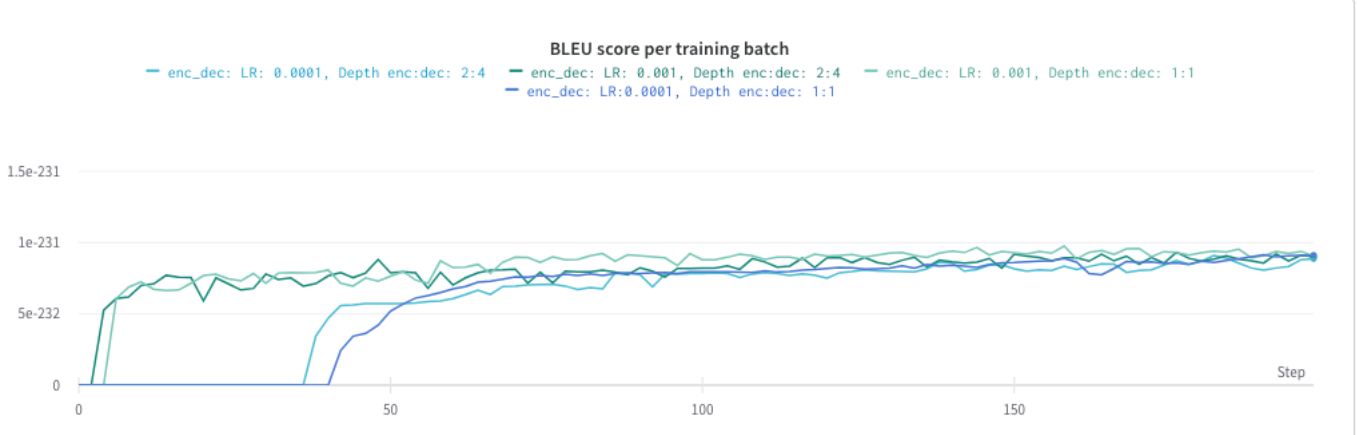

Figure 12: BLEU score: Training

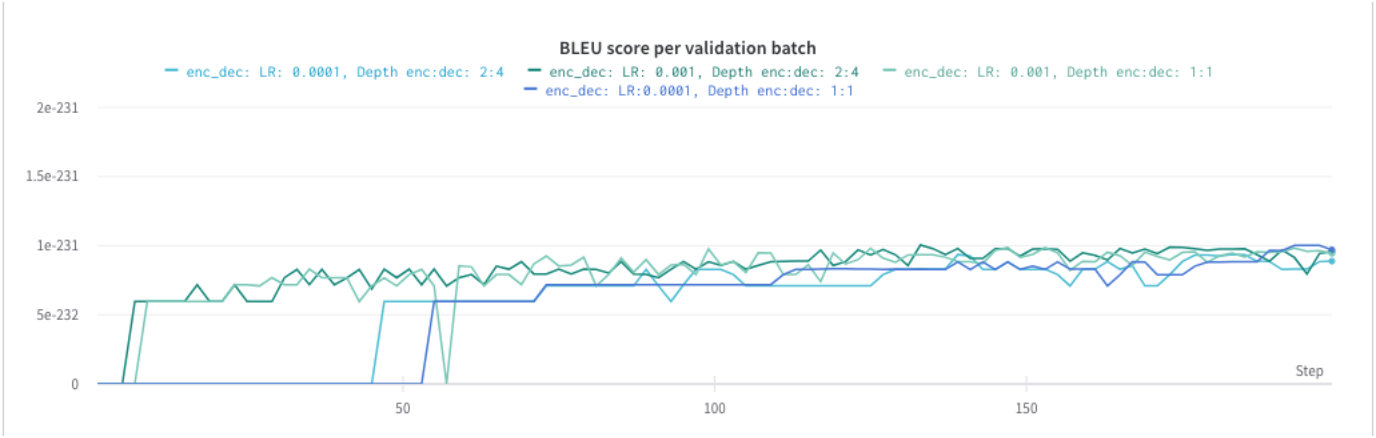

Figure 13: BLEU score: Validation

B.2 Transformer with attention mechanism combined with GRU layers

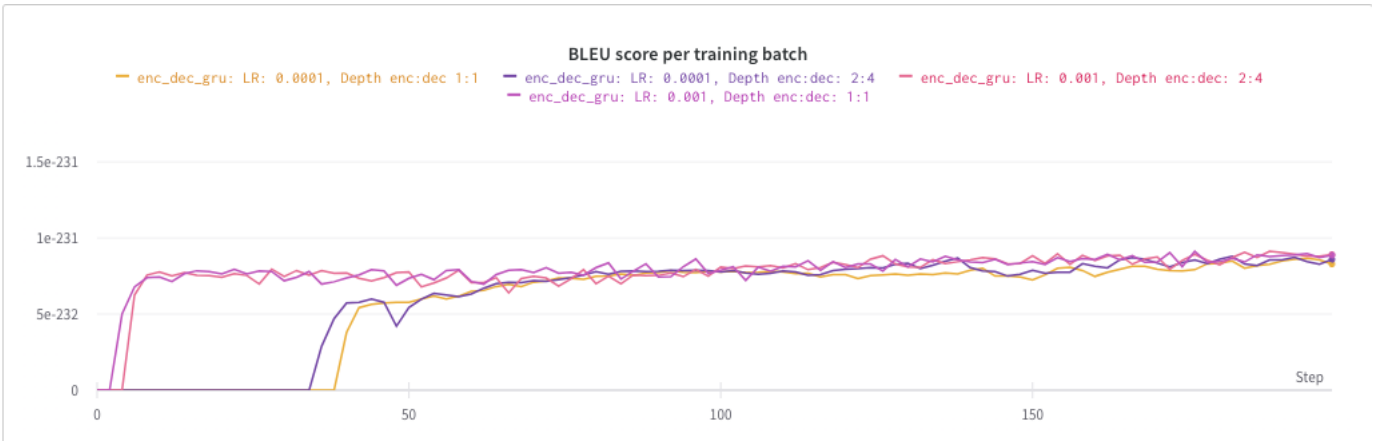

Figure 14: BLEU score: Training

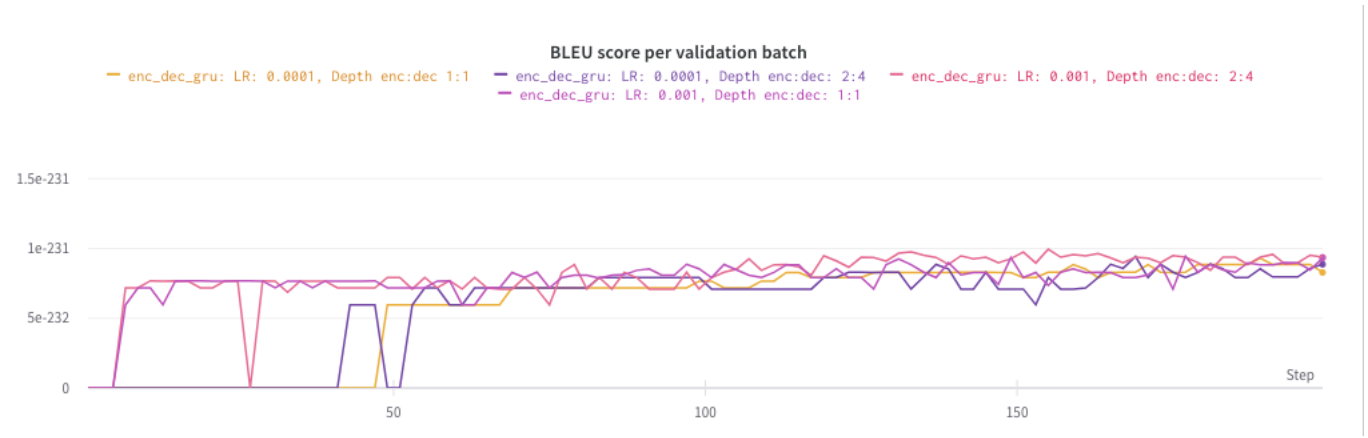

Figure 15: BLEU score: Validation

C Encoded and Decoded true and predicted sentences

C.1 Transformer with pure attention mechanism

The figures in this section represent the performance of the transformer with a pure attention mechanism, a learning rate of 0.0001 and a depth of enc:dec of 2:4.
The sentences are generated in the 99th epoch of the run and are part of the last batch.


```
[[101, 29195, 10165, 183, 20638, 10840, 45626, 10107, 154, 112, 45626, 46336, 72153, 10119, 45626, 83182, 10272, 10106, 20412,
 146, 112, 16147, 10137, 146, 112, 49410, 10111, 154, 112, 45626, 46336, 72153, 10182, 11125, 10109, 30251, 10109, 10302, 119,
 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1
01, 154, 112, 36719, 146, 112, 79411, 10102, 45983, 10934, 16508, 16418, 10182, 11016, 58650, 112, 10145, 16140, 146, 112, 248
05, 10143, 41353, 10344, 10106, 13519, 31255, 146, 112, 10249, 36983, 117, 15265, 117, 10344, 11010, 32310, 117, 10105, 28410,
 10137, 11010, 10173, 12054, 63320, 10870, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0], [101, 10145, 28009, 10344, 10152, 24774, 90975, 10102, 154, 112, 13819, 10169, 12553, 10391, 11010, 45986
, 10107, 10109, 48364, 10735, 15863, 10368, 143, 23994, 10109, 12485, 30939, 65936, 117, 10272, 10106, 56279, 10143, 91187, 10
107, 10137, 50242, 10107, 10102, 17238, 33499, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 12413, 117, 10779, 17290, 11479, 154, 112, 28290, 143, 10106, 11070, 146,
 112, 48364, 10169, 31289, 10102, 154, 112, 24798, 10343, 154, 112, 61176, 10272, 10106, 36753, 10137, 10106, 13154, 146, 112,
 22822, 10343, 10130, 37394, 45626, 12501, 146, 112, 12789, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 11679, 117, 29201, 20486, 10343, 10106, 11831, 13488, 10904, 20010
, 42918, 117, 10119, 47984, 10137, 61560, 18584, 10177, 10379, 10128, 25022, 18565, 10263, 154, 112, 19556, 146, 112, 64914, 1
0119, 45626, 81082, 11184, 66478, 146, 112, 10119, 11016, 39805, 10272, 10106, 10857, 10102, 15607, 119, 102, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 11790, 117, 11144, 13806, 13780, 10
289, 11542, 10343, 10106, 13806, 81082, 11184, 10102, 15607, 10137, 10109, 12545, 117, 10725, 13780, 10102, 13870, 10343, 3189
5, 10106, 13806, 25961, 146, 112, 48364, 10169, 11309, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10130, 11309, 11443, 22883, 13056, 22822, 11930, 85
374, 117, 10379, 35897, 143, 79411, 17492, 26534, 10137, 48753, 39891, 10107, 10102, 154, 112, 37376, 29153, 10343, 10106, 123
64, 10102, 10106, 10708, 87880, 143, 154, 112, 48753, 64861, 10111, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 29195, 10165, 183, 20638, 10840, 29195, 10165, 393
64, 15717, 29201, 113, 14137, 118, 11126, 114, 10145, 10182, 25261, 11043, 10130, 14161, 10102, 154, 112, 48364, 10257, 11309,
 39364, 15717, 29201, 58541, 10109, 14137, 143, 14823, 68430, 10208, 112, 161, 81787, 117, 10109, 15354, 119, 102, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 13396, 26291, 10348, 17950, 112
72, 11536, 11191, 17654, 14784, 10688, 52501, 117, 10145, 143, 49630, 143, 16711, 16563, 117, 14055, 10608, 10102, 40931, 1191
2, 117, 10143, 21064, 10344, 154, 112, 27000, 18080, 11818, 117, 10173, 12054, 63320, 10123, 12105, 10106, 25965, 43378, 10102
, 21064, 10257, 12898, 10109, 10106, 40921, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0], [101, 29195, 10165, 183, 20638, 10840, 29195, 10165, 10779, 17290, 11479, 113, 15412, 118, 11989, 114, 10145, 143, 102
35, 10352, 154, 112, 48364, 74268, 10257, 16182, 10551, 10137, 10668, 10779, 17290, 11479, 58541, 10109, 15412, 143, 12291, 58
```


Figure 16: Reference image: Encoded


```
[[['[CLS]', 'la', 'releve', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]
', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]',
'en', 'tout', ',', 'le', 'satellite', 'pese', '250', 'kilo', '##gramm', '##es', '.', '[SEP]'
, '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'clarence', 'augustus', 'c
hant', 'il', 'est', 'considere', 'comme', 'le', 'pere', 'de', 'l', "'", 'astronomie', 'au',
'canada', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'jack', 'newton', 'ses', 'photographie', '##
s', 'sont', 'rep', '##ute', '##es', 'dans', 'le', 'monde', 'entier', '[SEP]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '
[PAD]', '[PAD]'], ['[CLS]', 'king', 'publie', 'plusieurs', 'articles', 'scientifiq
ues', 'en', 'astronomie', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'le', 'nouvel', 'observatoire', 'est', 'inaugur
e', 'en', '1918', 'et', 'pla', '##ske', '##tt', 'en', 'devient', 'le', 'directeur', '.', '[SE
P]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[
PAD]'], ['[CLS]', 'chant', 'en', 'sera', 'le', 'president', 'entre', '1904', 'et', '1907',
'.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[
PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'il', 'p
eut', 'ainsi', 'passer', '60', 'jours', 'a', 'observer', 'en', 'continu', 'la', 'meme', 'et
oile', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'l', "'", 'observatoire', 'entier',
'pese', '##ra', '660', '##0', 'kilo', '##gramm', '##es', '.', '[SEP]', '[PAD]', '[PAD]', '[
PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]'], ['[CLS]', 'l', "'", 'astronomie', 'cesse', 'alors', 'd', "'", 'etre', 'en
seigne', '##e', 'au', 'departement', 'de', 'physique', 'de', 'l', "'", 'universite', '.',
'[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'sa',
'duree', 'de', 'vie', 'devrait', 'etre', 'de', '5', 'a', '10', 'ans', '.', '[SEP]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'sequence', 'video', 'jay', '##mie', 'mark', 'mat
thews', 'explique', 'ce', 'qu', "'", 'est', 'most', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD
]'], ['[CLS]', 'l', "'", 'observatoire', 'doit', 'son', 'nom', 'au', 'dieu', 'nor', '##veg',
```


Figure 17: Reference image: Decoded

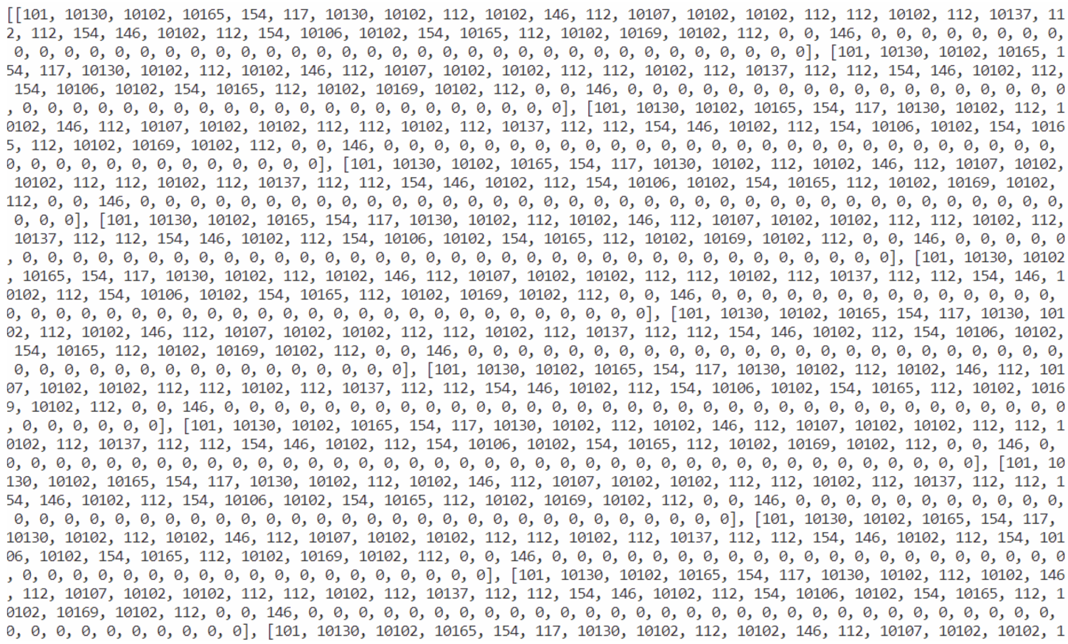

Figure 18: Predicted image: Encoded

Figure 19: Predicted image: Decoded

C.2 Transformer with attention mechanism combined with GRU layers

The figures in this section represent the performance of the transformer with an attention mechanism and GRUs, a learning rate of 0.0001 and a depth of enc:dec of 2:4.
The sentences are generated in the 99th epoch of the run and are part of the last batch.

```
[[101, 29195, 10165, 183, 20638, 10840, 45626, 10107, 154, 112, 45626, 46336, 72153, 10119, 45626, 83182, 10272, 10106, 20412,
 146, 112, 16147, 10137, 146, 112, 49410, 10111, 154, 112, 45626, 46336, 72153, 10182, 11125, 10109, 30251, 10109, 10302, 119,
 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1
01, 154, 112, 36719, 146, 112, 79411, 10102, 45983, 10934, 16508, 16418, 10182, 11016, 58650, 112, 10145, 16140, 146, 112, 248
05, 10143, 41353, 10344, 10106, 13519, 31255, 146, 112, 10249, 36983, 117, 15265, 117, 10344, 11010, 32310, 117, 10105, 28410,
 10137, 11010, 10173, 12054, 63320, 10870, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0], [101, 10145, 28009, 10344, 10152, 24774, 90975, 10102, 154, 112, 13819, 10169, 12553, 10391, 11010, 45986
, 10107, 10109, 48364, 10735, 15863, 10368, 143, 23994, 10109, 12485, 30939, 65936, 117, 10272, 10106, 56279, 10143, 91187, 10
107, 10137, 50242, 10107, 10102, 17238, 33499, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 12413, 117, 10779, 17290, 11479, 154, 112, 28290, 143, 10106, 11070, 146,
 112, 48364, 10169, 31289, 10102, 154, 112, 24798, 10343, 154, 112, 61176, 10272, 10106, 36753, 10137, 10106, 13154, 146, 112,
 22822, 10343, 10130, 37394, 45626, 12501, 146, 112, 12789, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 11679, 117, 29201, 20486, 10343, 10106, 11831, 13488, 10904, 20010
, 42918, 117, 10119, 47984, 10137, 61560, 18584, 10177, 10379, 10128, 25022, 18565, 10263, 154, 112, 19556, 146, 112, 64914, 1
0119, 45626, 81082, 11184, 66478, 146, 112, 10119, 11016, 39805, 10272, 10106, 10857, 10102, 15607, 119, 102, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 11790, 117, 11144, 13806, 13780, 10
289, 11542, 10343, 10106, 13806, 81082, 11184, 10102, 15607, 10137, 10109, 12545, 117, 10725, 13780, 10102, 13870, 10343, 3189
5, 10106, 13806, 25961, 146, 112, 48364, 10169, 11309, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10130, 11309, 11443, 22883, 13056, 22822, 11930, 85
374, 117, 10379, 35897, 143, 79411, 17492, 26534, 10137, 48753, 39891, 10107, 10102, 154, 112, 37376, 29153, 10343, 10106, 123
64, 10102, 10106, 10708, 87880, 143, 154, 112, 48753, 64861, 10111, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 29195, 10165, 183, 20638, 10840, 29195, 10165, 393
64, 15717, 29201, 113, 14137, 118, 11126, 114, 10145, 10182, 25261, 11043, 10130, 14161, 10102, 154, 112, 48364, 10257, 11309,
 39364, 15717, 29201, 58541, 10109, 14137, 143, 14823, 68430, 10208, 112, 161, 81787, 117, 10109, 15354, 119, 102, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 13396, 26291, 10348, 17950, 112
72, 11536, 11191, 17654, 14784, 10688, 52501, 117, 10145, 143, 49630, 143, 16711, 16563, 117, 14055, 10608, 10102, 40931, 1191
2, 117, 10143, 21064, 10344, 154, 112, 27000, 18080, 11818, 117, 10173, 12054, 63320, 10123, 12105, 10106, 25965, 43378, 10102
, 21064, 10257, 12898, 10109, 10106, 40921, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0], [101, 29195, 10165, 183, 20638, 10840, 29195, 10165, 10779, 17290, 11479, 113, 15412, 118, 11989, 114, 10145, 143, 102
35, 10352, 154, 112, 48364, 74268, 10257, 16182, 10551, 10137, 10668, 10779, 17290, 11479, 58541, 10109, 15412, 143, 12291, 58
419, 54662, 10337, 113, 49905, 114, 117, 10109, 24078, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 14586, 10840, 21387, 10102, 10106, 11070, 117, 10379, 18814, 10405, 12193, 10143, 66299
, 10102, 39808, 81082, 12324, 117, 10106, 12355, 10137, 154, 112, 19725, 10102, 13056, 12819, 10102, 37090, 11792, 10735, 3129
```


Figure 20: Reference image: Encoded


```
[[['[CLS]', 'la', 'releve', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]
', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]',
'en', 'tout', ',', 'le', 'satellite', 'pese', '250', 'kilo', '##gramm', '##es', '.', '[SEP]'
, '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'clarence', 'augustus', 'c
hant', 'il', 'est', 'considere', 'comme', 'le', 'pere', 'de', 'l', "'", 'astronomie', 'au',
'canada', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'jack', 'newton', 'ses', 'photographie', '##
s', 'sont', 'rep', '##ute', '##es', 'dans', 'le', 'monde', 'entier', '[SEP]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '
[PAD]', '[PAD]'], ['[CLS]', 'king', 'publie', 'plusieurs', 'articles', 'scientifiq
ues', 'en', 'astronomie', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'le', 'nouvel', 'observatoire', 'est', 'inaugur
e', 'en', '1918', 'et', 'pla', '##ske', '##tt', 'en', 'devient', 'le', 'directeur', '.', '[SE
P]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[
PAD]'], ['[CLS]', 'chant', 'en', 'sera', 'le', 'president', 'entre', '1904', 'et', '1907',
'.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[
PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'il', 'p
eut', 'ainsi', 'passer', '60', 'jours', 'a', 'observer', 'en', 'continu', 'la', 'meme', 'et
oile', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'l', "'", 'observatoire', 'entier',
'pese', '##ra', '660', '##0', 'kilo', '##gramm', '##es', '.', '[SEP]', '[PAD]', '[PAD]', '[
PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]'], ['[CLS]', 'l', "'", 'astronomie', 'cesse', 'alors', 'd', "'", 'etre', 'en
seigne', '##e', 'au', 'departement', 'de', 'physique', 'de', 'l', "'", 'universite', '.',
'[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'sa',
'duree', 'de', 'vie', 'devrait', 'etre', 'de', '5', 'a', '10', 'ans', '.', '[SEP]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]'], ['[CLS]', 'sequence', 'video', 'jay', '##mie', 'mark', 'mat
thews', 'explique', 'ce', 'qu', "'", 'est', 'most', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD
]'], ['[CLS]', 'l', "'", 'observatoire', 'doit', 'son', 'nom', 'au', 'dieu', 'nor', '##veg',
```


Figure 21: Reference image: Decoded

```
[[101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10
102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 1
0102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 14
6, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 1010
2, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10
165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112,
 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117,
10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 15
4, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165
, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 11
7, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 1
0107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 1010
7, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112,
 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102,
117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117,
 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10
109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 1010
2, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 101
02, 154, 10107, 10102, 146, 10102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152,
 112, 10107, 112, 10165, 10102, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 10109, 10165, 10165, 10102, 10102, 154, 10107, 10102, 146, 1
0102, 112, 112, 112, 10102, 10102, 117, 10102, 154, 117, 10102, 10102, 112, 117, 112, 146, 10152, 112, 10107, 112, 10165, 1010
2, 10102, 117, 117, 10102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 22: Predicted image: Encoded

```
[[['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D
]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P
A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[
 P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'
, '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A
D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P
 A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]',
'[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd
e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A
 D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[
 P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]',
 '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D
 ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l
 ', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '
[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]
 ', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A
D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[
P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]']
, ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D
]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P
A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[
 P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'
, '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A
D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd e', 'l', 'd e', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P
 A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]',
'[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]', '[ P A D ]'], ['[ C L S ]', 'l', 'd e', 'l', 'd
```

Figure 23: Predicted image: Decoded

D Calculations for the research questions

D.1 What is the comparative performance of a transformer with a pure attention mechanism versus a transformer with an attention mechanism combined with Gated Recurrent Units (GRUs) in text translation tasks, specifically when trained for 100 epochs?

Calculate the t-test based on the given validation accuracies:
Transformer: Pure attention model: Validation accuracies 0.36, 0.354, 0.37, 0.366
Transformer: Attention model with GRUs: Validation accuracies 0.367, 0.364, 0.368, 0.368

Performing the t-test:
Calculate the mean accuracy for each model:
Transformer: Pure attention model: Mean 1 = $\frac{0.36+0.354+0.37+0.366}{4} \approx 0.3625$

Transformer: Attention model with GRUs: Mean 2 = $\frac{0.367+0.364+0.368+0.368}{4} \approx 0.36675$

Calculate the standard deviation of the validation accuracies for each model:
Transformer: Pure attention model: Standard Deviation 1 ≈ 0.00680
Transformer: Attention model with GRUs: Standard Deviation 2 ≈ 0.00118

Perform an independent samples t-test using the means and standard deviations:
Calculate the pooled standard deviation (Sp) using the formula:

$$Sp = \sqrt{\frac{(n_1 - 1) \cdot Standard\_Deviation1^2 + (n_2 - 1) \cdot Standard\_Deviation2^2}{n_1 + n_2 - 2}}$$

$$Sp = \sqrt{\frac{(801 - 1) \cdot 0.00680^2 + (801 - 1) \cdot 0.00118^2}{801 + 801 - 2}} \approx 0.00488$$

Calculate the t-value using the formula:

$$t = \frac{Mean1 - Mean2}{Sp \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

$$t = \frac{0.3625 - 0.36675}{0.00488 \cdot \sqrt{\frac{1}{801} + \frac{1}{801}}} \approx -17.42$$

Determine the critical t-value at a 95% confidence level ($\alpha = 0.05$) and degrees of freedom ($df = n_1 + n_2 - 2 = 801 + 801 - 2 = 1600$).
The critical t-value for a two-tailed t-test with $df = 1600$ and $\alpha = 0.05$ is approximately ± 1.962.
Compare the calculated t-value to the critical t-value. If the calculated t-value falls within the range of the critical t-value (i.e., between -1.962 and 1.962), we fail to reject the null hypothesis. Otherwise, if the calculated t-value is outside that range, we reject the null hypothesis.

Since the calculated t-value (-17.42) falls outside the range of the critical t-value (-1.962 to 1.962), we reject the ***H0:*** *There is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism combined with GRUs in the text translation task after training for 100 epochs.*

D.2 Do deeper encoder-decoder blocks affect the performance of the transformers?

Calculate the difference in validation accuracies between the deeper and less deep models for each model.
Transformer: Pure attention
Group A (deeper model, 2:4) accuracies: 0.36, 0.37
Group B (less deep model, 1:1) accuracies: 0.354, 0.366

Transformer: Attention with GRU layers
Group C (deeper model, 2:4) accuracies: 0.367, 0.368
Group D (less deep model, 1:1) accuracies: 0.364, 0.368

Calculations:
Mean accuracy for each group:
Group A mean (mean1) = $\frac{0.36+0.37}{2} = 0.365$

Group B mean (mean2) = $\frac{0.354+0.366}{2} = 0.36$

Group C mean (mean3) = $\frac{0.367+0.368}{2} = 0.3675$

Group D mean (mean4) = $\frac{0.364+0.368}{2} = 0.366$

The standard deviation for each group:
Group A standard deviation (s1) $\approx \sqrt{\frac{(0.36-0.365)^2+(0.37-0.365)^2}{2-1}} \approx 0.00707$
Group B standard deviation (s2) $\approx \sqrt{\frac{(0.354-0.36)^2+(0.366-0.36)^2}{2-1}} \approx 0.00849$
Group C standard deviation (s3) $\approx \sqrt{\frac{(0.367-0.3675)^2+(0.368-0.3675)^2}{2-1}} \approx 0.000707$
Group D standard deviation (s4) $\approx \sqrt{\frac{(0.364-0.366)^2+(0.368-0.366)^2}{2-1}} \approx 0.002828$

Pooled standard deviation (Sp):
$\text{Sp} = \sqrt{\frac{(n1-1)\cdot s1^2+(n2-1)\cdot s2^2+(n3-1)\cdot s3^2+(n4-1)\cdot s4^2}{n1+n2+n3+n4-4}}$

$\text{Sp} \approx \sqrt{\frac{(2-1)\cdot 0.00707^2+(2-1)\cdot 0.00849^2+(2-1)\cdot 0.00071^2+(2-1)\cdot 0.00283^2}{2+2+2+2-4}} \approx 0.006$

t-value calculation:
$\text{t} = \frac{mean1-mean2}{Sp\cdot\sqrt{\frac{1}{n1}+\frac{1}{n2}}}$

$\text{t} \approx \frac{0.3635-0.366}{0.006\cdot\sqrt{\frac{1}{2}+\frac{1}{2}}} \approx 0.42$

Critical t-value:
At a 95% confidence level and degrees of freedom (df = n1 + n2 + n3 + n4 - 4 = 2 + 2 + 2 + 2 - 4 = 4), the critical t-value is approximately ±2.92.

$\text{t}_{AB} = \frac{0.365-0.36}{0.006\cdot\sqrt{\frac{1}{2}+\frac{1}{2}}} \approx 0.8333$
$t_{CD} = \frac{0.3675-0.366}{0.006\cdot\sqrt{\frac{1}{2}+\frac{1}{2}}} \approx 0.25$

The critical t-value for a two-tailed t-test with $df = 4$ and $\alpha = 0.05$ is approximately ±2.776.

Comparing the calculated t-values to the critical t-value (±2.776), we can make the following conclusions:

For Group A vs Group B: Since $|t_{AB}| = 0.8333 < 2.776$, we fail to reject the null hypothesis.

For Group C vs Group D: Since $|t_{CD}| = 0.25 < 2.776$, we fail to reject the null hypothesis.

Therefore, based on the available data and using a 95% confidence level, we fail to reject ***H0:*** *There is no significant difference in the performance of a transformer with a pure attention mechanism and a transformer with an attention mechanism with GRUs when deeper encoder-decoder blocks are implemented.*