Implementing Transformers

Johannes Widera

Seminar Paper

Professorship: Dialog Systems and Machine Learning

Submission: January 2025

Supervisor: Supervisor: Dr. Carel van Niekerk

1 Introduction

Transformers have revolutionized natural language processing (NLP) by replacing slow, sequential methods like recurrent neural networks (RNNs)[Cho+14] with parallelizable attention mechanisms. These models excel at tasks such as translation, text generation, and summarization because they process entire sentences at once and capture relationships between distant words. However, training and fine-tuning large Transformers remains computationally expensive, limiting their accessibility for resource-constrained applications.

This paper explores the implementation of a Transformer model from scratch, emphasizing efficiency and adaptability. It examines **Low-Rank Adaptation (LoRA)**, a parameter-efficient fine-tuning technique that reduces the number of trainable parameters, a step towards lowering said computational costs.

By analyzing the model architecture, training process, and the effects of LoRA, this work provides insights into optimizing Transformer efficiency. The study evaluates LoRA with ranks ranging from **4 to 32**, aiming to uncover the trade-offs between model complexity and performance. By addressing the question of whether performance gains continue at higher ranks or plateau due to diminishing returns, this research provides valuable insights into optimizing Transformer efficiency.

2 Transformer

The Transformer architecture, proposed by [Vas17], consists of two key components: an **encoder** and a **decoder** (see Figure 1). The encoder constructs a contextual representation of the input, while the decoder generates output text autoregressively, one token at a time. The Transformer owes its widespread popularity to its unique architectural design, which centers around these components. The encoder processes input sequences using stacked layers of multi-head self-attention and feed-forward networks. In contrast, the decoder autoregressively generates output tokens using masked self-attention and cross-attention over the encoder outputs. To produce text after the decoder, the output logits are passed through a final linear layer, followed by a Softmax function, which generates a probability distribution over the entire vocabulary for each token. The simplest method to generate a sequence involves selecting the token with the highest probability at each step, a process known as greedy decoding.

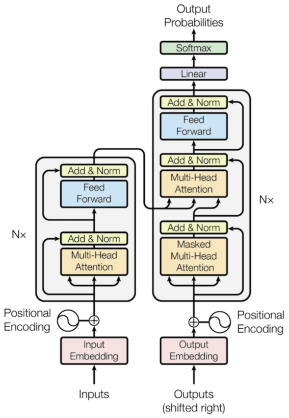

Figure 1: Encoder-decoder architecture of the Transformer model [Vas17].

In the following sections, we highlight the most essential components of the Transformer architecture.

2.1 Input Representation: Tokenization and Embedding

The process of transforming raw text into a format suitable for the Transformer involves three critical steps: tokenization, embedding, and positional encoding. This pipeline ensures that the input text is converted into a numerical representation that the model can process.

BPE Tokenizer

The first step involves tokenizing the input text using the Byte Pair Encoding (BPE) tokenizer [Gag94]. BPE iteratively identifies the most frequently occurring pair of symbols in the text and replaces all occurrences of this pair with a new symbol. This process shortens the text while maintaining its original information. The newly created symbols, along with the pairs they replace, are stored in a lookup table, enabling the reconstruction of the original text [KV24]. The number of new symbols is predefined as vocabulary size to achieve efficient compression. Our initial vocabulary size was 50000, but we later adjusted it to 50048, the closest multiple of 64, as suggested by Andrej Karpathy to speed up training [Kar].

Embedding

After tokenization, tokens are converted into dense vector representations using an embedding layer. This layer maps token indices to high-dimensional embeddings that capture semantic and syntactic relationships. Mathematically, embeddings are retrieved as:

$$\text{Emb}(\mathbf{x}) = \mathbf{1}(\mathbf{x})\mathbf{E}$$

where $\mathbf{E}$ is the embedding matrix.

Positional Encoding

Since transformer architectures lack an inherent sense of word order, positional encodings are added to token embeddings to inject information about the position of each token in the sequence. This step ensures that the model can account for word order, which is crucial for processing natural language. A common approach is to use sinusoidal positional encodings, as introduced by [Vas17], defined as follows:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{1}$$

where each position **pos** in a sequence is mapped to a unique vector using alternating sine and cosine functions. The even-indexed dimensions apply a sine function, while the odd-indexed dimensions use a cosine function. The index i represents the dimension index of the encoding vector.

The denominator includes a scaling factor of 10000, ensuring that different encoding dimensions operate at different frequencies. Lower dimensions correspond to higher frequency variations, capturing local positional relationships, while higher dimensions correspond to lower frequencies, capturing broader positional trends. The total model dimension d_{model} controls how these frequencies are distributed across different encoding dimensions.

The authors justify the use of Sinusoidal embeddings based on three key properties, which we prove in 6.

1. **Linear Offset Representation**: For a fixed offset k, PE_{pos+k} can be represented as a linear transformation of PE_{pos}.
2. **Geometric Wavelength Spacing**: The wavelengths span 2π to $2\pi \cdot 10000$ in geometric progression.
3. **Extrapolation Capability**: Sinusoidal encodings generalize to unseen sequence lengths better than learned embeddings.

The Transformer's input representation is computed through element-wise summation of token embeddings and positional encodings:

$$\mathbf{h}_0 = \text{Embedding}(x) + \text{PositionalEncoding}(x)$$

where $\mathbf{h}_0 \in \mathbb{R}^{d_{\text{model}}}$ represents the initial hidden state containing both semantic and positional information, which is subsequently processed by the Transformer encoder layers.

2.2 Scaled Dot-Product Attention

The attention mechanism addresses key limitations of previous models, enabling dynamic focus on relevant parts of input sequences, improving parallelization, and mitigating information loss from compressing encoder outputs into a single vector [Voi20; Hua]. The model computes token interactions utilizing three matrices: **Query (Q)**, **Key (K)**, and **Value (V)**. These matrices are derived from the input embeddings X. Specifically, they are obtained by multiplying X with learned weight matrices W_Q, W_K, and W_V, respectively:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

Here, X represents the input token embeddings, and W_Q, W_K, and W_V are the weight matrices for Queries, Keys, and Values.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

This mechanism prioritizes input elements by following a sequence of steps. First, it performs **Query-Key Matching**, where it computes similarity scores between queries and keys using dot products. Next, these scores are scaled by dividing them by $\sqrt{d_k}$, a process known as **Scaling**, which stabilizes learning in high-dimensional spaces. The scaled scores are then passed through a **Softmax** function, which converts them into probabilities, thereby emphasizing the most relevant matches. Finally, **Value Aggregation** is performed, where values are combined and weighted by these probabilities.

Attention types include:

- **Self-Attention**: Queries, keys, and values derive from the same sequence, associating context-sensitive tokens.
- **Cross-Attention**: Queries come from one sequence, keys and values from another, such as aligning source and target sequences between encoder and decoder.

Multi-Head Attention

Multi-head attention enhances the core mechanism by processing multiple attention computations in parallel, each capturing diverse patterns in subspaces. Outputs from all heads are concatenated and projected back to the original dimensionality, enabling the model to represent complex dependencies and long-range relationships effectively.

2.3 Masking

Masking is a key part of how Transformers work, helping the model focus on the right parts of the input. It is used to ignore tokens that should not affect the output. There are two types of masking.

Masking Padded Tokens

When working with sequences of different lengths, we add `<pad>` tokens to make them the same size. In our case our max sequence was 64 therefore we added `<pad>` token to any sequence that was less then 64 tokens long. These extra tokens do not have any real meaning, so we need to make sure the model ignores them. A good strategie is to use a binary mask to block these tokens from affecting the attention mechanism.

Future Masking

In tasks like text generation, the decoder part of the model should only use tokens up to the current position. it shouldn't "cheat" at what comes next. To enforce this, we added a mask that blocks attention to future tokens by setting their attention scores to $-\infty$ as visualized in 2.

2.4 Residual Connections and Layer Normalization

- **Residual Connections**: Maintain gradient flow and representation quality in deep networks, preventing rank collapse and ensuring diverse and informative representations [DCL21].

$$x_{\text{out}} = x + \text{Dropout}(\text{Sublayer}(x)) \tag{3}$$

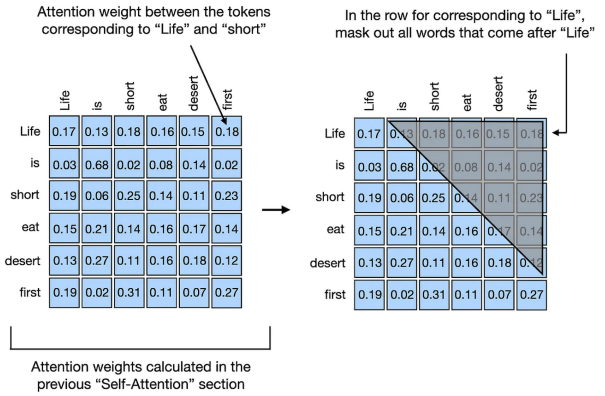

Figure 2: Illustration of future Masking [SR]

- **Layer Normalization**: Stabilizes training by standardizing activations within each layer:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sigma + \epsilon} + \beta \tag{4}$$

These components mitigate vanishing gradients and accelerate convergence.

2.5 Position-Wise Feed Forward Network

The last important component of a Transformer is the feed-forward network (FFN), a deceptively simple yet indispensable component. Defined as:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{5}$$

it processes each token independently through two linear layers separated by a ReLU activation, expanding inputs to a hidden dimension (typically 4× the model size) before compressing them back [Vas17]. While attention layers capture global relationships, the FFN operates locally, refining features by injecting **nonlinearity**, a critical capability absent in the mostly linear attention operations. This allows the model to learn hierarchical patterns that attention alone cannot encode [Wan].

3 Low-Rank Adaptation (LoRA)

Recent advances in parameter-efficient fine-tuning have demonstrated that full weight updates are unnecessary for adapting large language models to downstream tasks. Among emerging methods, last-layer tuning, adapter layers [Hou+19], and prefix tuning [LL21], LoRA (Low-Rank Adaptation) [Hu+21] offers distinct advantages through its mathematical formulation and computational efficiency.

3.1 Theoretical Foundation

The approach builds on the intrinsic dimensionality hypothesis [AZG20], which states that neural networks possess low-dimensional subspaces sufficient for effective task adaptation. Formally, for pretrained weights $W_0 \in \mathbb{R}^{d\times k}$, LoRA constrains weight updates ΔW to a low-rank decomposition:

$$W' = W_0 + \Delta W = W_0 + BA, \quad B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times k},\ r \ll \min(d,k) \tag{6}$$

where r represents the update rank. This rank constraint ($r \approx 1-64$) captures the essential task-specific directions in the high-dimensional parameter space while reducing trainable parameters by $O(dr+rk)$ vs $O(dk)$ for full fine-tuning.

3.2 Implementation Details

The decomposition matrices B and A are initialized with zero-centered normal distributions for A and zeros for B, ensuring $\Delta W = 0$ at initialization. During forward propagation:

$$h = W_0 x + \alpha\left(\frac{BAx}{r}\right) \tag{7}$$

where α scales the magnitude of updates relative to original weights. The rank-normalized scaling (α/r) balances preservation of pretrained knowledge with task adaptation 3.

In transformer architectures, LoRA typically modifies query/key/value projections in attention layers.

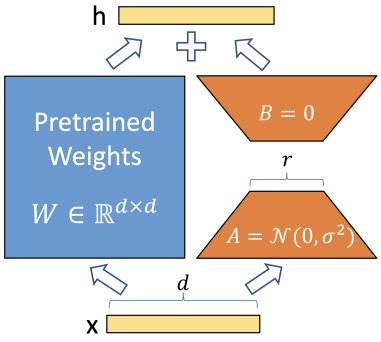

Figure 3: LoRa reparametrization. Only A and B are trained [Hu+21]

3.3 Advantages

Key benefits emerge from this design:

- **Parameter Efficiency**: Reduces trainable parameters by >90% versus full fine-tuning, enabling GPU memory reduction [Hu+21].
- **Task Switching**: Rapid deployment of multiple specialized models through interchangeable BA matrices.

- **Rank Flexibility**: Optimal r (typically 1-64) adapts to task complexity - lower ranks suffice for semantically similar tasks, higher ranks handle domain shifts.

Experiments show LoRA achieves comparable accuracy to full fine-tuning, establishing it as a principled approach for efficient model adaptation. The method's mathematical elegance stems from directly parameterizing the update manifold rather than the weights themselves, providing both theoretical guarantees and practical efficiency. In the next section, we will highlight LoRA's adaptability by fine-tuning a Base Transformer on a synthetic task and examining how varying the Rank affects its accuracy.

4 Training

4.1 Training the Base Transformer

Model Variation

Our implementation adopts reduced-scale parameters in comparison to [Vas17], including $N = 4$ encoder and decoder layers, 4 attention heads per multi-head block, a model dimension of $d_{\text{model}} = 128$, a feed-forward network dimension of $d_{\text{ff}} = 512$, and a maximum sequence length of 64 tokens. This scaled-down design addresses computational constraints and mitigates overfitting risks given limited training data. The model contained in total 8.2M parameters.

Learning Rate Scheduling

A warmup phase stabilizes early training:

$$lr = d_{\text{model}}^{-0.5} \cdot \min(\text{step}^{-0.5}, \text{step} \cdot \text{warmup}^{-1.5}) \tag{8}$$

For $d_{\text{model}} = 128$ and warmup = 4000, the learning rate peaks at step 4,000.

Optimizer

We initialized the AdamW optimizer with a learning rate of $\alpha = 0.5$, which demonstrated more stable training. The remaining parameters were set as follows, following [Vas17]: $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$. The configuration balances momentum-based gradient smoothing and adaptive regularization while preserving the original Transformer's training dynamics.

Parameter Sharing

We implemented weight tying between the token embedding layers (source/target) and output projection layer, following the Transformer's standard practice. This strategy (1) ensures dimensional alignment between

embedding and prediction spaces for stable training, and (2) enhances generalization through consistent token representations across encoder-decoder interactions.

Hardware

All experiments were conducted on an Apple M1 Max chip (32GB RAM) using PyTorch's Metal Performance Shaders (MPS) for hardware acceleration. The base model training took approximately 26 hours on this hardware.

Dataset

The Base Model was trained on a 1M big subset of the WMT17 English-German translation corpus [Boj+17], which was preprocessed through UTF-8 normalization and regex-based cleaning, removing URLs/HTML tags while preserving alphanumeric characters and core punctuation. Sequences were lowercased and filtered to 5–64 tokens via length validation. Each batch contained 128 sentence pairs.

4.2 LoRa Rank Experiment

We fine-tuned the Base Model with LoRa on a synthetic dataset containing 4 tasks, all involving sequences of length 4 with numbers between 1 and 100. The tasks were as follows:

1. Sorting sequences in increasing/decreasing order.
2. Predicting sequences where each number is incremented by 1 or 2.

To inform the model of the specific task to perform, we added a Hard Prompt that specifies the task. For example, given the input "add 2: 8 13 2 5", the expected output is "10 15 4 7".

LoRA

We implemented LoRA with different fixed ranks (4, 8, 16, 32) and a scaling factor set to 1. At least 97% and at most over 99% of the model parameters were frozen, with updates applied exclusively to the attention layers. This approach significantly reduced the number of trainable parameters while maintaining the performance of the base model.

5 Results and Analysis

5.1 Base Transformer Performance

We achieved a mean BLEU score of **6.9**. Although this score is modest, it can be largely attributed to the limited computational resources and the constrained dataset, which seemingly did not fully capture the

diversity of the task. Our next steps would involve scaling the model up to the original size described by [Vas17] and training on the complete dataset. We are optimistic that this approach will yield significantly better results. Nonetheless, the current results were sufficient to demonstrate how our model learned the translation task 1.

5.2 LoRa Rank Results

Our analysis of LoRA rank influence on model performance reveals several crucial insights:

- **Task Difficulty Hierarchy:** We observed a clear hierarchy in task complexity, with the descending sort being consistently the most challenging task (28% at rank 8) and the +1 operation being the easiest (99% at rank 32). This suggests that reverse ordering operations require more complex internal representations than simple arithmetic operations.
- **Critical Rank Threshold:** A minimum rank threshold between 4 and 8 is crucial for meaningful learning. At rank 4, the model fails to learn any task (all accuracies $\leq 0.4\%$), while rank 8 enables basic task comprehension (42% multi-task accuracy).
- **Optimal Performance Point:** Rank 16 emerges as an optimal trade-off point, achieving 88% multi-task accuracy with 144k parameters. The improvement to rank 32 (95% accuracy, 188k parameters) shows diminishing returns relative to the parameter increase.
- **Task-Specific Learning:** The performance gap between ascending and descending sort (31 percentage points at rank 8) indicates an inherent bias toward learning increasing sequences. Arithmetic operations demonstrate more stable learning patterns across ranks compared to sorting tasks.

Table 1: LoRA Rank vs. Performance Metrics

LoRA Rank (r)	Multi-Task Acc	Sort asc	Sort desc	+1	+2	~ Total Trained Params
4	0.2%	0.4%	0.1%	0.4%	0%	36k
8	42%	59%	28%	40%	42%	72k
16	88%	86%	81%	97%	88%	144k
32	95%	91%	91%	99%	95%	288k

6 Conclusion

This study explored the implementation of a Transformer model from scratch, emphasizing LoRA as a parameter-efficient fine-tuning technique.

Our findings reveal a non-linear relationship between LoRA rank and accuracy, highlighting trade-offs between performance and computational efficiency. Rank 32 achieves the highest accuracy (95%), but rank 16 offers a practical alternative (88%) with fewer parameters. The sharp performance jump from rank 4 to 8 (0.2% to 42%) suggests an efficiency threshold beyond which accuracy gains diminish.

Optimal rank selection depends on task complexity: simple tasks perform well at low ranks, while complex ones require higher ranks. This supports dynamic rank allocation instead of a fixed setting, with hybrid approaches potentially improving efficiency. Identifying an "elbow point" (Figure 2) in the efficiency curve could inform better rank allocation for large-scale applications.

While our findings offer practical insights, the study's limitations include the use of synthetic tasks, making direct generalization to real-world domains uncertain. Additional research should explore factors like training stability, inference latency, and memory footprint. Future work should investigate adaptive rank allocation and alternative LoRA architectures to further enhance efficiency and applicability across diverse NLP tasks.

Appendix

1 Positional Encoding: Properties

1. **Linear Offset Representation**: For a fixed offset k, PE_{pos+k} can be represented as a linear transformation of PE_{pos}.

   Applying trigonometric angle addition formulas to the sinusoidal components:

$$\sin\left(\frac{pos+k}{\lambda_i}\right) = \sin\left(\frac{pos}{\lambda_i}\right)\cos\left(\frac{k}{\lambda_i}\right) + \cos\left(\frac{pos}{\lambda_i}\right)\sin\left(\frac{k}{\lambda_i}\right),$$
$$\cos\left(\frac{pos+k}{\lambda_i}\right) = \cos\left(\frac{pos}{\lambda_i}\right)\cos\left(\frac{k}{\lambda_i}\right) - \sin\left(\frac{pos}{\lambda_i}\right)\sin\left(\frac{k}{\lambda_i}\right),$$

   where $\lambda_i = 10000^{2i/d_{\text{model}}}$. This shows $\mathbf{PE}_{pos+k}$ is a linear combination of $\mathbf{PE}_{pos}$, parameterized by a block-diagonal matrix of 2×2 rotation matrices.

2. **Geometric Wavelength Spacing**: The wavelengths span 2π to $2\pi \cdot 10000$ in geometric progression.

   For $i = 0$, the effective wavelength is:

$$\lambda_{\text{effective},0} = 2\pi \cdot 10000^0 = 2\pi.$$

   For $i = d_{\text{model}}/2 - 1$, the wavelength becomes:

$$\lambda_{\text{effective},d_{\text{model}}/2-1} = 2\pi \cdot 10000^{2(d_{\text{model}}/2-1)/d_{\text{model}}} \approx 2\pi \cdot 10000^{1-2/d_{\text{model}}}.$$

   When d_{model} is large (e.g., 512), $10000^{1-2/d_{\text{model}}} \approx 10000$, yielding $\lambda_{\text{effective}} \approx 2\pi \cdot 10000$. The ratio between consecutive wavelengths is constant:

$$\frac{\lambda_{\text{effective},i+1}}{\lambda_{\text{effective},i}} = 10000^{2/d_{\text{model}}}.$$

   This geometric spacing ensures the model captures positional relationships at multiple scales.

3. **Extrapolation Capability**: Sinusoidal encodings generalize to unseen sequence lengths better than learned embeddings.

   Learned embeddings bind positional information to discrete training positions, while sinusoidal encodings are continuous functions of pos. For an unseen position $L_{\text{train}} + k$, the encoding can be

derived via:

$$\mathbf{PE}_{L_{\text{train}}+k} = \mathbf{M}_k \cdot \mathbf{PE}_{L_{\text{train}}},$$

where $\mathbf{M}_k$ is the same linear transformation matrix used during training (see Property 1). This structural bias enables extrapolation, as demonstrated empirically in [Vas17]. In contrast, learned embeddings lack such continuity and fail to generalize beyond L_{train}.

2 Base Model Evaluation

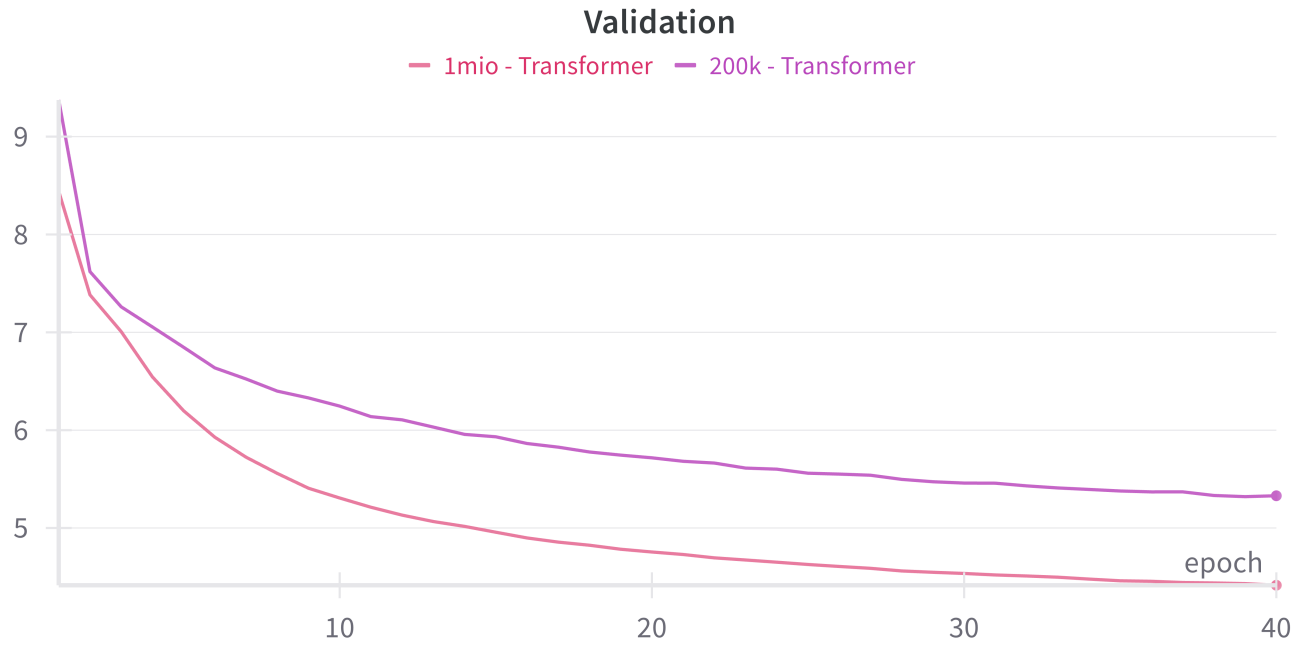

Figure 1: Comparison of the validation loss of the same transformer over 40 epochs, suggesting that increasing the dataset results in a significant improvement in the loss.

Table 1: Best Performing Translations

Source (German)	**Prediction (English)**	**Reference (English)**	**BLEU**
aber wenn new york veränderung ist, dann ist es auch neuerfindung.	but if new york is change, it is also new-to-date.	but if new york is change, it is also reinvention.	0.8801
die anschläge am donnerstag und freitag erfolgten wenige tage nach einem umstrittenen referendum über eine von der militärjunta ausgearbeitete verfassung.	the attacks on thursday and friday were a few days after a controversial referendum on a constitution drawn up by the military	the attacks on thursday and friday came just a few days after a controversial referendum on a constitution drawn up by the military junta.	0.7967
die marke ulster bank wird bereits für kunden in der republik irland und nordirland verwendet.	the brand bank is already used for customers in the republic of ireland and northern ireland. is	the ulster bank brand is already used for customers in the republic of ireland and northern ireland.	0.7798
es ist faszinierend zuzusehen, wie sich die welt verändert.	it is fascinating to see how the world is changing.	it is fascinating to see how the world turns.	0.7598
russland und türkei: eine allianz von außenseitern?	russia and turkey: an alliance of external members?	russia and turkey: an alliance of misfits?	0.6804

Table 2: Worst Performing Translations

Source (German)	**Prediction (English)**	**Reference (English)**	**BLEU**
es verändert ein wenig den ausblick.	there is little change in the direction. is no	it alters your outlook a bit.	0.0000
der metereologe alek krautmann sagt, dass beide flüsse viele häuser in den vorortgegenden nahe baton rouge überschwemmen könnten.	the minute, mr kraututmann, says that	forecaster alek krautmann said both rivers could flood many houses in suburban areas near baton rouge.	0.0000
schorndorf: einbrecher mit hubschrauber gesucht	schororg: leave to helicopters, thennnnn	schorndorf: search for intruders made by helicopter	0.0000
auf der speisekarte stehen sandwiches mit speck und käse und es gibt garnierungen wie kapern und sautierte zwiebeln.	then, then, then, then, then, then, then,	bacon-egg-and-cheese sandwiches are on the menu, and toppings like capers and sauted onions are available.	0.0000
die beeinträchtigung kann ganz unterschiedlich aussehen: man ist abgeschlagen, müde, unausgeschlafen, hat schwierigkeiten mit der aufmerksamkeit und der konzentration, ist leicht reizbar, hat stimmungsschwankungen oder körperliche beschwerden wie magen- oder kopfschmerzen.	the fact that the european union is very differe	they can impact in various different ways: you could be exhausted, tired, poorly rested, suffer from difficulties concentrating and staying alert, be irritable, have mood swings or physical complains such as stomach ache or headaches.	0.0000

3 LoRa Plots

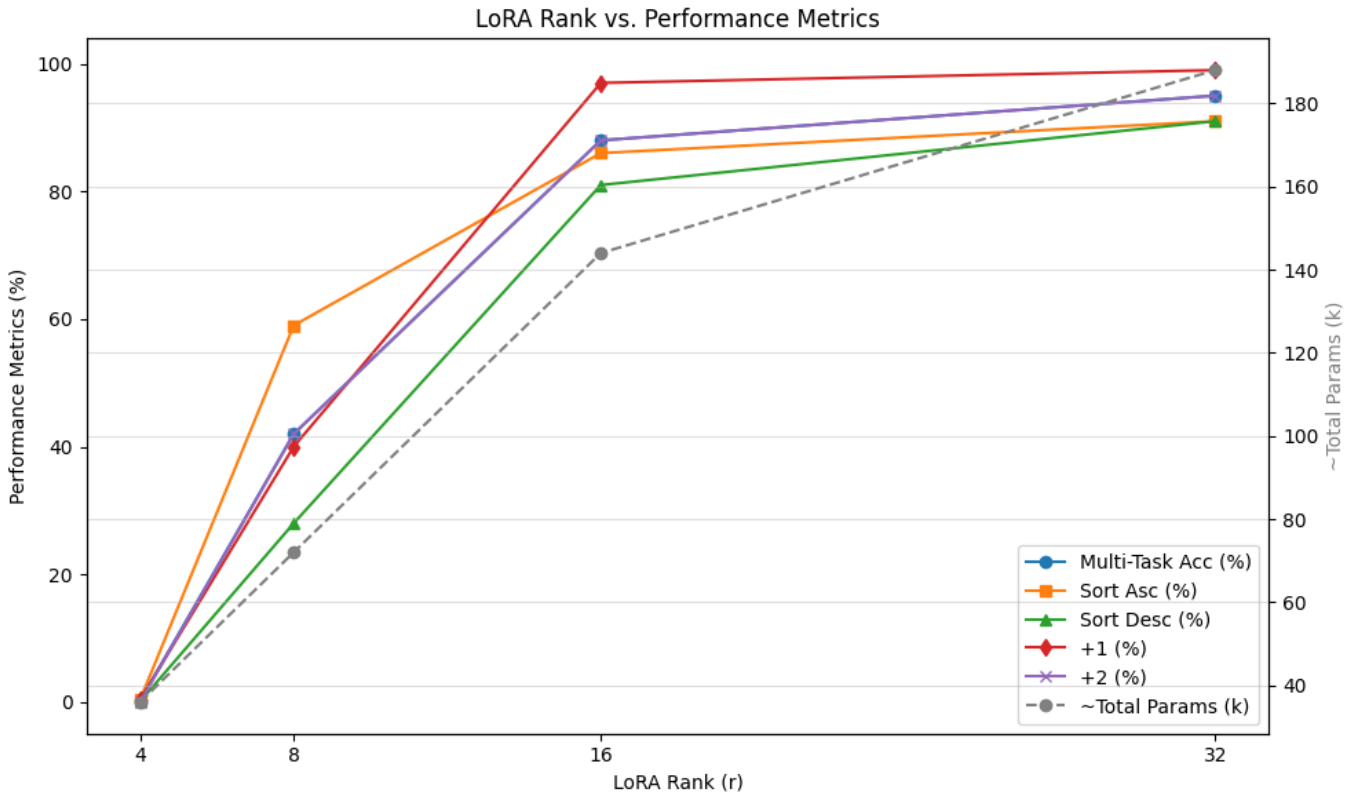

Figure 2: LoRA Rank vs. Performance Metrics. The chart demonstrates how performance metrics and total parameters vary with LoRA rank.

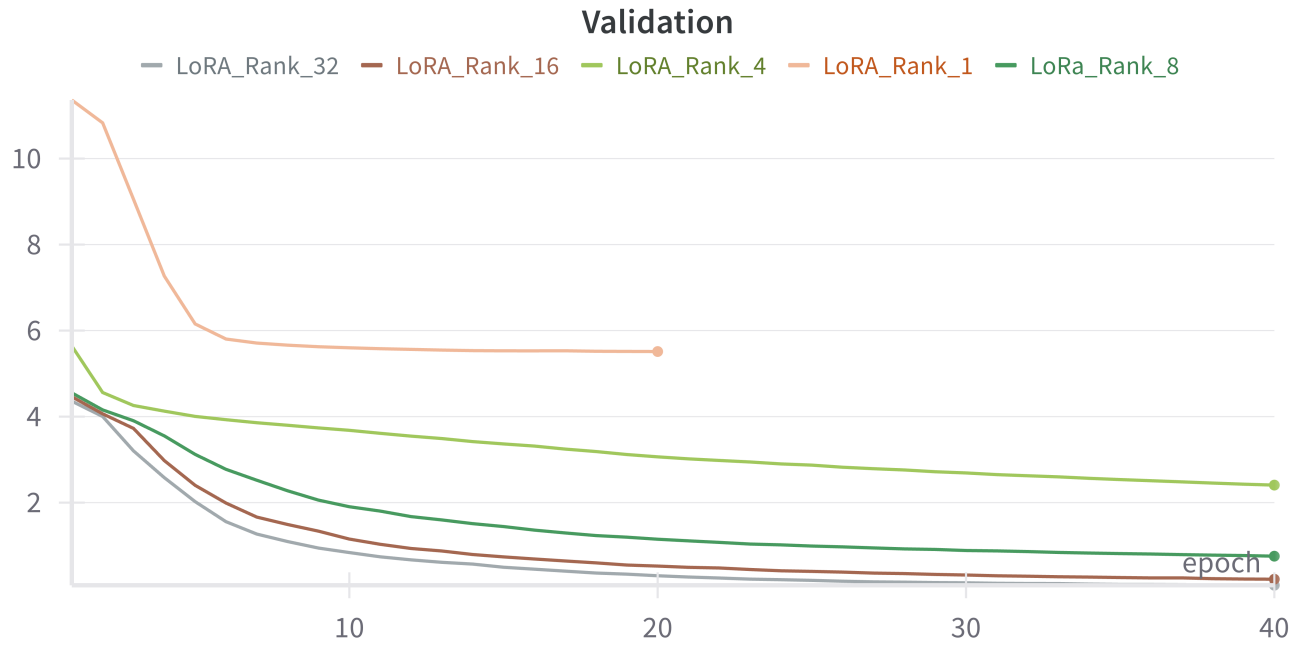

Figure 3: LoRA Rank validation loss. The chart demonstrates how the loss decreases with increased number of parameters due to increased rank

Bibliography

[AZG20] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. "Intrinsic dimensionality explains the effectiveness of language model fine-tuning". In: *arXiv preprint arXiv:2012.13255* (2020).

[Boj+17] Ond rej Bojar et al. "Findings of the 2017 Conference on Machine Translation (WMT17)". In: *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 169–214. URL: http://www.aclweb.org/anthology/W17-4717.

[Cho+14] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[DCL21] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. "Attention is not all you need: Pure attention loses rank doubly exponentially with depth". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2793–2803.

[Gag94] Philip Gage. "A new algorithm for data compression". In: *The C Users Journal* 12.2 (1994), pp. 23–38.

[Hou+19] Neil Houlsby et al. "Parameter-efficient transfer learning for NLP". In: *International conference on machine learning*. PMLR. 2019, pp. 2790–2799.

[Hu+21] Edward J Hu et al. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[Hua] *Speaking Your Language: The Transformer in Machine Translation — blog.huawei.com*. https://blog.huawei.com/2022/02/01/speaking-your-language-transformer-machine-translation. [Accessed 26-01-2025].

[Kar] Andrej Karpathy. *x.com — x.com*. https://x.com/karpathy/status/1621578354024677377. [Accessed 26-01-2025].

[KV24] László Kozma and Johannes Voderholzer. "Theoretical Analysis of Byte-Pair Encoding". In: *arXiv preprint arXiv:2411.08671* (2024).

[LL21] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation". In: *arXiv preprint arXiv:2101.00190* (2021).

[SR] PhD Sebastian Raschka. *Understanding and Coding Self-Attention, Multi-Head Attention, Cross-Attention, and Causal-Attention in LLMs — magazine.sebastianraschka.com*. https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention. [Accessed 26-01-2025].

[Vas17] A Vaswani. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (2017).

[Voi20] Elena Voita. *NLP Course For You*. 2020. URL: https://lena-voita.github.io/nlp_course.html.

[Wan] Xupeng Wang. *Understand the role of FFNs in Transformers — marvelous_catawba_otter_200*. https://medium.com/@marvelous_catawba_otter_200/understand-the-role-of-ffns-in-transformers-51606a56e654. [Accessed 26-01-2025].