Heinrich Heine Universität Düsseldorf

Course: Implementing Transformers
Degree: M.Sc. AI and Data Science

Encoder–Decoder Transformer for German to English Translation

Hani Ouraghene
Heinrich Heine Universität Düsseldorf

January 24, 2026

1 Introduction

Transformer models replaced recurrence with attention-based sequence modeling and became the dominant paradigm for neural machine translation. In this project, I implemented an encoder–decoder transformer from scratch for German to English translation in a WMT-style setup, following the core ideas of [1] while focusing on understanding and implementing each component end-to-end: tokenization, masking, attention, optimisation, decoding, and evaluation.

Beyond reproducing a baseline architecture, several modern extensions were integrated. In particular, Rotary Position Embeddings (RoPE) [2] were implemented to inject relative positional information directly into self-attention. Furthermore, the standard position-wise feed-forward network was replaced by a SwiGLU variant [3], and beam search with length normalisation [4] was applied during inference. The report documents (i) architectural and training design choices, (ii) implementation details required for correctness and stability, and (iii) experimental results analysing the impact of model scale, data size, and inference-time strategies.

Empirically, scaling the model dimension and training data led to substantial improvements in translation quality. The best-performing configuration achieved a BLEU score of 21.3 on the test set. The experiments confirm that while architectural refinements improve stability and fluency, data scale and model capacity remain the dominant factors for performance.

The code is available at: GitHub repository

2 Method

2.1 Encoder–decoder transformer

The implemented model follows the standard encoder–decoder transformer architecture. The encoder processes the full source sentence $x_{1:S}$ and produces contextual representations $\mathbf{H} \in \mathbb{R}^{S \times d_{\text{model}}}$. The decoder generates the target sequence autoregressively, predicting y_t conditioned on $\mathbf{H}$ and previously generated tokens $y_{<t}$.

Each encoder layer consists of multi-head self-attention followed by a position-wise feed-forward network (FFN), with residual connections and layer normalisation applied after each sublayer. Each decoder layer contains masked self-attention, encoder–decoder cross-attention, and an FFN.

2.2 Scaled dot-product attention and masking

Given query Q, key K, and value V, scaled dot-product attention is defined as

$$\text{Attn}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V, \quad (1)$$

where M is an additive mask that assigns $-\infty$ to disallowed positions.

Two masking mechanisms are required:

- **Padding mask:** prevents attention to `<pad>` tokens introduced for batching.
- **Causal mask:** used in decoder self-attention to block access to future tokens, ensuring autoregressive behaviour.

2.3 Multi-head attention

Multi-head attention projects inputs into h subspaces and performs attention independently per head:

$$\text{MHA}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O, \quad (2)$$

where $\text{head}_i = \text{Attn}(QW_i^Q, KW_i^K, VW_i^V)$. This allows different heads to specialise in modelling complementary relationships, such as local alignment or long-range dependencies.

2.4 Feed-forward networks, residuals, and normalisation

The position-wise feed-forward network (FFN) is applied independently at each sequence position:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \quad (3)$$

This formulation corresponds to the standard ReLU-based FFN used in the original transformer.

SwiGLU feed-forward network. In addition to the standard FFN, a SwiGLU variant was implemented following [3]. SwiGLU replaces the ReLU activation with a gated mechanism that modulates the hidden representation:

$$\text{SwiGLU}(x) = (xW_1 + b_1) \odot \text{SiLU}(xW_2 + b_2), \quad (4)$$

where $\odot$ denotes element-wise multiplication and SiLU is the sigmoid linear unit activation. The gated structure

increases the expressive capacity of the FFN without increasing network depth. In practice, the hidden dimension is adjusted such that the overall parameter count remains comparable to the standard FFN.

Residual connections and layer normalisation. Residual connections combined with layer normalisation stabilise optimisation and enable deeper architectures:

$$\mathrm{LN}(x) = \gamma \odot \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta. \tag{5}$$

Layer normalisation is applied after each sublayer in a pre-normalisation configuration, improving training stability for deeper transformer stacks.

2.5 Positional information

Self-attention is permutation-invariant; therefore, explicit positional information is required to encode token order.

2.5.1 Sinusoidal positional encodings

The original transformer uses fixed sinusoidal encodings added to token embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{6}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right). \tag{7}$$

These encodings allow relative offsets to be expressed as linear transformations.

2.5.2 Rotary Position Embeddings

RoPE [2] applies position-dependent rotations to query and key vectors. For position m,

$$f(x_m, m) = R_m x_m, \tag{8}$$

where R_m is a block-diagonal rotation matrix. This yields attention dot-products that depend on relative distance:

$$q_m^\top k_n = x_m^\top R_{m-n} x_n. \tag{9}$$

RoPE was applied to self-attention layers in both encoder and decoder but not to cross-attention.

3 Implementation Details

3.1 Code structure and reproducibility

The project is organised into modular components covering modelling, data preprocessing, tokenization, training utilities, and evaluation scripts. The training loop includes gradient clipping and experiment tracking via Weights & Biases.

3.2 Dataset pipeline and cleaning

The dataset consists of German–English sentence pairs. Preprocessing removes noisy samples, applies length filtering, and eliminates URLs and HTML artefacts using a whitelist-based character filter.

3.3 Tokenization

Byte-Pair Encoding (BPE) is used to handle open-vocabulary translation. Special tokens include `<pad>`, `<s>`, `</s>`, and `<unk>`. During batching, targets are split into decoder input (BOS-prefixed) and decoder output (EOS-suffixed), enabling teacher forcing.

3.4 Parameter sharing

Input embedding weights are shared with the output projection layer, reducing the number of parameters and improving generalisation.

3.5 Loss and optimisation

Training minimises cross-entropy loss with label smoothing ($\varepsilon = 0.1$), ignoring padding tokens. Optimisation uses AdamW with decoupled weight decay and the Noam learning rate schedule:

$$lr = d_{\text{model}}^{-0.5} \min\left(step^{-0.5}, step \cdot warmup^{-1.5}\right). \tag{10}$$

3.6 Inference

Translation is performed autoregressively, generating the target sequence token by token conditioned on the source sentence and all previously generated target tokens.

Greedy autoregressive decoding. In greedy decoding, the model selects at each time step the token with the highest conditional probability:

$$y_t = \arg\max_y \; p(y \mid y_{<t}, x). \tag{11}$$

This procedure is simple and computationally efficient, as only a single hypothesis is maintained throughout decoding. However, greedy decoding makes locally optimal decisions and cannot revise earlier choices. As a result, it is prone to suboptimal global translations, premature sentence termination, and reduced fluency, particularly for longer sequences.

Beam search decoding. Beam search [4] addresses these limitations by maintaining a fixed number K of partial hypotheses (the beam width) at each decoding step. Instead of committing to a single most probable token, beam search expands all current hypotheses with their top-K next-token candidates and retains the K sequences with the highest cumulative log-probability:

$$\text{score}(y_{1:t}) = \sum_{i=1}^{t} \log p(y_i \mid y_{<i}, x). \tag{12}$$

This allows the decoder to explore multiple plausible translations in parallel and reduces the risk of early irreversible errors.

Length normalisation. Without correction, beam search tends to favour shorter sequences due to the accumulation of negative log-probabilities. To counteract this bias, length normalisation is applied:

$$\text{score}_{\text{norm}}(y) = \frac{1}{(5+|y|)^{\alpha}} \sum_{t=1}^{|y|} \log p(y_t \mid y_{<t}, x), \tag{13}$$

where $|y|$ denotes the sequence length and α is a hyperparameter controlling the strength of normalisation. This encourages complete translations and improves fluency.

Beam search is applied only at inference time and does not alter the training objective, which remains token-level cross-entropy with teacher forcing. In the conducted experiments, beam search significantly improved translation quality compared to greedy decoding, particularly for longer sentences.

3.7 Runtime considerations

All experiments were conducted on GPU hardware. Training was performed on an NVIDIA GeForce RTX 5090 GPU with 32GB RAM. One full training epoch over the complete training set of approximately 5.9 million sentence pairs required about three hours.

Training on CPU was found to be impractical for larger configurations, while GPU acceleration enabled stable optimisation and reasonable turnaround times for experimentation. Inference latency increased moderately when using beam search compared to greedy decoding, but remained acceptable for offline evaluation.

4 Experiments and Results

This section reports quantitative and qualitative results for the German–English translation task. The experiments investigate the effects of model scale, training data size, and inference-time strategies. In addition to scaling experiments, three targeted extensions were evaluated as part of this project: (i) Rotary Position Embeddings (RoPE) in self-attention, (ii) a SwiGLU-based feed-forward network replacing the standard FFN, and (iii) beam search decoding with length normalisation. All three modifications were implemented from scratch and integrated into the baseline transformer architecture described in the previous sections.

4.1 Experimental setup

All models are trained using teacher forcing with cross-entropy loss and label smoothing. Evaluation is performed on a held-out test set of 2,500 sentence pairs using BLEU.

4.2 Model configurations

Three configurations were evaluated. Model 0 serves as a small baseline, while Models 1, 2 and 3 increase model dimension and training data size.

4.3 Model development and experimental progression

The sequence of models was developed iteratively, guided by empirical observations on data scale, model capacity, and convergence behaviour. Rather than fixing a single configuration upfront, successive models were trained to test specific hypotheses about the limiting factors of translation quality.

The initial baseline (Model 0) was trained on a small subset of approximately 50,000 sentence pairs for five epochs using a reduced model dimension. Despite extended training, the model failed to learn meaningful translations and achieved a BLEU score close to zero. This indicated that insufficient data, rather than optimisation time, was the primary bottleneck.

Table 1: Model architectures and training configurations (ordered by increasing BLEU score).

	Model 0	Model 1	Model 2	Model 3
Architecture				
d_{model}	256	512	512	512
Attention heads	8	8	8	8
Encoder layers	6	6	6	6
Decoder layers	6	6	6	6
FFN dimension	1024	2048	2048	2048
Dropout	0.1	0.1	0.1	0.1
Max sequence length	128	128	128	128
Vocabulary size	32k	32k	32k	32k
Parameters	31.8M	73.1M	73.1M	73.1M
Training				
Training samples	0.05M	2.0M	5.0M	5.9M
Validation samples	2.5k	2.5k	2.5k	2.5k
Test samples	2.5k	2.5k	2.5k	2.5k
Epochs	5	3	2	5
Batch size	32	32	32	32
Learning rate	1.0	1.0	1.0	1.0
Warmup steps	4000	4000	4000	4000
Gradient clipping	1.0	1.0	1.0	1.0
Weight decay	0.01	0.01	0.01	0.01
Min / Max length filter	5 / 100	5 / 100	5 / 100	5 / 100
Decoding				
Positional encoding	RoPE	RoPE	RoPE	RoPE
FFN variant	ReLU	SwiGLU	SwiGLU	SwiGLU
Decoding strategy	Greedy	Beam search	Beam search	Beam search
Result				
BLEU (test)	0.30	18.86	20.92	**21.34**

Based on this observation, both model capacity and dataset size were increased. Model 1 was trained with $d_{\text{model}} = 512$ on approximately 2 million sentence pairs for three epochs, resulting in a substantial improvement to 18.9 BLEU. This confirmed that increased representational capacity and data scale are critical for effective neural machine translation.

To further investigate the impact of data scale, Model 2 was trained on a larger subset of 5 million sentence pairs. After only two epochs, this configuration already outperformed Model 1, suggesting faster convergence and better generalisation when trained on a broader data distribution.

Finally, the full training set of approximately 5.9 million sentence pairs was used to train Model 3 for five epochs. This configuration achieved the best performance with a BLEU score of 21.3 and serves as the final model reported in this work. The progression from Model 0 to Model 3 illustrates that data scale and sufficient training time dominate performance, while architectural refinements primarily improve stability and inference quality.

4.4 Architectural and inference improvements

All larger models incorporate RoPE, SwiGLU feed-forward networks, and beam search decoding. RoPE proved stable but did not yield large BLEU gains in isolation. SwiGLU improved training stability and fluency, while beam search substantially improved translation quality during inference.

4.5 Qualitative analysis

Qualitative examples show that the model captures core semantic content and produces mostly grammatical translations. Common errors include mistranslated named entities, literal translations of idioms, and occasional semantic drift in long sentences.

For more details: GitHub repository (results)

5 Discussion

Several factors were critical for stable optimisation: correct masking, learning rate warmup, gradient clipping, and label smoothing. RoPE did not significantly improve BLEU compared to sinusoidal encodings but provided robust positional behaviour across sequence lengths. The experiments highlight that architectural refinements alone cannot compensate for insufficient data or model capacity.

Error analysis reveals recurring issues with named entities, idiomatic expressions, and long-distance dependencies. These limitations are consistent with medium-scale transformer models trained from scratch without domain-specific adaptation.

6 Conclusion

This project implemented an encoder–decoder transformer from scratch for German→English translation, covering the full pipeline from preprocessing and tokenization to attention mechanisms, optimisation, and inference. Modern extensions including Rotary Position Embeddings, SwiGLU feed-forward networks, and beam search decoding were successfully integrated.

The best-performing configuration achieved a BLEU score of 21.3 on the test set. Results demonstrate that model scale and data size dominate translation performance, while architectural improvements primarily enhance stability and fluency. Overall, the project provides a comprehensive and transparent evaluation of transformer-based machine translation implemented from first principles.

A Additional Plots

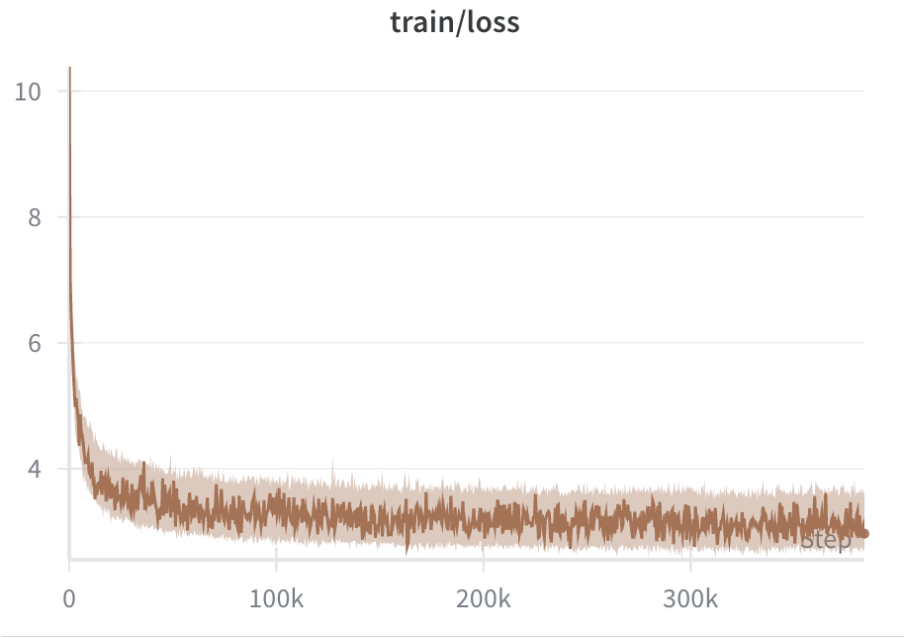

Figure 1: Model 3 train loss

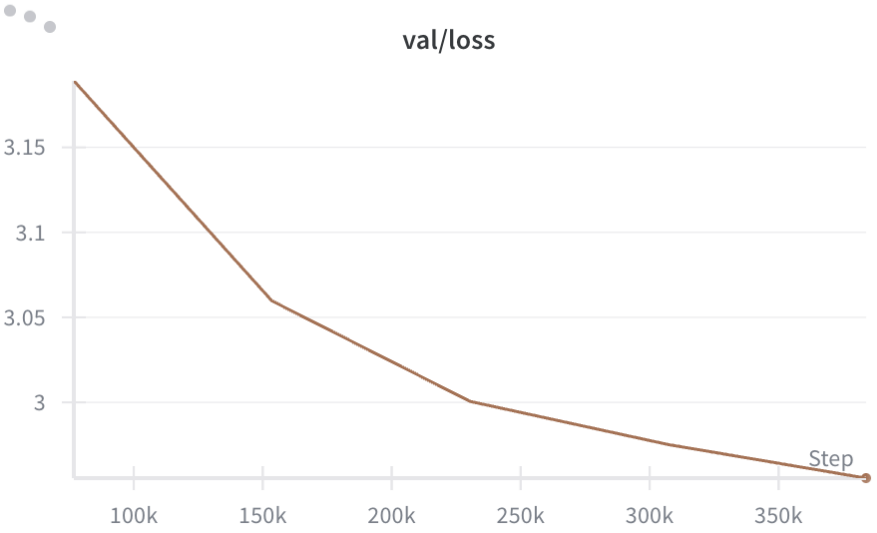

Figure 2: Model 3 val loss

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[2] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," 2023.

[3] N. Shazeer, "Glu variants improve transformer," 2020.

[4] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," in *Proceedings of the First Workshop on Neural Machine Translation*, p. 56–60, Association for Computational Linguistics, 2017.