Implementing Transformer Models

Modar Alkanj

06.02.2026

Supervisors:
Dr. Carel van Niekerk
Dr. Hsien-Chin Lin

1 Introduction

Transformers have fundamentally reshaped natural language processing (NLP) by replacing sequential architectures such as recurrent neural networks with highly parallelizable attention-based mechanisms. This architectural shift enables Transformers to process entire sequences simultaneously while effectively modeling long-range dependencies between words. As a result, Transformer models have achieved state-of-the-art performance in tasks including machine translation, text generation, and text summarization.

Despite their success, the training and fine-tuning of large Transformer models remain computationally demanding, which restricts their applicability in resource-constrained environments. This work investigates the implementation of a Transformer model from scratch with a focus on efficiency and adaptability. In particular, we explore the use of *FlashAttention* [1], an optimized attention algorithm designed to accelerate training and reduce memory and computational overhead. By examining the model architecture, training process, and attention optimization techniques, this study provides insights into design choices that impact both performance and computational efficiency.

2 Transformer Architecture

Introduced by Vaswani et al. [3], the Transformer consists of an encoder and a decoder (Figure 1). The encoder converts the input sequence into contextual embeddings, while the decoder generates the target sequence autoregressively, one token at a time.

The model relies on attention rather than recurrence. The encoder stacks layers of multi-head self-attention and feed-forward networks to capture local and global dependencies. The decoder uses masked self-attention to prevent future token access and cross-attention to attend to encoder outputs.

Decoder outputs are mapped to vocabulary scores via a linear layer and normalized with a Softmax. Tokens are typically generated by selecting the highest-probability prediction at each step (greedy decoding).

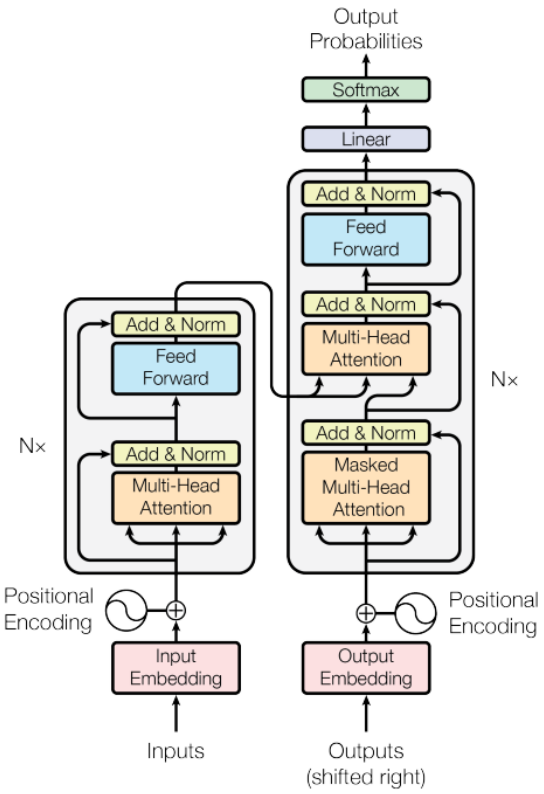

Figure 1: The Transformer - model architecture.

Figure 1: Overview of the architecture of the transformer model [3]

2.1 Text Preprocessing for Transformers

Transforming raw text into a format suitable for processing by a Transformer model involves three essential stages: **tokenization**, **embedding**, and **positional encoding**. Together, these steps convert textual input into

numerical representations that the model can interpret effectively.

2.1.1 Tokenization with BPE

The first step is tokenization, which in this work is performed using the **Byte Pair Encoding (BPE) tokenizer**. BPE operates by iteratively identifying the most frequent pair of symbols in the text and merging them into a new symbol, compressing the text while preserving its semantic content. Both the new symbols and the original pairs are stored in a lookup table, enabling reconstruction of the original text.

Example: Consider the sentence:

"Machine learning is a subset of artificial intelligence."

After tokenization using BPE, it could be represented as the following sequence of token IDs:

```
[1, 15270, 853, 8115, 136, 78, 2629, 157, 124, 14502, 13272, 2]
```

The vocabulary size determines the number of new symbols and controls compression efficiency. The used vocabulary size in this work is 50000.

2.1.2 Embedding

Once tokenized, each token is mapped to a dense vector representation using an **embedding layer**. This layer assigns high-dimensional embeddings to token indices, capturing semantic relationships between tokens. Formally, given a token x, its embedding is retrieved as:

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

where E denotes the embedding matrix.

2.1.3 Positional Encoding

Since Transformers lack a notion of word order, **positional encodings** are added to token embeddings to incorporate sequence information. A common choice is the **sinusoidal encoding** [3]:

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

The input to the Transformer is the element-wise sum of token embeddings and positional encodings:

$$h_0 = \text{Embedding}(x) + \text{PositionalEncoding}(x)$$

These encodings alternate sine and cosine across dimensions and have three key properties:

1. Linear Offset Representation: Using trigonometric angle addition formulas, the encoding at a shifted position can be expressed as a linear combination of the original encoding:

$$\sin \frac{\text{pos}+k}{\lambda_i} = \sin \frac{\text{pos}}{\lambda_i} \cos \frac{k}{\lambda_i} + \cos \frac{\text{pos}}{\lambda_i} \sin \frac{k}{\lambda_i}, \quad \cos \frac{\text{pos}+k}{\lambda_i} = \cos \frac{\text{pos}}{\lambda_i} \cos \frac{k}{\lambda_i} - \sin \frac{\text{pos}}{\lambda_i} \sin \frac{k}{\lambda_i}.$$

This shows $PE_{\text{pos}+k}$ is a linear transformation of PE_{pos}.

2. Geometric Wavelength Spacing: Wavelengths span 2π to $2\pi \cdot 10000$ in geometric progression, ensuring the model captures positional relationships at multiple scales. The ratio between consecutive wavelengths is approximately constant.

3. Extrapolation Capability: Since encodings are continuous functions of position, unseen positions can be computed via the same linear transformations, unlike learned embeddings which are fixed to training positions.

2.2 Attention Mechanism

The attention mechanism addresses key limitations of earlier sequence models by enabling dynamic focus on relevant input elements, improving parallelization, and avoiding information loss caused by compressing sequences into a single vector. It operates on three learned projections of the input embeddings X: Queries (Q), Keys (K), and Values (V), defined as

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

Scaled dot-product attention is computed as

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

where dot products measure query–key similarity, scaling by $\sqrt{d_k}$ stabilizes training, and the Softmax highlights the most relevant tokens. The resulting weights are used to aggregate the value vectors.

Depending on the source of Q, K, and V, attention can be categorized as *self-attention*, where all projections originate from the same sequence, or *cross-attention*, where queries attend to a different sequence, such as between encoder and decoder.

2.2.1 Multi-Head Attention

Multi-head attention extends the core mechanism by computing multiple attention operations in parallel over different representation subspaces. The outputs of all heads are concatenated and projected back to the original dimensionality, enabling the model to capture diverse and long-range dependencies.

2.2.2 Masking

Masking ensures that attention is applied only to valid tokens. *Padding masks* prevent artificial padding tokens, introduced to equalize sequence lengths, from influencing attention scores. *Future masks* are used during autoregressive decoding to block access to future tokens by assigning their attention scores a value of $-\infty$, ensuring that predictions depend only on previously generated tokens 2.

Figure 2: future masking [2]

2.3 Residual Connections and Layer Normalization

Residual connections and layer normalization are essential for stable training in deep Transformer models. Residual connections preserve gradient flow and prevent representation degradation by adding the sublayer output to its input:

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

Layer normalization stabilizes activations by normalizing features within each layer:

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

Together, these mechanisms reduce vanishing gradients and accelerate convergence.

2.4 Position-Wise Feed-Forward Network

Each Transformer layer includes a position-wise feed-forward network (FFN) that processes tokens independently using two linear transformations with a ReLU activation:

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

The FFN expands the representation to a higher-dimensional space (typically four times the model size) before projecting it back. While attention captures global dependencies, the FFN introduces nonlinearity, allowing the model to learn richer local feature representations.

2.5 FlashAttention

2.5.1 Problem Motivation

Self-attention in Transformers has $\mathcal{O}(N^2)$ time and memory complexity with respect to the sequence length N. While many prior works reduce FLOPs (Floating Point Operations) using approximate attention, they often fail to reduce runtime in practice because they ignore the cost of memory accesses between different levels of GPU memory.

Modern GPUs are *memory-bound*: computation is fast, but reading and writing from High Bandwidth Memory (HBM) is slow compared to on-chip SRAM. Standard attention implementations materialize the full attention matrix $QK^\top \in \mathbb{R}^{N\times N}$ in HBM, causing excessive memory traffic.

2.5.2 Key Idea: IO-Aware Attention

FlashAttention proposes an *IO-aware* exact attention algorithm that minimizes reads and writes to HBM. The core principle is:

> *Avoid materializing the $N \times N$ attention matrix in HBM by computing attention in blocks that fit in SRAM.*

This is achieved using:

- **Tiling (blocking)** of Q, K, V
- **Incremental Softmax Computation**
- **Recomputation** instead of storing intermediates for backpropagation
- **Kernel fusion** into a single CUDA kernel

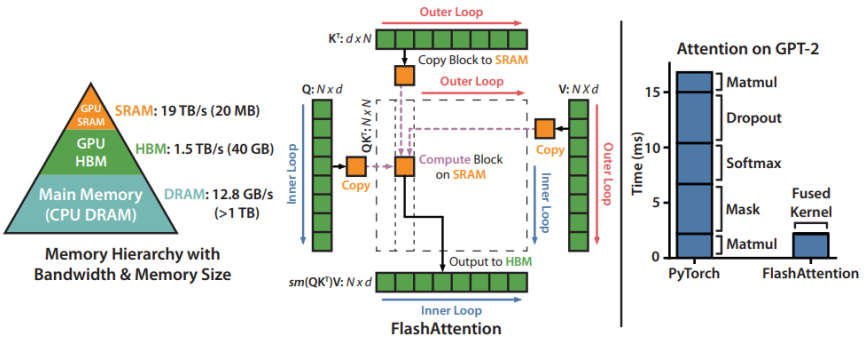

Figure 3: General architecture of FlashAttention [1]

2.5.3 FlashAttention Algorithm

FlashAttention computes the same output:

$$O = \text{softmax}(QK^\top)V$$

but never stores $S = QK^\top$ or $P = \text{softmax}(QK^\top)$ in HBM.

Tiling The matrices are split into blocks:

$$Q = \{Q_i\}, \quad K = \{K_j\}, \quad V = \{V_j\},$$

where each block fits into SRAM.

Incremental Softmax Computation To avoid materializing the full attention matrix and ensure numerical stability, FlashAttention computes softmax incrementally over blocks of $QK^\top$.

For a vector x, standard softmax is:

$$\text{softmax}(x)_i = \frac{e^{x_i - m}}{\sum_k e^{x_k - m}}, \quad m = \max_k x_k.$$

When processing blocks, let x be split into consecutive blocks $x^{(1)}, x^{(2)}, \ldots$. FlashAttention maintains for each row i:

- m_i: running maximum over all processed blocks
- ℓ_i: running normalization constant (cumulative sum of exponentials adjusted for the running maximum)

Suppose a new block $x^{(b)}$ is processed. The running maximum and normalization are updated as:

$$m_i \leftarrow \max(m_i, \max_k x_{i,k}^{(b)}), \qquad \ell_i \leftarrow \ell_i \cdot e^{m_i^{\text{old}} - m_i} + \sum_k e^{x_{i,k}^{(b)} - m_i}.$$

The softmax for elements in the current block is then computed using these updated m_i and ℓ_i, ensuring that the overall softmax across all blocks is exact and numerically stable, without storing the full vector.

This allows softmax to be computed exactly across blocks without ever seeing the full vector.

Recomputation Instead of storing S or P for the backward pass, FlashAttention stores:

$$O, \quad m, \quad \ell.$$

During backpropagation, attention scores are recomputed on-the-fly in SRAM. Although this increases FLOPs, it *reduces HBM access*, resulting in faster execution.

3 Training

We trained a base Transformer model with the following configuration: $vocab\_size$ = 50,000, $d_{\text{model}} = 512$, 8 attention heads, 6 encoder layers, 6 decoder layers, a feed-forward dimension of $d_{\text{ff}} = 2048$, dropout rate 0.1, and a maximum sequence length of 64 tokens.

We employed a warmup learning rate schedule as proposed in the original Transformer paper [3]:

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

using $d_{\text{model}} = 512$ and a warmup of 4000 steps. For optimization, we used the AdamW optimizer with learning rate $\alpha = 0.5$ and parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$, which provided stable convergence while preserving standard Transformer training dynamics. The model has in total approximately $70M$ parameters.

The base model was trained on a subset of the WMT17 German-English dataset, using 1 M datapoints for training and 200,000 for testing, for the task of machine translation from German to English.

The model was trained on an NVIDIA RTX 3060 GPU for 4 epochs with a batch size of 128, resulting in a total of 36,952 training steps.

The training employed teacher forcing, a technique in which the model is provided with the ground-truth previous token at each step to predict the next token, rather than relying on its own predictions. This approach accelerates convergence and stabilizes training, especially in autoregressive sequence generation tasks such as machine translation.

4 Results

The plots in figure 4 show that the model was not overfitting since both train and validation loss were decreasing.

We also computed the BLEU score on the entire validation dataset and obtained a score of **27.8**.

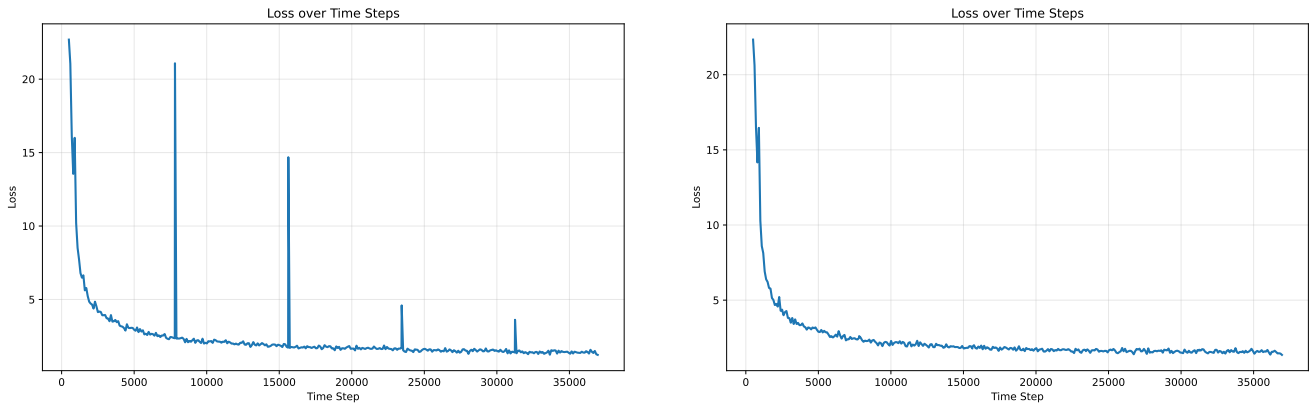

Figure 4: Training(left) and validation(right) loss curves

In some cases, the model exhibited repetitive word generation within a sentence. This behavior results from greedy decoding, where the token with the highest probability is selected at each step. To mitigate this issue, we implemented beam search and introduced a repetition penalty during decoding. This approach reduced repetitive outputs and improved overall translation quality. The BLEU score reported above was computed using beam search decoding.

To gain qualitative insight into the model's translation behavior, we randomly sampled a batch of 32 sentences from the validation dataset and generated translations using the base model. Table 1 presents a comparison between the best and worst generated translations.

Case	Example
Best	**BLEU:** 75.98 **SRC:** von wem sonst, wenn nicht von der europäischen union? **HYP:** what else, if not by the european union? **REF:** by whom, if not by the european union?
Worst	**BLEU:** 8.25 **SRC:** der vorschlag schöpft alle möglichkeiten aus, um das europäische parlament mit den kontrollrechten auszustatten, die ihm zustehen. **HYP:** the proposal offers all the possibilities for the european parliament to provide itself with the rights of control that it has to offer. **REF:** the proposal indeed goes as far as possible towards granting the european parliament the control rights that, as a co-legislator, it should possess.

Table 1: Best and worst translations from a random validation batch.

4.1 Comparison between FlashAttention and Standard Attention

To evaluate the efficiency of FlashAttention, we conducted experiments using the same model parameters as before, increasing the batch size to 512 and training on 80,000 datapoints due to limited computational resources. The model was trained for 5 epochs on an NVIDIA A100 GPU. For each run, we recorded the average time per batch, total training time, and memory usage. The experiment was repeated twice: once using standard attention and once using FlashAttention, allowing for a direct comparison of performance and resource utilization.

Attention Type	Avg. Time per Batch (s)	Total Training Time (s)	Memory Usage (GB)	Validation Loss (final epoch)
Standard Attention	0.98	931.00	46	16.15
FlashAttention	0.168	368.61	32	14.38

Table 2: Comparison of standard attention and FlashAttention in terms of runtime, memory usage, and validation loss.

Table 2 demonstrates that FlashAttention substantially improves training efficiency compared to standard attention. The average time per batch is reduced from 0.98 s to 0.168 s, and the total training time decreases from 931 seconds to 368.6 seconds. Memory usage is also lowered from 46 GB to 32 GB. Importantly, these efficiency gains are achieved without degrading model performance, as the validation loss at the final epoch is slightly improved when using FlashAttention. Overall, the results highlight the significant computational and memory advantages of FlashAttention, making it particularly valuable for large models and resource-constrained environments.

5 Conclusion

In this work, we implemented a Transformer model from scratch and analyzed its core architectural components, training dynamics, and performance on a machine translation task. Using the WMT17 German–English dataset, the base model achieved a BLEU score of **27.8**, demonstrating that a carefully designed and scaled Transformer can produce competitive results even under constrained computational resources. Qualitative analysis further highlighted both the strengths and limitations of greedy decoding, motivating the use of beam search with repetition penalties to improve translation quality.

A central contribution of this study was the exploration of *FlashAttention* as an optimized alternative to standard attention. Experimental results showed that FlashAttention significantly reduced training time and memory consumption while maintaining—and slightly improving—model performance in terms of validation loss. These findings confirm that IO-aware attention mechanisms can yield substantial practical benefits on modern GPU architectures, particularly for large batch sizes and long sequences.

Overall, this report demonstrates that architectural efficiency and system-level optimizations are critical for scalable Transformer training. FlashAttention provides a promising direction for reducing computational overhead without sacrificing accuracy, making Transformer models more accessible in resource-limited settings.

References

[1] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2205.14135.

[2] Sreedath Panat. Why do we need "masking" in attention? `https://www.vizuaranewsletter.com/p/why-do-we-need-masking-in-attention`, 2025.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Word count: *1934*