Computional Linguistic Task 4 - report

Jerzy Boksa

October 2025

1 Introduction

The goal of this assignment is to compare several modern memory-optimization techniques used during Transformer training. Using the same dataset, same model architecture, and same hyperparameters (except for memory-related settings), your task is to measure and analyze how different optimization techniques influence:

- GPU memory usage
- Maximum batch size that fits into memory
- Training speed (time per step and total time for 1 epoch)
- Final model performance (perplexity after 1 epoch)

The report presents a comparative analysis of four attention-based models trained for this study:

- standard attention mechanism,
- bfloat16-optimized model,
- FlashAttention implementation,
- windowed FlashAttention variant.

The evaluation focuses on their computational efficiency, associated trade-offs, and the overall quality of the resulting training dynamics.

2 Experimental Setup

In this section, the experimental setup is described in detail. Specifically, present the dataset and tokenizer employed, the hardware configuration, a description of the accompanying files, and the software libraries utilized in the experiments.

2.1 Dataset

The dataset employed in this experiment is the `wolne_lektury_corpus` from Speakleash [1], which comprises 6,619 Polish books, poems, and other literary texts. The corpus was preprocessed to eliminate non-essential metadata, such as authors' names and publication dates occurring at the beginning of the poems. For computational efficiency, a subset of the corpus was used: 1,080 documents for training and 120 documents for evaluation.

2.2 Hardware

Since the FlashAttention mechanism [2] requires a specific class of GPU, the experiment will be conducted on the athena.cyfonet.pl host equipped with an NVIDIA A100 GPU featuring 40 GB of VRAM.

2.3 Tokenizer

For the tokenization process, the Polish GPT-2 Small tokenizer was used [3]. This tokenizer is based on the Byte-Pair Encoding (BPE) algorithm.

2.4 Implementation and Libraries

The complete implementation is available in the project repository. The core of the training pipeline and Transformer model was built using the PyTorch framework[4]. The architecture follows the implementation developed for the first task, with necessary extensions for the experiments conducted in this report. For models utilizing FlashAttention, the `flash-attn` package [5] was integrated to simplify the implementation and significantly reduce the overhead of custom attention kernels.

The repository includes the following files:

- `transformer.py` — implementation of all model variants used in the experiments,
- `training.py` — training loop, metric logging, and memory profiling using PyTorch utilities,
- `run.py` — execution script for launching full training sessions for selected model configurations,
- `dataset.py` — dataset loading, preprocessing, and dataloader construction.
- `measure_gpu.py` - a script for measuring GPU stats using fixed batch size.

Additionally, the repository contains a `results` directory with training plots, measured performance metrics, and a summary file aggregating the key findings.

2.5 Models Description

Four model variants were evaluated in this study:

1. a standard Transformer model using `float32` precision with a standard attention mechanism,
2. a standard Transformer model using `bfloat16` precision with standard attention,
3. a `bfloat16` model employing the FlashAttention mechanism,
4. a `bfloat16` model utilizing windowed FlashAttention.

All models share the same architectural configuration, with the only differences arising from the attention mechanism and numerical precision. The unified model hyperparameters are as follows:

- `seq_len` — 128,
- `embedding_dim` — 512,
- `blocks_count` — 8,
- `num_heads` — 8.

Due to varying GPU memory requirements across the attention mechanisms, the batch size differs between models; these details are discussed in the Training section.

3 Training

All models were able to learn meaningful structural patterns from the dataset, despite its relatively small size. The primary objective of the training analysis is therefore not to maximize absolute model performance, but to investigate how the choice of attention mechanism and numerical precision (`dtype`) influences the training dynamics, stability, and efficiency.

3.1 Parameters

To maximize GPU utilization, the optimal batch size was determined individually for each model. Due to differences in numerical precision (`dtype`) and the employed attention mechanism, GPU memory consumption varies, which directly affects the feasible batch size. Table 1 summarizes the resulting batch sizes for each model configuration.

Table 1: Batch sizes and training steps for different model configurations

Model	**Batch Size**	**Training Steps Performed**
Standard Transformer (float32, standard attention)	201	419
Standard Transformer (bfloat16, standard attention)	446	189
Transformer (bfloat16, FlashAttention)	489	172
Transformer (bfloat16, Windowed FlashAttention)	489	172

It is evident that the batch size is inversely correlated with the number of training steps performed: models trained with larger batch sizes completed fewer training steps within the same epoch.

Furthermore, several common hyperparameters were applied across all models:

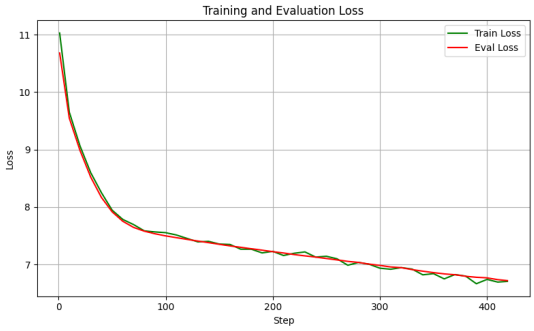
- `learning_rate` set to 10^{-4},
- `epochs` — only a single epoch was performed,
- `dropout` — 0.15,
- `window_size` — 0.8 (only applicable to the windowed FlashAttention model).

To ensure some GPU memory margin and avoid out-of-memory errors, batch sizes were set to utilize approximately 90% of the maximum available GPU VRAM, leaving a 10% buffer.

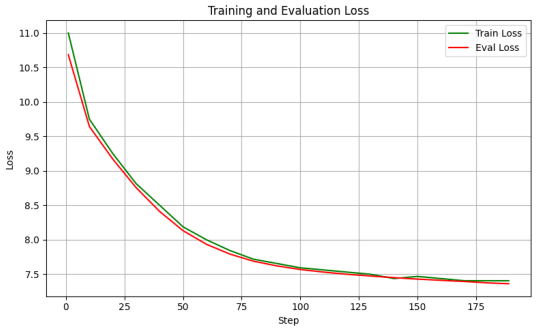
3.2 Results

As mentioned earlier, all models were able to reduce both the training and evaluation loss. The results of the experiments are summarized below:

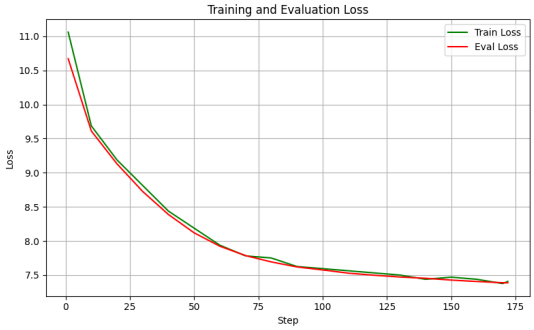
Table 2: Training and evaluation loss curves for different model configurations



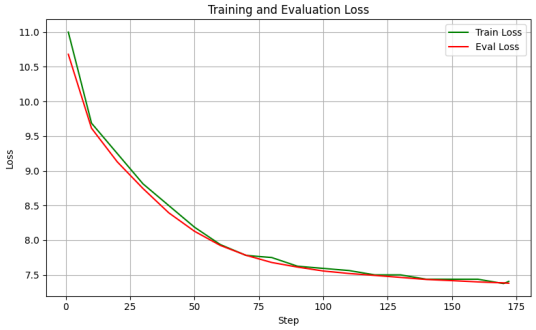
(a) Standard Transformer (float32)



(b) Standard Transformer (bfloat16)



(c) FlashAttention (bfloat16)



(d) Windowed FlashAttention (bfloat16)

To provide a clearer comparison, Table 3 summarizes key training metrics for all evaluated models. The table includes the average training time per epoch, the final evaluation loss, the final perplexity calculated after training, and the average batch calculation time used during training.

Table 3: Training metrics for different model configurations

Model	**Avg Epoch Time (m)**	**Final Eval Loss**	**Final Perplexity**	**Avg Batch Time (s)**
Standard Transformer (float32)	12.04	**6.72**	**13.63**	0.30
Standard Transformer (bfloat16)	1.44	7.36	56.45	0.16
FlashAttention (bfloat16)	**0.99**	7.39	64.36	0.02
Windowed FlashAttention (bfloat16)	**0.99**	7.38	63.86	**0.019**

The best-performing model in terms of evaluation metrics is the Standard Transformer (float32). However, its training time was the longest—more than 11 times longer than the other models - although it achieved the lowest perplexity and final evaluation loss. Additionally, the average batch duration was the highest for this model, reflecting both the smaller batch size and the computational overhead of full attention in float32.

It should be noted that this does not necessarily mean the Standard model is unequivocally superior. The other models performed better considering the full training time for one epoch, completed training approximately 11 times faster. If these models were trained for a longer duration, they would surpass the Standard model in terms of evaluation loss and perplexity.

It is worth highlighting the average batch time, as it demonstrates that while the Standard model has the longest batch time, the smaller batch size limits its throughput. In contrast, the FlashAttention-based models processed larger batches much faster, performing more computations in less time. This efficiency gain comes from the FlashAttention mechanism, which significantly reduces computation time, although these models slightly underperformed in training metrics compared to the Standard model.

The difference between the standard FlashAttention and Windowed FlashAttention models is minimal at this scale, but for larger-scale training, the windowed variant may provide additional efficiency benefits without substantially affecting training quality.

3.3 Comment on Training

The most important factor differentiating these models is the choice of `dtype`, which significantly reduces the training time. Additionally, the introduction of the FlashAttention mechanism contributed to a substantial decrease in training time, reducing the average epoch duration from 1.44 m to 0.99 m. The windowed variant did not provide noticeable benefits at this scale, although it may have a greater impact in larger-scale experiments.

Batch size also plays an important role. It is likely that smaller batch sizes for FlashAttention models could further improve performance, as the number of training steps performed was relatively low compared to the Standard model.

Overall, for larger-scale training, the combination of FlashAttention and `bfloat16` precision appears to be a highly effective configuration.

4 Used GPU Memory

In this section, we focus on inference time and GPU memory utilization for the different models.

4.1 GPU Memory with Optimal Batch Size

GPU memory was measured during training. However, peak memory is considered a less critical metric here, as the batch size was automatically adjusted to utilize approximately 90% of the available GPU VRAM. The following table summarizes different aspects of memory usage:

- **Avg Before GPU Mem**: average memory allocated before the forward pass,
- **Avg Forward GPU Mem**: average memory used during the forward pass,
- **Avg Backward GPU Mem**: average memory used during the backward pass,
- **Avg Peak GPU Mem**: average peak memory allocated during training.

Table 4: GPU memory utilization for different model configurations (in MB)

Model	**Avg Before GPU Mem**	**Avg Forward GPU Mem**	**Avg Backward GPU Mem**	**Avg Peak GPU Mem**
Standard Transformer (float32)	1219	17132	6316	27000
Standard Transformer (bfloat16)	619	18250	6267	29419
FlashAttention (bfloat16)	619	19033	6829	31280
Windowed FlashAttention (bfloat16)	619	19033	6829	31280

As mentioned before, the values presented above correspond to the maximum memory usage for the optimal batch size. For this reason, there are no significant differences in the forward and backward memory consumption across the models. The main differences are visible in the memory allocated before the forward pass. To better illustrate these differences, Table 5 presents the memory usage for a constant batch size across all models.

4.2 GPU Memory with Constant Batch Size

To better illustrate the differences in GPU memory usage, we measured the memory consumption over 10 evaluation steps with a constant batch size of 128 for all models. One step corresponds to a single pass of the evaluation dataloader, and the GPU memory usage was averaged per batch. The results are summarized in Table 5.

Table 5: Average GPU memory usage for constant batch size of 128 (in MB)

Model	**Avg GPU Memory Used**
Standard Transformer (float32)	10128
Standard Transformer (bfloat16)	5067
FlashAttention (bfloat16)	5066
Windowed FlashAttention (bfloat16)	5066

As observed with a constant batch size, the `bfloat16` models utilized approximately half of the GPU memory compared to the Standard Transformer (float32). Moreover, the addition of the FlashAttention mechanism resulted in only a minor further reduction in memory usage compared to the `bfloat16` model without FlashAttention. This is probably because of the `seq_len` and for higher values the difference would be more significant.

4.3 Inference Time

One of the most important metrics for language models is inference time, as it directly impacts how quickly users receive responses. Inference time was measured on the evaluation datasets, with 10 repeated runs for each model. The average time per batch was then calculated using a constant batch size of 128. The results are summarized in Table 6.

Table 6: Average inference time per batch (batch size = 128) for different model configurations (in seconds)

Model	**Avg Batch Evaluation Time (s)**
Standard Transformer (float32)	0.12
Standard Transformer (bfloat16)	0.03
FlashAttention (bfloat16)	0.02
Windowed FlashAttention (bfloat16)	0.02

As observed, the `bfloat16` models outperform the Standard Transformer in terms of inference time. Moreover, the FlashAttention models are slightly faster than the Standard `bfloat16` model. Although the difference is not significant for the current sequence length ($seq\_len = 128$), it is expected that the advantage of FlashAttention would become more pronounced for longer sequences.

5 Conclusion

The conducted experiments demonstrate that the choice of numerical precision (`dtype`) is the most influential factor affecting both training efficiency and GPU memory utilization. Switching from `float32` to `bfloat16` resulted in a substantial reduction in memory consumption, enabling significantly larger batch sizes. This, in turn, led to faster training and more efficient GPU utilization without critically degrading model performance.

The introduction of the FlashAttention mechanism further improved computational efficiency, particularly in terms of training and inference speed. Although the memory savings introduced by FlashAttention were relatively modest for the chosen sequence length (`seq_len` = 128), the reduction in average batch processing time was clearly observable. This confirms that FlashAttention is especially beneficial for throughput-oriented training setups and is expected to provide even greater advantages for longer sequences and larger-scale datasets.

The windowed FlashAttention variant did not show noticeable improvements over the standard FlashAttention implementation in this experimental setting. This outcome is likely due to the relatively short sequence length and limited dataset size. For larger contexts, where full attention becomes computationally expensive, the windowed approach may offer a better trade-off between efficiency and modeling capacity.

In summary, the most effective configuration for this task was the Transformer model using `bfloat16` precision combined with the FlashAttention mechanism. This setup achieved a strong balance between training speed, memory efficiency, and model performance. While the standard `float32` model attained the lowest perplexity after a single epoch, its significantly longer training time makes it less practical for large-scale or time-constrained training scenarios.

Overall, these results highlight that modern memory- and compute-optimized techniques, such as reduced-precision training and FlashAttention, are crucial components for

efficient Transformer training, particularly when scaling to larger models and longer input sequences.

References

[1] "Wolne Lektury Speakleash". In: (). URL: https://speakleash.org/en/dashboard-en.

[2] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG]. URL: https://arxiv.org/abs/2205.14135.

[3] Research And Development Laboratory 13 (radlab). *radlab/polish-gpt2-small-v2*. Model językowy GPT-2 small dla języka polskiego udostępniony na Hugging Face, dostęp: 11.12.2025. Hugging Face. Nov. 2023. URL: https://huggingface.co/radlab/polish-gpt2-small-v2.

[4] PyTorch. *torch — PyTorch Python Package*. PyPI / PyTorch Foundation. Nov. 2025. URL: https://pypi.org/project/torch/.

[5] Minjun Kim (mjun0812). *flash-attn: Prebuilt Flash Attention Wheels*. Version: 2.6.3, CUDA 12.4, for PyTorch 2.5, Python 3.10 (accessed: 2025-12-11). GitHub. 2025. URL: https://github.com/mjun0812/flash-attention-prebuild-wheels/releases/download/v0.5.4/flash_attn-2.6.3+cu124torch2.5-cp310-cp310-linux_x86_64.whl.