FOUNDATIONS OF ML FOR SYS RESEARCHERS
Assignment 1

Aris Karatzikos
Eid: ak58437

Summary

In this project we had the assignment of implementing and training a language model Transformer. The key point of this assignment is to understand the strengths and limitations of practical techniques and common variables of these type of models. At first level, we had to analyse architectural designs of the system, such as sequence/context length and model number of layers and compare throughput, GPU memory usage and accuracy. We also explore optimization techniques and evaluate their contribution and limitations.

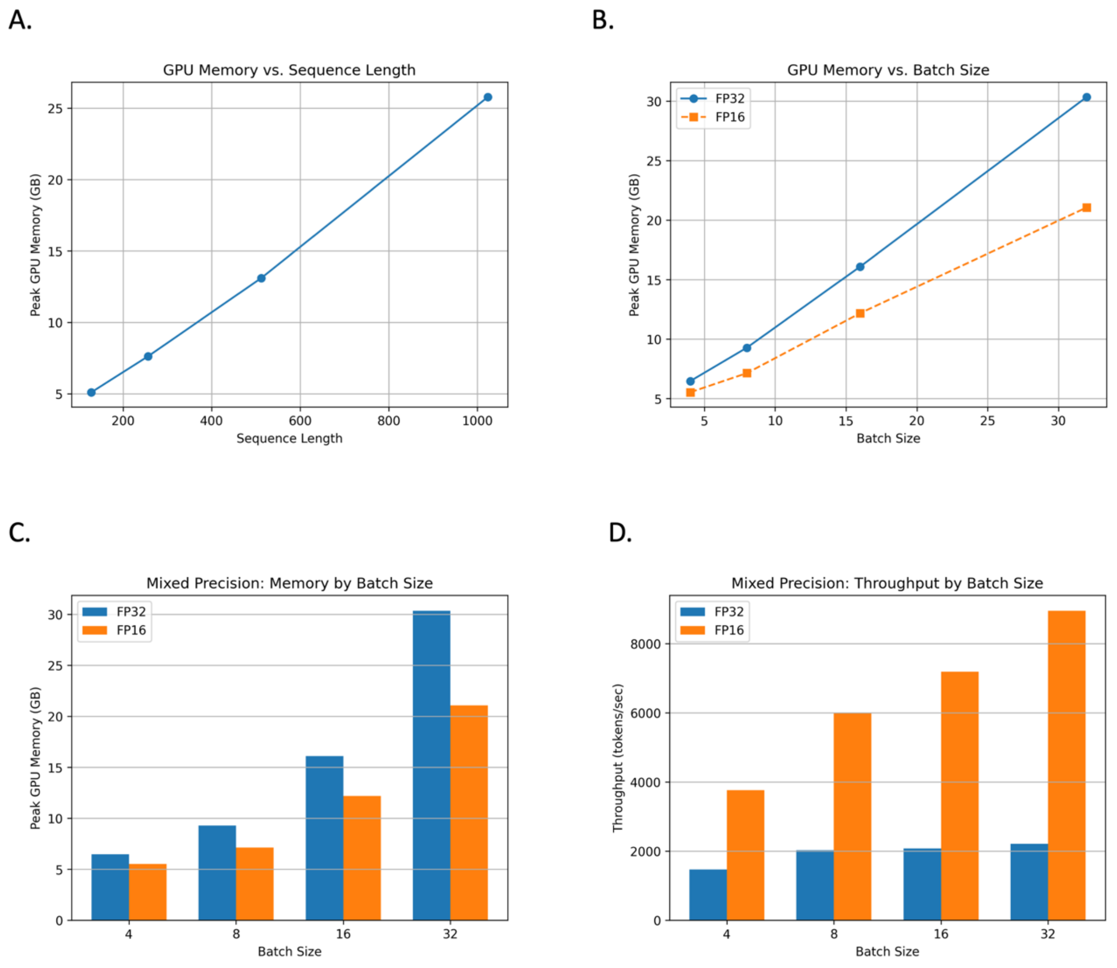
Experimental Setup

Dataset. We analyze the task as standard next-token prediction on **WikiText-103**, a long-form language modeling benchmark. The training portion contains approximately 103 million tokens, We load the dataset via Hugging Face datasets using its standard splits. For our experiments, we optimize on the train split and report perplexity and token-level accuracy on the validation split. Text is tokenized with a subword BPE tokenizer. In our runs it produces a 50k-token vocabulary.

Model architecture. The model is a compact Transformer. Tokens are embedded and augmented with sinusoidal positional encodings, then passed through a stack of Transformer encoder layers (multi-head self-attention + feed-forward). We apply a strict causal mask so each position attends only to the past. Training uses the next-token cross-entropy: logits are shifted by one step to align each position's prediction with the next ground-truth token, and the loss ignores padded positions.

Hardware. All experiments were executed on the TACC Lonestar6 cluster on a single **NVIDIA A100 (40 GB)** GPU.

2. Profiling Results

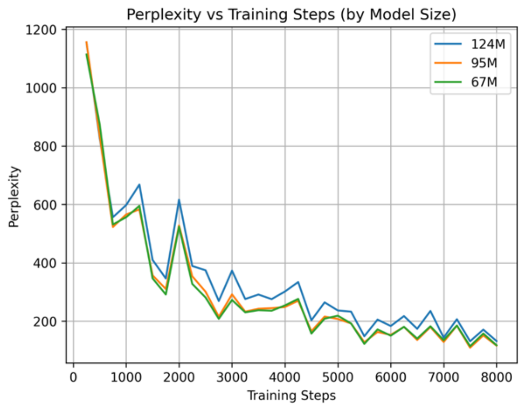


In figure 1A, we first measure how memory scales with context length. With batch size fixed at eight and mixed precision on, peak memory rises from roughly 5.0 GB at 128 tokens to 13 GB at 512 and 25 GB at 1024, reflecting the memory growth with sequence length mainly because of the attention matrices construction. In plot 1B, we experiment with batch sizes at a fixed sequence length of 512. Peak memory grows roughly linearly with batch size. The FP16 curve sits consistently below FP32. At batch size thirty-two, FP32 reaches about 30 GB of peak memory while FP16 uses about 21 GB, a reduction of roughly 30 %. In figures 1C and 1D, we include paired bar charts per batch size for throughput and memory. These plots summarize the following trends: FP16 achieves between two and four times the throughput of FP32 while lowering peak memory by roughly a quarter to a third, depending on batch size.

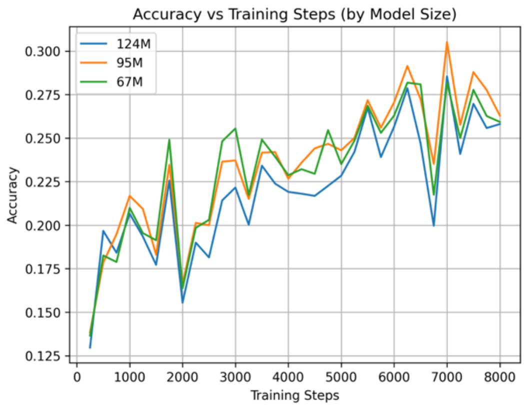
Base Model Training Evaluation

We did some analysis on the comparison between model size and context length vs task performance.

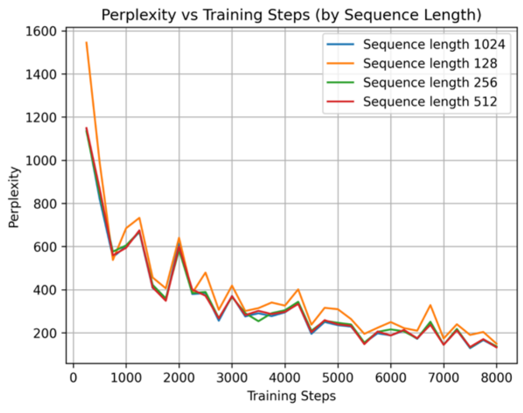
A.



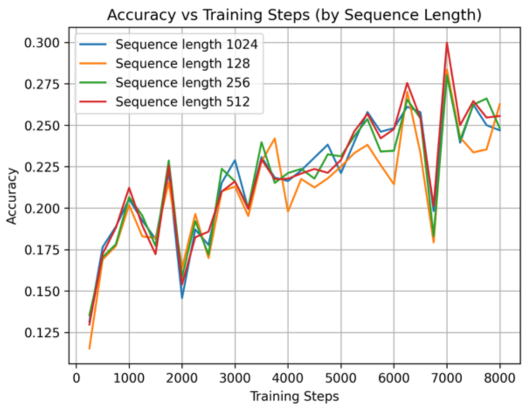
B.



C.



D.



We observe that for the different model sizes perplexity drops over time, reasonably for all model sizes and context lengths while accuracy increases having some fluctuations, and in fact accuracy and ppl for bigger models tend to be slightly worse which shows that bigger models need more time to converge. For sequence lengths we don't observe any major difference between the different configs, while the trend seems pretty similar to all.

We performed these tests on the training set since there are training steps and the full dataset is never reseen, there is no need for evaluation test results.

3. Background on the chosen optimizations

Gradient checkpointing

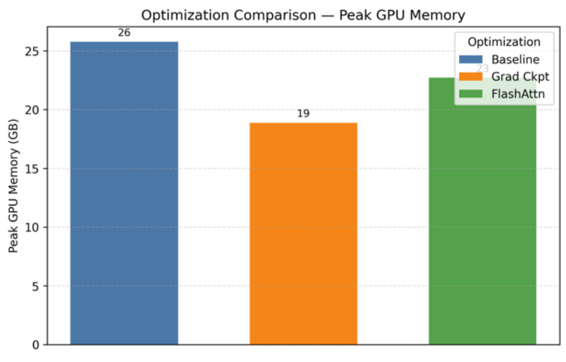
During a standard forward pass, each Transformer layer caches intermediate activations so backpropagation can compute gradients without recomputing them. The memory cost of storing these activations dominates the footprint for long sequences and deep stacks. Gradient checkpointing trades memory for compute by discarding some activations in the forward pass and recomputing them on the fly during the backward pass. It is especially valuable for fitting longer contexts or larger batches on a fixed-memory GPU, and its effect on accuracy is neutral because the computation is the same but just recalculated.

Flash Attention

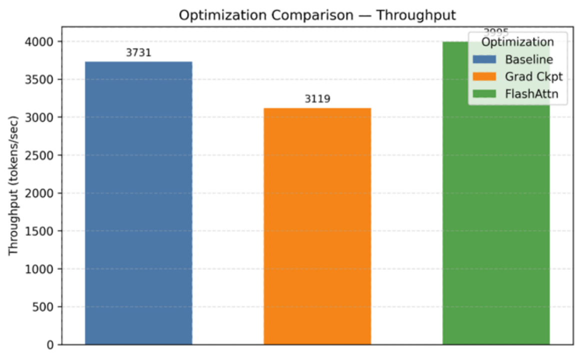
FlashAttention is a fused attention kernel that reorganizes the attention computation to minimize memory traffic. Instead of calculating the full L^2 score matrix, it tiles QKV pairs into blocks and performs the matmul–softmax–matmul pipeline in chunks fast on-chip memory, using an online softmax to maintain numerical stability. This design reduces activation memory and reads/writes to the slower GPU memory, which often translates into higher throughput and lower peak memory.

In order to make comparisons with our optimizations we have implemented a base model consisted of 12 layers, 12 heads, batch size of 12 and which has been trained on 8000 steps, specs which are the same as in the optimizations model architecture.

A.



B.



Finally, we compare before/after versions of two optimizations and the base model at batch size 16 and sequence length 1024. In Figure A we compare peak memory across the three configurations. Turning on gradient checkpointing cuts the footprint from 26 GB (Baseline) to 19 GB (−27%), as expected when activations are recomputed instead of stored. FlashAttention also reduces memory from 26 GB to 23 GB, because the fused kernel avoids materializing the attention score matrix and keeps blocks on-chip. In Figure 5B we see the speed trade-off. Checkpointing lowers throughput from 3,731 to 3,119 tok/s (−16%) due to recompute, while FlashAttention nudges throughput up to about 4,000 tok/s (+7% vs. Baseline). Overall, checkpointing is the lever for fitting longer contexts/larger batches at a moderate speed cost, and FlashAttention provides a small but real win in both memory and speed on this setup.