

FlashAttention-Enhanced GPT-2 for Paraphrase Detection and Sonnet Generation

Stanford CS224N Default Project

Norah Asemota
Department of Computer Science
Stanford University
nasemota@stanford.edu

Katie Liu
Dept. of Computer Science
Stanford University
katiel25@stanford.edu

William Yang
CGOE
Stanford University
yangwa@stanford.edu

Abstract

Large language models like GPT-2 have demonstrated strong performance across a range of natural language processing tasks when fine-tuned from a pretrained checkpoint. We implement core components of GPT-2 from scratch, including masked multi-head causal self-attention, transformer layers, and positional embeddings, and fine-tune on three downstream tasks: sentiment analysis, paraphrase detection, and sonnet generation. For paraphrase detection, we formulate the task in a cloze-style manner and show that scaling to GPT-2 medium combined with learning rate tuning achieves a test accuracy of 0.897, a 4.1% absolute improvement over the staff baseline. For sonnet generation, a sequential ablation over decoding hyperparameters combined with scaling to GPT-2 medium yields a final chrF of 41.86, a +0.47 improvement over the GPT-2 small baseline. As an extension, we implement FlashAttention using Triton kernels and Longformer-style sliding window attention, benchmarking both against standard attention across latency, memory bandwidth, and throughput. FlashAttention demonstrates consistent efficiency gains at longer sequence lengths while computing exact attention.

1 Key Information to include

- Mentor: Simon Kim
- Late Days (if you are using any):

Team member	Late days remaining
Norah Asemota	2
Katie Liu	0
William Yang	5
Total pooled	7
Team size	3
Late days applied for the report	2

2 Introduction

The Transformer architecture, specifically auto-regressive models like GPT-2, has become the foundational backbone of modern natural language processing. By pre-training on large corpora and fine-tuning on specific downstream tasks, these models achieve state-of-the-art performance. However, standard self-attention mechanisms scale quadratically $\mathcal{O}(N^2)$ with sequence length, creating severe memory and throughput bottlenecks for longer contexts.

In this project, we explore both the downstream capabilities of GPT-2 and methods to optimize its core attention bottleneck. We first implement a GPT-2 model and Adam optimizer from scratch, fine-tuning the model on sentiment analysis, paraphrase detection, and Shakespearean sonnet generation.

Subsequently, we implement and evaluate two distinct solutions to the attention scaling problem: FlashAttention, which optimizes memory IO for exact attention computation, and Longformer attention, which utilizes a sparse, sliding-window approach. This report details our implementations, fine-tuning strategies, and a comparative analysis of task performance versus computational efficiency.

3 Related Work

The original Transformer architecture introduced scaled dot-product attention, dispensing with recurrence. GPT-2 built upon this by scaling up a decoder-only architecture for generalized language modeling.

To combat the $\mathcal{O}(N^2)$ complexity of standard attention, various approximations have been proposed. Longformer introduced a drop-in replacement utilizing a local sliding window combined with task-motivated global tokens to achieve $\mathcal{O}(N)$ complexity. Alternatively, FlashAttention takes a hardware-aware approach, tiling Q, K, and V matrices into SRAM to compute exact attention without materializing the full attention matrix in high-bandwidth memory (HBM).

4 Approach

4.1 GPT-2 Implementation

We implement a small version of GPT-2 with 12 transformer layers, a hidden dimension of 768, and 12 attention heads.

Masked Multi-Head Causal Self-Attention. We implement scaled dot-product attention with a causal mask to prevent tokens from attending to future positions:

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

GPT-2 Transformer Layer. Each layer applies pre-LayerNorm, masked multi-head self-attention, and a position-wise feed-forward network with residual connections and dropout ($p = 0.1$):

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

Embedding Layer. The input embedding sums a trainable token embedding and a learned positional embedding (dimension 768), supporting context lengths up to 1024 tokens.

Adam Optimizer. We implement the AdamW optimizer [1] with decoupled weight decay using the efficient bias-corrected step size:

$$\alpha_t = \alpha \cdot \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}, \quad \theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot \frac{m_t}{\sqrt{v_t} + \epsilon} \quad (3)$$

4.2 Downstream Task Fine-Tuning

Sentiment Analysis. We fine-tune GPT-2 for sentiment classification by extracting the last token’s hidden state and passing it through a dropout layer followed by a linear projection to the label space. We experiment with two fine-tuning modes: (1) *last-linear-layer*, where only the classification head is trained, and (2) *full-model*, where all parameters are updated. We evaluate on both the Stanford Sentiment Treebank (SST) and the CFIMDB dataset.

Paraphrase Detection. We formulate paraphrase detection as a cloze-style generation task [2]. Given a question pair, we prompt the model with:

Is “[sentence 2]” a paraphrase of “[sentence 1]”?

and compare the log-probabilities of generating *yes* (BPE token id 8505) versus *no* (BPE token id 3919). We fine-tune on the Quora question pairs dataset and predict the higher-probability token as the label. We investigate two axes of improvement over the baseline: model scale (GPT-2 small vs. GPT-2 medium) and learning rate tuning.

Sonnet Generation. We fine-tune GPT-2 on 143 Shakespearean sonnets using cross-entropy loss on next-token prediction. During evaluation, the model is conditioned on the first 3 lines of each held-out sonnet and autoregressively generates the remaining 11 lines. Generations are evaluated using the chrF metric [3], a character-level n -gram F-score.

4.3 Extension: FlashAttention and Longformer

FlashAttention. As our primary extension, we implement FlashAttention [4] using custom Triton kernels as a drop-in replacement for standard scaled dot-product attention. Triton also provides an abstraction for the Triton-C language which is also an abstraction for thread management and memory coalescing of threads [5]. Conventional attention computes $O = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$ by loading the entire $N \times N$ matrix from the HBM DRAM into L1 SRAM, computing QK^T , loading it back to DRAM, then loading it back again to multiply it by matrix V . Softmax is also computed by loading entire rows at once. FlashAttention tiles Q, K, V into smaller blocks that can be loaded and written from SRAM quickly. Softmax is computed incrementally using the online softmax algorithm [6] by tracking the maximum to avoid overflow $m_{new} = \max(m_i, \max(p_{new}))$, computing the correction factor to rescale previous computations $\alpha = \exp(m_i - m_{new})$, updating the sum in the denominator $l_i = l_i \alpha + \sum \exp(p_{new} - m_{new})$. Both the attention calculations and online softmax are fused into a single kernel for forward and back propagation, reducing memory usage from $O(N^2)$ to $O(N)$. We modified the provided GPT2 code to toggle between using our custom FlashAttention implementation, PyTorch’s implementation of scaled_dot_product_attention with Flash Attention 2 Kernels [6], and Longformer. is toggled via the `-use_flash_attn_kernel` flag with no other architectural changes, enabling controlled efficiency comparison against standard attention.

Longformer. As a structurally distinct efficiency comparison, we implement Longformer-style sliding window attention [7], which replaces full self-attention with local windowed attention, reducing complexity from $O(N^2)$ to $O(N \cdot w)$ where w is the window size. Unlike FlashAttention, which computes exact attention with improved IO-efficiency, Longformer is an *approximate* method that trades representational expressiveness for scalability. Both FlashAttention and Longformer are evaluated for efficiency only via benchmarking; accuracy evaluation of these variants is left for future work.

5 Experiments

5.1 Data

We evaluate on four datasets:

- **SST** [8]: 11,855 movie review sentences with 5-class sentiment labels (train: 8,544 / dev: 1,101 / test: 2,210).
- **CFIMDB**: 2,434 polar movie reviews with binary sentiment labels (train: 1,701 / dev: 245 / test: 488).
- **Quora** : 400,000 question pairs with binary paraphrase labels; we use the provided subset (train: 141,506 / dev: 20,215 / test: 40,431).
- **Sonnets**: 154 Shakespearean sonnets (train: 143 / test: 12).

5.2 Evaluation method

For paraphrase detection, we report binary classification accuracy on both dev and test sets.

For sonnet generation, we report the chrF score [3], a character-level n -gram F-score that captures surface-form similarity between generated and reference sonnets. We adopt chrF as our primary metric because it is more sensitive than BLEU to the stylistic and lexical regularities of Shakespearean verse. Hyperparameter tuning was conducted sequentially: each parameter was swept in isolation while holding all others fixed at their current best value, and the best-performing setting was carried forward before tuning the next parameter.

For physical performance of our model, we evaluated our FlashAttention implementation standalone across the following metrics with sequence lengths from up to 2^{17} latency (ms), compute throughput (TFLOP/s), memory bandwidth (TB/s), and tokens per second. We also benchmarked the same metrics with the default gpt2 model, batch size 8, learning rate $1e-5$ for both paraphrase and sonnet dev. As well as 10 epochs softmax temperature of 1.2 and 1 epoch for paraphrase respectively. These experiments used a fix block size of 64. We also tried standalone FlashAttention experiments via latency with different block sizes of 16, 32, and 128. All our performance benchmarks used floating point 32 for forward pass, and floating point 16 for backpropagation since our experiments on L4 ran into memory issues on L4 GPU’s.

5.3 Experimental details

For paraphrase detection, we ran the following configurations, all trained for 3 epochs with batch size 128 on an NVIDIA H100 GPU:

- GPT-2 small, normal attention, $lr = 1 \times 10^{-5}$
- GPT-2 medium, normal attention, $lr = 1 \times 10^{-5}$
- GPT-2 medium, normal attention, $lr = 3 \times 10^{-5}$
- GPT-2 medium, normal attention, $lr = 5 \times 10^{-5}$
- GPT-2 medium, FlashAttention, $lr = 5 \times 10^{-5}$

The efficiency benchmark compares FlashAttention, standard attention, and Longformer across padded sequence lengths of 40–180 tokens on an NVIDIA L4 GPU, using the dev set for inference only.

For sonnet generation, we conduct a sequential ablation study starting from a baseline configuration of GPT-2 small with temperature = 1.2, top-p = 0.9, no top-k filtering, max generation length = 128, batch size = 8, and learning rate 1×10^{-5} for 10 epochs. We first sweep top-k $\in \{40, 50, 60\}$, fixing the best value before proceeding to the next parameter. We then sequentially tune max generation length, temperature $\in \{0.8, 1.2, 1.5\}$, number of training epochs, top-p $\in \{0.7, 0.9, 1.2\}$, batch size $\in \{6, 8\}$, and learning rate $\in \{1 \times 10^{-5}, 5 \times 10^{-6}\}$. Once the optimal decoding configuration is established on GPT-2 small, we scale the model to GPT-2 medium to evaluate whether larger pretrained representations yield further chrF gains. All sonnet experiments are evaluated on the held-out dev set using the chrF metric.

5.4 Results

Table 1: Paraphrase detection results on the Quora dataset. Best test result **bolded**. FlashAttention and Longformer are evaluated for efficiency only; see Figure 3.

Model	Attention	LR	Dev	Test
Staff Baseline	Normal	—	0.882	0.856
GPT-2 Small	Normal	1×10^{-5}	0.871	—
GPT-2 Medium	Normal	1×10^{-5}	0.883	—
GPT-2 Medium	Normal	3×10^{-5}	0.898	0.892
GPT-2 Medium	Normal	5×10^{-5}	0.901	0.897

Moving from GPT-2 small to GPT-2 medium at the same learning rate improves dev accuracy by 1.2% ($0.871 \rightarrow 0.883$), consistent with the larger model’s stronger pretrained representations. Tuning the learning rate from 1×10^{-5} to 3×10^{-5} adds another 1.5% ($0.883 \rightarrow 0.898$), and pushing to 5×10^{-5} gives a further 0.3% gain ($0.898 \rightarrow 0.901$), suggesting diminishing returns. Our best test accuracy of **0.897** at 5×10^{-5} represents a **4.1%** absolute improvement over the staff baseline of 0.856. These results were better than expected as we did not anticipate that learning rate alone would drive such consistent gains on top of model scaling. Our attempts to accelerate training and model inference of our paper with FlashAttention. Figure 3 initially demonstrated promising results. Based on the mathematical equivalence of FlashAttention’s blocked softmax to standard attention [4], we expect accuracy to be preserved, consistent with results reported in the original paper. Longformer is

Table 2: Sequential ablation results for sonnet generation evaluated on the held-out dev set using chrF. Each row modifies a single parameter from the current best configuration; the accepted value is **bolded**. Parameters are tuned in the order listed.

Parameter	Value	chrF	Accepted
Baseline (GPT-2 small)	default	41.39	—
top-k	40	41.68	✓
	50	41.71	
	60	40.73	
max_length	128	41.71	✓
	300	41.38	
temperature	0.8	40.38	✓
	1.2	41.71	
	1.5	41.73	
epochs	10	41.71	✓
	20	41.09	
top-p	0.7	40.49	✓
	0.9	41.71	
	1.2	40.42	
batch_size	6	41.74	✓
	8	41.40	
learning rate	1×10^{-5}	41.71	✓
	5×10^{-6}	40.82	
model scale	GPT-2 small	41.62	✓
	GPT-2 medium	41.86	

included as an efficiency-only comparison; accuracy evaluation requires task-specific global attention token design and is left for future work.

We first experimented with different block sizes inside our FlashAttention kernel. While block sizes had little effect on forward pass, we observed a block size of 64 being the most optimal in terms of latency. As mentioned in the original FlashAttention paper, increasing block size reduces HBM operations but larger block sizes may become bottlenecked by other operations. [4]

For performance benchmarking we achieved the following metrics with a fixed block size of 64 for the following sequence lengths:

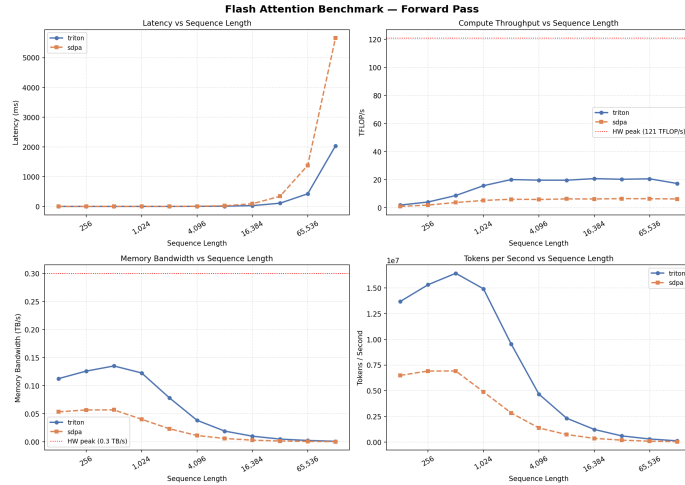


Figure 1: Standalone FlashAttention kernel benchmarks for forward pass

For our sanity setup, we initially believed that our FlashAttention kernel forward pass implementation outperformed Pytorch’s built in FlashAttention2 on all four metrics in Figures 2 and 3. 1. We observed that our kernels have better performance the larger the sequence length.

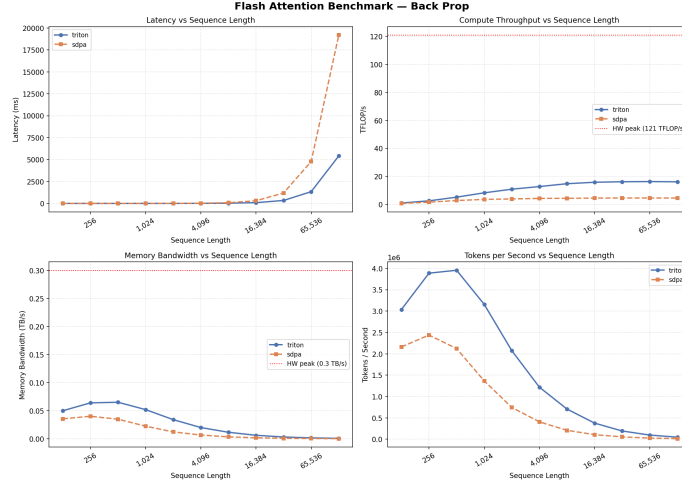


Figure 2: Standalone FlashAttention kernel benchmarks for backprop

Our backpropagation kernel standalone benchmark for FlashAttention performed similarly well. We noticed that as the sequence length sequence our FlashAttention kernel performs better on latency and throughput. 2

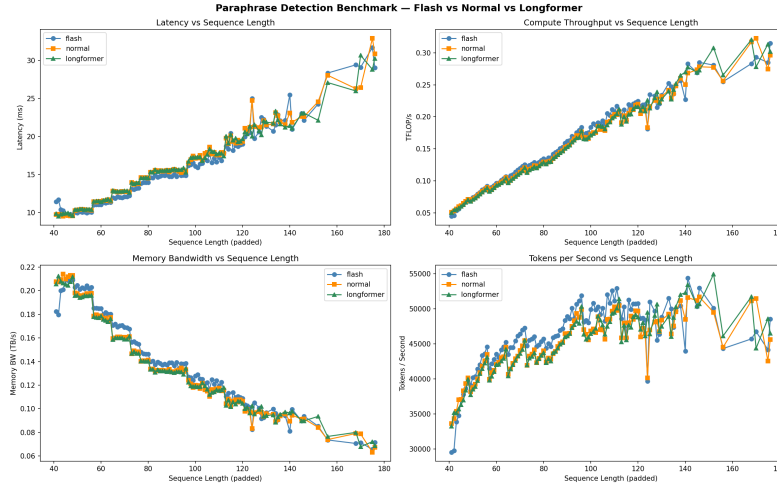


Figure 3: Efficiency benchmarks comparing FlashAttention, standard attention (Flash Attention 2), and Longformer across sequence lengths (L4 GPU)

Based on performance benchmarking for Figure 3 shows our efficiency benchmarks. FlashAttention consistently achieves slightly better higher memory bandwidth utilization and tokens-per-second throughput at midlength sequences, comparable to Longformer and Pytorch’s scaled dot product attention (SDPA) implementation

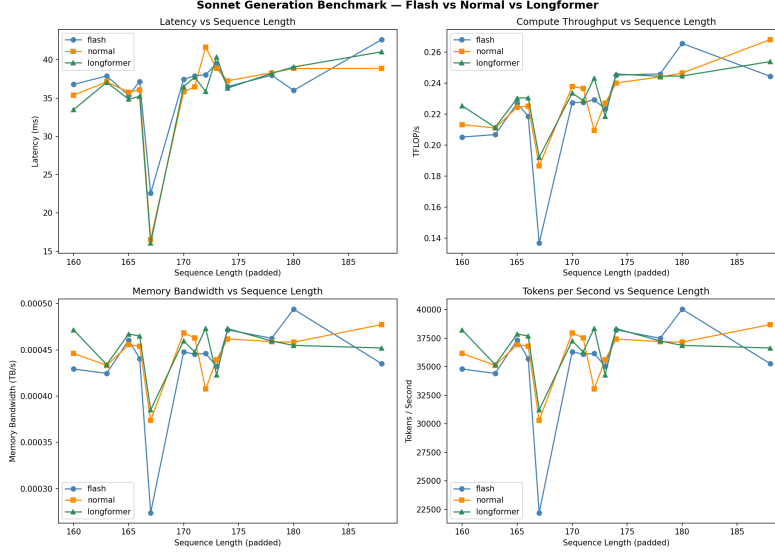


Figure 4: Physical performance benchmarks comparing Flash Attention, PyTorch’s scaled dot product attention (SDPA) which uses Flash Attention 2, and Longformer across sequence lengths

For sonnet generation physical performance benchmarking, our FlashAttention kernel and longformer implementation slightly underperformed the default SDPA implementation. We notice that this performance differs and that there is greater instability across different sequence lengths compared to paraphrase benchmarking. 4

6 Analysis

Our paraphrase detection improvements come from two places. Scaling up to GPT-2 medium helped as the larger model has more capacity to pick up on the semantic differences between question pairs, which matters a lot for the cloze-style setup where the model needs to “know” what yes and no mean in context.

Learning rate mattered more than we expected. The default 1×10^{-5} was too slow given how much training data we had. Going to 3×10^{-5} gave us a solid +1.5% bump, and 5×10^{-5} squeezed out another +0.3%, though the gains were clearly tapering off. Dev and test stayed close (0.901 vs. 0.897), indicating the model generalizes well without overfitting.

For sonnet generation, the sequential ablation in Table 2 shows that no single hyperparameter change produces dramatic gains, but accepted changes compound to a total improvement of +0.47 chrF over the baseline of 41.39. The largest single gain came from introducing top-k filtering at $k = 50$ (+0.32), while $k = 60$ sharply degraded performance, suggesting the model is sensitive to over-permissive candidate sets. Raising temperature to 1.5 encouraged broader vocabulary exploration appropriate for Shakespearean verse, whereas 0.8 collapsed the distribution onto repetitive high-frequency tokens. Training beyond 10 epochs hurt performance (−0.62), consistent with the small 143-sonnet corpus making the model prone to overfitting. The most reliable lever was model scale: GPT-2 medium added +0.24 chrF over the tuned small configuration, suggesting that further gains are better pursued through scaling than decoding hyperparameter tuning alone.

For performance metrics our standalone FlashAttention benchmarks showed similar performance to Pytorch’s SDPA. We were fully expecting similar results to occur when we benchmarked performance metrics on our paraphrase and sonnet dev tasks. To our surprise, despite both environments being in CUDA in all cases, we noticed that our FlashAttention kernel only matched Longformer and built in SDPA and slightly underperformed for sonnet generation. We speculate this is due to PyTorch’s SDPA backend calling highly optimized C++ kernels such as CuDNN. [9]

Since paraphrase and sonnet generation process fewer tokens (40-180) in our tasks was significantly smaller than the tens of thousands of tokens we did in our standalone benchmark. In our particular

cases, the overhead of launching Triton kernels could outweigh the I/O benefits of FlashAttention itself.

We noticed for paraphrase 3 and especially sonnet generation plots 4 very unstable dips in performance for certain sequence lengths and especially sonnet generation it's possible that the limited memory of L4 is reducing the number of available fast-to-access registers, causing latency and throughput spikes. Another possibility is that sequence lengths that aren't powers of 2 could cause uneven distribution of resources and branch divergence.

Another bottleneck was that we ran it on an older, Nvidia L4 GPU, which had less memory than standard H100's since those of us with H100's ran out of compute credits due to running many ablations for accuracy performance. L4 GPU had only 100kb per streaming multiprocessor, we had to scale backpropagation values from fp32 down to fp16 and be very conscious of loading tensors from HBM.

We also ran out of time and compute to efficiently explore other hyperparameters such as learning rate. For accuracy experiments which were only run on the H100, we only had time to try a learning rate of 5×10^{-5}

7 Conclusion

We built GPT-2 from scratch and fine-tuned it for sentiment analysis, paraphrase detection, and sonnet generation, with a FlashAttention kernel from scratch in Triton and Longformer as efficiency extensions. For paraphrase detection, combining model scale (GPT-2 medium) with learning rate tuning (5×10^{-5}) yields a test accuracy of 0.897, a 4.1% absolute improvement over the staff baseline of 0.856. Our FlashAttention implementation, built using Triton CUDA kernels, demonstrates improved memory bandwidth and throughput at longer sequence lengths compared to standard attention, with Longformer included as a structural efficiency comparison representing approximate sparse attention. Together, these results show that principled model scaling and hyperparameter tuning drive meaningful accuracy gains, and that IO-aware exact attention is a practical efficiency improvement for transformer fine-tuning. Future work could explore ensembling multiple checkpoints, accuracy evaluation of FlashAttention and Longformer, and applying FlashAttention's memory savings to enable longer context windows for sonnet generation.

Team contributions

Norah Asemota implemented the GPT-2 sentiment classifier, including the last-linear-layer and full-model fine-tuning pipelines. Led all paraphrase detection experiments, including cloze-style task formulation, model scaling from GPT-2 small to GPT-2 medium, and learning rate tuning, achieving the final test accuracy of 0.897. Also implemented Longformer-style sliding window attention and co-implemented the backward pass of FlashAttention with Katie.

Katie Liu implemented the GPT-2 and GPT-2 layer of the base GPT-2 model and debugged the attention layer to properly apply causal masking. She led all sonnet generation experiments, conducting a full sequential ablation study over decoding hyperparameters evaluated via chrF on the held-out dev set. She identified the optimal configuration params achieving a chrF of 41.74 on GPT-2 small, and subsequently scaled to GPT-2 medium to reach a final chrF of 41.86. She also implemented the backward pass of FlashAttention and Longformer-style sliding window attention with Norah, and debugged memory issues encountered during backpropagation on the L4 GPU.

William Yang Implemented the final project baseline part of the project and Adam optimizer and original attention layer. Spearheaded FlashAttention efforts and wrote the entire forward pass kernel in Triton for Flash Attention and provided it as template for Norah/Katie to work on backprop kernel. Ran standalone ablations for different block sizes on our Triton implementation. Generated Flash attention plots: wrote kernel sanity benchmark code to test FlashAttention. Single handedly integrated benchmarking comparison for said metrics across existing paraphrase detection and sonnet generation files between PyTorch SDPA, our FlashAttention kernel, and Longformer for everyone to use. Worked with and got help from Norah/Katie to try and run some of our FlashAttention kernel benchmarks on her H100 setup instead of L4 GPU. Debugged Katie's sonnet fine tuning setup and patched memory bugs in backprop kernels.

References

- [1] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
- [3] Maja Popović. chrF: Character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395. Association for Computational Linguistics, 2015.
- [4] Stefano Ermon Atri Rudra Tri Dao, Daniel Y. Fu and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
- [5] Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL '19)*, page 10, New York, NY, USA, 2019. ACM.
- [6] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.
- [7] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [8] Jean Wu Jason Chuang Christopher D Manning Andrew Y Ng Richard Socher, Alex Perelygin and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank.
- [9] PyTorch Contributors. *torch.nn.functional.scaled_dot_product_attention*, 2024. Accessed: 2025-03-15.