

Evaluating Performance, Efficiency, and Memory Trade-offs in GPT-2 Attention Mechanisms

Stanford CS224N {Default} Project

Lily Bailey

Department of Computer Science
Stanford University
lbailey1@stanford.edu

Devon Smith

Department of Computer Science
Stanford University
devon2@stanford.edu

Abstract

This paper describes the implementation of the GPT-2 architecture for the default project, with development details for sentiment classification, paraphrase detection, and sonnet generation tasks. The standard GPT-2 masked multi-headed self-attention mechanism scales quadratically ($O(T^2)$) in time and memory with sequence length, which creates a bottleneck effect on long-context inputs. To understand the trade-offs of attention mechanisms, we conducted controlled experiments to compare standard self-attention to flash attention, sliding window attention, and mixed local/global attention, evaluating the model's accuracy on each of the three downstream tasks. We also assessed efficiency and memory usage in a benchmarking analysis across sequence lengths on an NVIDIA A10G GPU. We found that flash attention is a strict improvement over standard attention, reducing latency by up to 40% and memory by up to 34% at sequence length 1024, without causing any loss in accuracy. Sliding window attention showed no wall-clock efficiency gains when implemented with sparse masking and degrades accuracy on long-context inputs (CFIMDB: 0.873 vs. 0.980). However, we also implemented an efficient loop-based version of sliding window which achieves $O(T \cdot w)$ memory at the attention layer ($4\times$ reduction in isolation, 116 MB vs. 467 MB, $w=64$). Mixed local/global attention (layers alternate between sliding window and masked multi-headed self-attention) recovers most of this accuracy loss (CFIMDB: 0.947) and substantially restores sonnet quality. This result implies that periodic global attention significantly helps to preserve long-range modeling capacity. We ran a global token ablation which further shows that just 2 global tokens (which attend to all other tokens while the rest attend locally) within sliding window attention recovers CFIMDB accuracy to 0.980, matching full attention results on this dataset. We also implemented Group Query Attention as part of our efficiency and memory benchmarking, and found that it reduces memory by ~ 37 MB across all sequence lengths, with its primary benefit being KV-cache reduction during inference. Ultimately, our findings clarify the trade-offs between attention efficiency and task performance.

1 Key Information to include

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

Team member	Late days remaining
Lily Bailey	6
Devon Smith	6
Total pooled	12
Team size	2
Late days applied for the report	3

2 Introduction

The standard GPT-2 implementation described in the default project uses masked multi-headed self-attention, which attends each token to all preceding tokens via scaled dot-product attention. Despite its language processing benefits, this mechanism scales quadratically in time and memory with sequence length, creating a bottleneck on long-context inputs such as long documents or generation tasks. Recent research papers have suggested certain modifications to attention mechanisms to address this complexity. Flash attention [1] achieves exact attention via IO-aware tiling by eliminating intermediate matrix materialization (break matrices into smaller blocks) without approximating. Sliding window attention [2] proposed in the Longformer paper restricts each token to a local context window, which reduces theoretical complexity from $O(T^2)$ to $O(T \cdot w)$, where w is the window size which is much smaller than the number of tokens T . Hybrid strategies such as Longformer’s mixed local/global attention layers combine both mechanisms to balance efficiency and long-range modeling capacity [2].

The aforementioned prior works largely evaluate each mechanism in isolation on a single task, and as such optimize for those outcomes. We thus wanted to investigate how their theoretical efficiency improvements translate to actual wall-clock gains, and their impact on model performance across tasks with different context-length requirements. One of the most relevant challenges we came across is that theoretical complexity improvements do not always translate to practical gains. Our naive PyTorch sparse masking implementation of sliding window attention still computes the full $T \times T$ dot-product matrix before applying the mask, so no memory or latency is saved. Achieving true $O(T \cdot w)$ complexity requires either custom CUDA kernels or an implementation that avoids materializing non-contiguous views.

We implemented GPT-2 with modular support for four main attention variants and evaluated each under identical training conditions and hyperparameters across sentiment classification, paraphrase detection, and sonnet generation. Our results show that flash attention is the only variant that delivers wall-clock efficiency gains without the collateral of implementation complexity or accuracy loss. Sliding window attention in our masked implementation (which still materializes the full $T \times T$ matrix) provides no efficiency improvement. We explored a more efficient loop-based implementation, which ended up achieving $O(T \cdot w)$ memory scaling (a $4\times$ reduction in attention-layer memory at sequence length 1024), but does still incur significant Python loop overhead at short sequences. Sliding window attention also degrades task accuracy on longer-context inputs, with CFIMDB sentiment accuracy dropping from 0.980 to 0.873 for window size 64. We found in our qualitative analysis that sonnet generation deviates from normal Shakespearean structure when sliding window attention is used, producing run-on lines and weaker thematic coherence. Mixed local/global attention, which alternates sliding window and full-attention layers, recovers most of this performance loss, indicating that periodic global context is important for long-form generation to remain coherent.

3 Related Work

Sparse Attention / Sliding Window Mechanism Beltagy et al.’s Longformer paper [2] introduces sliding window attention. With sliding window attention, most tokens attend locally solely to those tokens within a specific context window, while a few tokens may be assigned global attention (attend to all other tokens). This reduces the quadratic complexity of masked multi-headed self-attention to approximately linear in sequence length. The Longformer paper also introduces an architecture in which some layers use sparse attention (sliding window) while others use complete global attention.

Flash Attention Mechanism Dao et al.’s FlashAttention paper [1] introduced the flash attention mechanism, which preserves the exact attention pattern of masked multi-headed self-attention (to

maintain model accuracy), but utilizes a memory-efficient GPU implementation that fuses attention operations in order to avoid materializing large intermediate matrices. This leads to efficiency gains in training.

Group Query Attention Mechanism Ainslie et al. [3] introduced Group Query Attention, which describes reducing the number of key and value heads by sharing them across groups of query heads. This lowers memory and computational costs, but was shown to achieve the same performance metrics as standard multi-head attention. Since GQA is designed for efficiency rather than accuracy, we evaluated it solely in the efficiency benchmark rather than across downstream tasks.

In our project, we sought to understand how implementations of these attention mechanisms result in trade-offs between efficiency, storage, and model performance on downstream tasks.

4 Approach

We implemented the GPT-2 architecture that is outlined in the default project, which consists of a token embedding layer, positional embedding layer, 12 Transformer decoder blocks, and a final language model head with attention weights tied to the token embedding matrix, optimized with AdamW with decoupled weight decay. Sentiment classification involves a linear classification head appended to the final hidden state of the last non-padding token. We evaluated both last-linear-layer (frozen GPT-2, head only) and full-model (all weights updated) fine-tuning using cross-entropy loss. For paraphrase detection, the inputs are formatted as [CLS] sentence1 [SEP] sentence2 and classified with a binary head on the last token’s hidden state. For sonnet generation, we take the full language model and fine-tune it on a corpus of 154 Shakespearean sonnets with next-token prediction loss. To generate the sonnets after fine-tuning, we use top- p sampling to select from the next most likely tokens ($p = 0.9$, $\tau = 1.2$). We evaluate the sonnet generation task with a ChrF score which calculates n-gram overlap between the generated text and reference text.

4.1 Attention Mechanism Variants

All four variants of attention mechanisms share the same projection layers (Q, K, V, output), but differ in how the attention weights are actually computed.

4.1.1 Masked Multi-Headed Self-Attention (Standard)

Masked multi-headed self-attention uses scaled dot-product attention with a causal mask so that tokens do not attend to those that come in the future:

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

where $M_{\text{causal}} \in 0, -\infty^{T \times T}$ masks future positions and d_k is the head dimension.

4.1.2 Flash Attention

We used PyTorch’s `F.scaled_dot_product_attention` with `is_causal=True`, which uses IO-aware tiling to avoid the computational expense of computing the full $T \times T$ attention matrix. This reduces memory from $O(T^2)$ to $O(T)$, without compromising task performance results at all.

4.1.3 Sliding Window Attention

Each token attends only to the previous w tokens within a local context window. This limits the effective attention span from $O(T)$ to $O(w)$ to aim for a complexity of $O(T \cdot w)$ rather than $O(T^2)$.

We implemented two versions. The masked version adds a window mask M_{window} on top of the causal mask:

$$M = M_{\text{causal}} \cup M_{\text{window}}, \quad M_{\text{window}}[i, j] = -\infty \text{ if } i - j > w$$

This implementation establishes the local window constraint, but still materializes the full $T \times T$ score matrix, so memory remains $O(T^2)$ and we do not see any gains.

To achieve $O(T \cdot w)$ memory, we implemented a more efficient version using looping that computes attention scores only within each token’s local window using contiguous slices of padded key/value tensors to avoid the cost of computing the full matrix entirely. Both of these versions produce identical outputs (since effects on weights calculated are the same) and share the same Q/K/V projection weights, but they differ in memory required and latency. The efficient loop-based version incurs Python loop overhead at short sequences, but the effects of this diminish at longer sequence lengths.

4.1.4 Mixed Local / Global Attention Layers

Inspired by Longformer [2], we alternate which attention mechanism is used between layers in our mixed implementation: layers with even indices use the standard masked multi-headed self-attention, while the layers with odd indices use sliding window attention in our implementation. This alternating layer structure enables the model to capture both long-range dependencies through the global layers and local context with the sliding window layers, while halving the number of computationally expensive masked multi-headed self-attention layers.

5 Experiments

5.1 Data

For sentiment classification, we used the Stanford Sentiment Treebank (SST) dataset (11,855 single sentences from movie reviews) and the CFIMDB dataset (2,434 labeled positive/negative movie reviews). For paraphrase detection, we used the Quora Question Pairs dataset (400,000 labeled question pairs for binary paraphrase classification). For sonnet generation, we used the Sonnet dataset, which consists of 154 Shakespeare sonnets.

5.2 Evaluation Method

For sentiment classification and paraphrase detection, we looked at the model’s accuracy on the dev sets for masked multi-headed and sliding window attention, and reported the test set accuracy (from Gradescope) for the standard implementation which we found had the best performance on the dev sets. For sonnet generation, we used ChrF scores (which measure character n-gram F-scores between generated and reference continuations) for all variants on the dev set, and the standard on test set since this scored highest. We also explored how sonnets qualitatively changed across attention variants on bases of coherence, adherence to structure, prose, and diction. To assess the efficiency impact of each attention mechanism, we ran a benchmarking experiment to gather the wall-clock time per batch (ms) and peak GPU memory (GB) measured across sequence lengths 128, 256, 512, 1024 on an NVIDIA A10G GPU.

5.3 Experimental Details

All of our experiments use GPT-2 (gpt2, 117M parameters) loaded from pretrained HuggingFace weights, trained on a single NVIDIA A10G GPU via Modal cloud compute. All of the tasks use the AdamW optimizer with weight decay 0.01 and batch size 8. The classifier (last-layer) uses learning rate 1×10^{-3} for 10 epochs, while the classifier (full-model), paraphrase, and sonnet tasks use learning rate 1×10^{-5} for 5, 5, and 10 epochs respectively (see Table 8 in Appendix). For the attention variant experiments, we held all hyperparameters fixed for consistency across comparisons. Sliding window and mixed attention use window size $w = 64$ for task performance evaluation, and group query attention uses 4 key-value heads.

6 Results

6.1 Sentiment Classification

Table 1 shows the dev set accuracy for sentiment classification on the SST and CFIMDB datasets (see Figure 1 in Appendix). Based on these outcomes, we found that full model fine-tuning performs

better than last-linear-layer fine-tuning on both datasets, with gains of 4.4 points on SST (0.443 vs. 0.487) and 13.5 points on CFIMDB (0.845 vs. 0.980). This showed us that task-specific adaptation of the GPT-2 model enables it to better capture the sentiment-based features that are maybe not already encoded in the pretrained representations.

Because flash attention is mathematically equivalent to standard attention, we excluded it from Table 1, as the performance results are the same.

Sliding Window Attention ($w=64$) achieves comparable SST accuracy (0.515) to the standard attention mechanism, but suffers a notable loss in accuracy on CFIMDB (0.873 vs. 0.980), marking a drop of 10.7 points. This degradation is attributable to CFIMDB’s longer review texts, in which sentiment signals likely span distances exceeding the window size (as an example, a negative conclusion may come at the very end of a review following language that may otherwise sound positive, which would require the model to integrate context across the full review). We conducted a larger window size ablation on CFIMDB, which further confirms that performance degrades monotonically as the local context window size gets smaller (see Appendix A.2.1). Mixed local/global attention partially recovers this loss (achieving 0.947 dev accuracy on CFIMDB), demonstrating that alternating between full-attention layers and sliding window layers restores the model’s ability to capture long-range dependencies, while retaining the efficiency of local attention in alternating layers.

Table 1: Sentiment Classification Dev Set Accuracy

Model	SST Dev	CFIMDB Dev
Last-linear-layer (standard)	0.443	0.845
Full-model (standard)	0.487	0.980
Full-model (sliding window, $w=64$)	0.515	0.873
Full-model (mixed, $w=64$)	0.512	0.947

We conducted a global token ablation to understand how within sliding window attention, having certain tokens that are assigned global attention affects dev accuracy on the CFIMDB dataset. We found that even just two global tokens on this particular task restores accuracy to the standard self-attention result.

Table 2: Global token ablation on sliding window attention ($w=128$). Table shows best dev accuracy across 5 epochs of full-model fine-tuning.

Global Tokens (g)	CFIMDB Dev Acc
0	0.939
1	0.963
2	0.980
16	0.980
Standard (full attention)	0.980

6.2 Paraphrase Detection

Table 3 reports dev set accuracy on the Quora dataset for the paraphrase detection task. We found that the attention variants achieved accuracies within a very small range (0.894-0.896), with no material performance loss resulting from sliding window or mixed attention (flash attention is excluded because of equivalence). Despite contrasting with the CFIMDB results described above, this outcome is explained by the fact that paraphrase pairs in the Quora dataset are short questions, so the full context fits within the window size of 64 tokens that we used. When sequence lengths are short relative to the window, sliding window attention is effectively equivalent to full attention, so we would not expect to see any accuracy loss.

6.3 Sonnet Generation

To evaluate the sonnet generation downstream task, we used ChrF scores, which measure character n-gram overlap between generated and reference texts. Our GPT-2 model fine-tuned on a corpus

Table 3: Paraphrase Detection Dev Set Accuracy

Model	Dev Accuracy	Test Accuracy
Standard	0.896	0.874
Sliding window ($w = 64$)	0.894	-
Mixed ($w = 64$)	0.894	-

of 154 Shakespeare sonnets achieves its highest dev ChrF score of 41.597 when standard attention (masked multi-headed self-attention) is used.

Table 4: Sonnet Generation Dev ChrF Scores Across Attention Variants

Model	Dev ChrF	Test ChrF
Standard	41.597	40.957
Flash	41.337	-
Sliding window ($w = 64$)	41.224	-
Mixed ($w = 64$)	41.127	-

Flash attention is not shown in the classification tables (Tables 1, 3) because at inference time with dropout disabled, it results in numerically identical outputs to standard attention. However, we see a small difference here (0.26 points lower than standard), which is because of the stochastic variation in dropout sampling that happens during the fine-tuning process. Ultimately, this is a minimal quality cost for the reduction in wall-clock time that the flash attention mechanism achieves (see Section 6.4). Sliding window and mixed attention score progressively lower, which is consistent with how limiting attention to a local window results in loss of long-range context.

Despite losing sonnet structure and coherence in sliding window outputs (see Section 7), all variants score within 0.5 ChrF points of standard attention. The quantitative effects are likely not as acute as we might expect because the three provided context lines are identical across all variants, which heightens character n-gram overlap and inflates scores. ChrF is therefore not a wholly sufficient metric to identify and understand structural collapse in generated continuations, which served as motivation for the qualitative analysis we completed in Section 7.

6.4 Efficiency Benchmarking

Table 5 shows the wall-clock time (ms) and peak GPU memory (GB) per forward pass run on an NVIDIA A10G GPU, averaged over 30 runs after 5 warmup runs (see Figure 2 in Appendix).

Based on these results, we can conclude that flash attention is the only variant that delivers meaningful efficiency gains over standard attention in the full forward pass. At sequence length 512, flash attention reduces latency by 26% (40.7 ms vs. 55.3 ms) and memory by 11% (0.595 GB vs. 0.665 GB). At a sequence length of 1024, the gains from flash attention are even greater, as flash attention is 40% faster than standard (84.7 ms vs. 140.3 ms) and uses 34% less memory (0.677 GB vs. 1.018 GB). The magnitude of these efficiency and memory benefits increases with sequence length, which is consistent with flash attention’s IO-aware tiling that avoids the expensive computation of the full $T \times T$ matrix.

The mask-based sliding window implementation shows no improvement over standard attention in the full forward pass, as it still computes the full $T \times T$ dot-product matrix before applying the window mask. Our efficient sliding window implementation also shows no memory improvement in the full forward pass, because peak GPU memory is dominated by model weights and activations across all 12 layers (we see a lot of memory saved within a single attention layer (Table 6), but these savings are overrun at this scale). However, the efficient implementation incurs significant latency overhead at short sequences due to its Python loop over window offsets (77.4 ms vs. 11.4 ms at seq=128), but these effects amortize at longer sequences. To confirm that our efficient sliding window implementation achieves $O(T \cdot w)$ memory in isolation, we ran an attention-only microbenchmarking exercise to isolate the score-matrix memory footprint of a single `CausalSelfAttention` call (Table 6; see Figure 3 in Appendix). Our loop-based implementation takes contiguous slices of the padded key/value tensors rather than materializing a full window tensor, which avoids the hidden

.contiguous() call that PyTorch’s batched matmul calls on non-contiguous views. We saw a $4.0\times$ reduction in memory at a single attention layer for sequence length 1024 (116.4 MB vs. 467.7 MB for $w=64$), with savings growing linearly with the sequence length. In order to actually realize full end-to-end memory reduction, it would likely require either longer sequences, larger batches, or an implementation that also avoids storing intermediate activations across layers.

When we tested group query attention with 4 key-value heads, we found that memory was reduced by approximately 37-38 MB across all sequence lengths relative to standard attention. This outcome is aligned with how GQA reduces key-value projection parameters. However, the primary efficiency benefit of GQA is KV-cache reduction, which takes place during autoregressive inference, which is not captured by the forward-pass benchmarking exercise we conducted.

Table 5: Efficiency benchmark on NVIDIA A10G GPU. Time is ms per forward pass; memory is peak GPU memory in GB (units are ms/GB). [†]Mixed and Eff. sliding are slower at seq=128 due to Python loop overhead over window offsets, which is lesser at longer sequences.

Variant	Seq=128	Seq=256	Seq=512	Seq=1024
Standard	11.4 / 0.534	23.5 / 0.562	55.3 / 0.665	140.3 / 1.018
Flash	10.3 / 0.535	20.3 / 0.554	40.7 / 0.595	84.7 / 0.677
Sliding window ($w=64$)	11.5 / 0.535	24.6 / 0.564	55.4 / 0.665	140.7 / 1.018
Mixed ($w=64$)	[†] 42.6 / 0.537	23.7 / 0.564	55.1 / 0.665	141.4 / 1.018
Eff. sliding ($w=64$)	[†] 77.4 / 0.537	23.7 / 0.564	55.1 / 0.665	141.0 / 1.018
GQA (4 KV heads)	11.9 / 0.497	22.0 / 0.526	51.3 / 0.627	133.8 / 0.980
GQA + Flash	9.8 / 0.497	22.1 / 0.526	51.3 / 0.627	133.2 / 0.980

Table 6: Attention-only memory microbenchmark. Delta (MB) is peak memory allocated during a single CausalSelfAttention call, excluding model weights.

Variant	Seq=128	Seq=256	Seq=512	Seq=1024
Standard	14.2 MB	41.0 MB	132.4 MB	466.6 MB
Sliding window ($w=64$)	14.2 MB	41.0 MB	132.6 MB	467.7 MB
Eff. sliding ($w=64$)	15.9 MB	30.3 MB	59.0 MB	116.4 MB
Eff. sliding ($w=128$)	20.9 MB	38.6 MB	74.5 MB	145.3 MB

At seq=1024, efficient sliding window ($w=64$) uses $4.0\times$ less attention memory than standard (116 MB vs. 467 MB). We also see that savings grow linearly with sequence length.

7 Analysis

To understand how attention mechanisms affect creative generation, we compared generated sonnets qualitatively across variants. In all cases, the first three lines are provided as context and the model generates the remainder.

Flash attention quality matches standard attention. As expected from the mathematical equivalence of how weights are calculated during training in flash and standard attention, both variants maintain Shakespearean diction and thematic coherence throughout the sonnets, and both are structured with indented couplets in many of the outputs. Any differences we observed between the sonnets generated from each variant are attributable to random seed variation as opposed to any systematic effect.

Sliding window attention results in the greatest quality degradation. First, line structure collapses: with $w = 64$, the model cannot attend to the beginning of the poem when generating the final portion of the sonnet, which leads to run-on phrases as it loses awareness of how often lines should break. The following generated example represents the loss of syllabic structure (usually lines are about 10 syllables, while this one is closer to 30): *"Is 't not enough to torture me alone, is not enough my friend one touches at deeds if the bones of mine mind need him?"*. Second, the model’s ability to maintain consistent themes across the sonnet deteriorates. Without access to the full poem context, the model cannot continue with the argument established in the opening lines, and ends up generating

content 5-8 lines later that is completely disconnected from the initial theme. None of the sonnets that were generated by sliding window attention exhibited a recognizable couplet structure.

Mixed local/global attention recovers qualitative performance significantly. When we alternated between full-attention layers and sliding window layers, every other layer offered access to long-range context. Subsequently, when we ran sonnet generation, we found that the line lengths returned to their more characteristic Shakespearean lengths, the poem had full thematic consistency, and the distinct couplet structure reappeared. While the mixed attention sonnets did not fully match the quality of those generated with standard attention (the meter remained somewhat irregular and occasional drift occurred), it represented a clear qualitative improvement over solely sliding window attention. This demonstrates that periodic global attention is necessary for coherent long-form generation.

The findings of our qualitative analysis reinforce the sentiment classification results in which model performance suffers with sliding window attention on long-context inputs. This failure mode is again directly observable in the generated sonnet text, where the model visibly loses the thread of what it is writing when restricted to local context windows.

Table 7 summarizes the key qualitative differences across attention mechanisms.

Table 7: Qualitative comparison of sonnet generation across attention variants

Property	Standard	Flash	Sliding Window	Mixed
Shakespearean diction	✓	✓	Partial	✓
Controlled line length	Partial	Partial	×	✓
Thematic coherence	Partial	Partial	×	Partial
Couplet structure	Partial	Partial	×	Partial

8 Conclusion

Based on our results and analysis, we concluded that flash attention is a strict improvement over standard attention, given that it sacrifices effectively no performance by generating mathematically equivalent results, but offers up to 40% lower latency and 34% less memory at sequence length 1024. Sliding window attention gave no wall-clock gains, and our efficient loop-based implementation is slower at short sequences due to Python loop overhead, but it does achieve genuine $O(T \cdot w)$ memory scaling at the attention layer, with a $4\times$ reduction in attention-layer memory at sequence length 1024 (116 MB vs. 467 MB, $w = 64$). This gain is not reflected in full forward-pass memory due to model weights dominating. Sliding window degrades accuracy on long-context tasks (CFIMDB -10.7 points and sonnet generation structural collapse). Our global token ablation analysis shows that adding just 2 global tokens recovers CFIMDB accuracy from 0.939 to 0.980, matching full attention. Mixed local/global attention recovers most of the accuracy loss (CFIMDB: 0.947 vs. 0.980) from sliding window attention and significantly restores sonnet quality, evidencing that periodic global attention helps to preserve long-range modeling capacity in most applications.

The primary limitation of our project is that sliding window latency savings require fused custom CUDA kernels to realize in practice [2]. Our "efficient" loop-based implementation achieves the memory scaling, but not the speed scaling of $O(T \cdot w)$ attention. An extension of our project might be to try to combine the flash attention's tiling with the sliding window mask in a single kernel. Our results show that flash improves latency and memory, while sliding window gets asymptotic memory scalings with its sparsity, so a combined implementation would integrate both benefits. Another possible extension would address how our loop-based efficient sliding window achieves $O(T \cdot w)$ memory scaling, but also carries with it the loop overhead that reduces the wall-clock gains on short sequences. A Triton or custom CUDA kernel that combines the windowed matmul could give us the memory savings without sacrificing latency.

Team Contributions

Lily and Devon split work evenly throughout the entire project. Both contributed to the codebase via Github by implementing different functions, and both wrote and polished this report.

References

- [1] Tri Dao, Daniel Y Fu, and Stefano Ermon. Flashattention: Fast and memory-efficient exact attention. *arXiv preprint arXiv:2205.14125*, 2022.
- [2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

A Appendix

A.1 Training Hyperparameters

Table 8: Training hyperparameters for each task

Hyperparameter	Classifier (last-layer)	Classifier (full)	Paraphrase	Sonnet
Epochs	10	5	5	10
Learning rate	1×10^{-3}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Batch size	8	8	8	8
Optimizer	AdamW	AdamW	AdamW	AdamW
Weight decay	0.01	0.01	0.01	0.01

A.2 Sentiment Classification Results

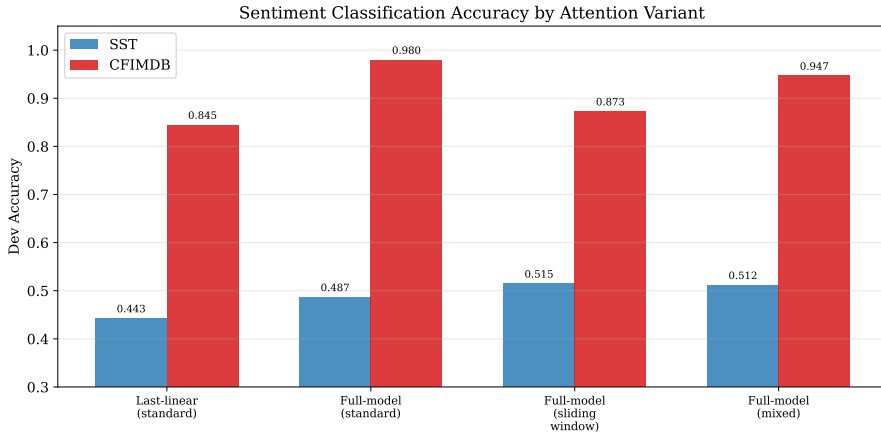


Figure 1: Dev set accuracy on SST and CFIMDB across attention variants. All variants perform comparably on SST (short sentences), but sliding window degrades notably on CFIMDB (longer reviews), with mixed attention partially recovering this loss.

A.2.1 Window Size Ablation

To understand how the size of the local context window impacts sentiment classification, we conducted a window size ablation on CFIMDB, since the reviews in this dataset routinely exceed hundreds of tokens (unlike SST where mostly short sentences would fall entirely within most context window sizes). We used sliding window attention with local context window sizes of $w \in \{16, 32, 64, 128, 256\}$ and compared these outcomes against standard full causal attention as a baseline.

As shown in Table 9, accuracy degrades monotonically as the window size shrinks, dropping from 97.96% under full attention to 62.86% at $w = 16$, which marks a 35-point reduction. We see that

$w = 256$ matches full attention exactly which is likely due to the fact that CFIMDB reviews are rarely longer than 256 tokens. The steepest drop occurs between $w = 32$ and $w = 16$ (15.5 points), suggesting that 32 tokens falls below the minimum span needed for sentiment-bearing clauses. These results confirm that long-range dependencies are critical for sentiment classification across longer documents.

Table 9: Window size ablation on CFIMDB (full-model fine-tune). Restricting the local window monotonically degrades accuracy on long movie reviews.

Window Size	CFIMDB Dev Acc
Full (standard)	0.980
$w = 256$	0.980
$w = 128$	0.939
$w = 64$	0.861
$w = 32$	0.784
$w = 16$	0.629

A.3 Efficiency Benchmark Plots

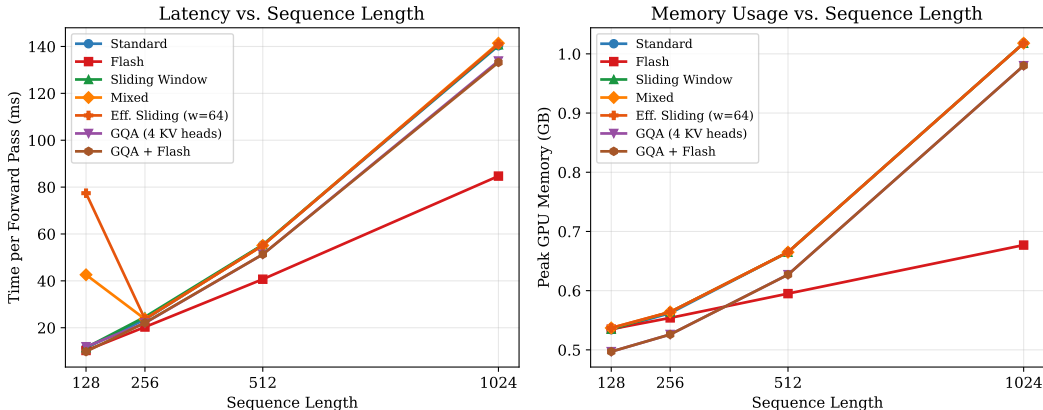


Figure 2: Latency (ms) and peak GPU memory (GB) of the full forward pass across sequence lengths on an NVIDIA A10G GPU. Flash Attention diverges from standard attention at longer sequences, while masked sliding-window attention remains close to standard because the full score matrix is still materialized before masking. The efficient loop-based variant reduces attention-layer memory in isolation, but does not improve end-to-end runtime and introduces overhead at short sequence lengths due to Python loop execution.

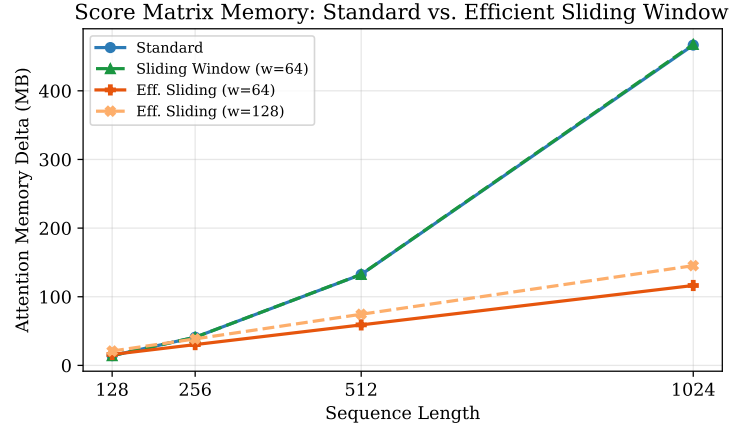


Figure 3: Attention-layer memory footprint (MB) isolated from model weights, measured as peak memory allocated during a single `CausalSelfAttention` call. Standard and masked sliding window attention both materialize the full $T \times T$ score matrix, growing quadratically with sequence length (466 MB at $T=1024$). Our loop-based efficient sliding window computes scores only within a local window of size w , achieving $O(T \cdot w)$ memory and a $4\times$ reduction at $T=1024$, $w=64$ (116 MB vs. 466 MB). Savings grow linearly with sequence length, so at $T=4096$ the reduction should exceed $16\times$.

A.4 Paraphrase Detection Error Analysis

With 89.6% dev accuracy on paraphrase detection, the model makes errors in two systematic categories.

False positives (predicted paraphrase, actually not):

- *"How much does it cost to purchase one cotton bale?"* vs. *"Where can I purchase cotton lint in bulk?"*. These phrases are thematically related (both about cotton purchasing), but are asking fundamentally different questions. The model likely attends to overlapping content words (such as "cotton" and "purchase") without distinguishing question intent.
- *"What is the best resource to learn CosmoMC?"* vs. *"What are the best books to learn Java?"*. These phrases utilize the same syntactic template (*"What is the best [X] to learn [Y]?"*), but have completely different subject matters. This suggests that the model is sensitive to surface syntactic similarity independent of semantic content.

False negatives (missed real paraphrases):

- *"What do Non-Muslim women think about Hijab or burqa?"* vs. *"What do non-muslims think about the hijab?"*. This is a true paraphrase where the second question uses broader terminology. The model may fail when paraphrase requires recognizing that a subset term ("Non-Muslim women") implies the superset ("non-muslims").
- *"Why do people believe in flat earth?"* vs. *"How is it possible that anowadays there is someone believing in flat earth?"*. This is again a true paraphrase, but it uses different syntactic framing (why vs. how is it possible). This suggests that reformulations that change the interrogative structure appear to challenge the model even when the semantic content is the same.

These error patterns reveal that the model conflates the similarity of topics with semantic equivalence (false positives), and also struggles with paraphrases that use substantially different surface forms (false negatives). Both of these failure modes suggest that the model's classification performance is influenced more so by lexical overlap rather than deep semantic understanding. This takeaway is consistent with the observed parity between attention variants on paraphrase detection. Since question pairs are short, the limiting factor is not context length, but rather representational depth.