Reducing Inference Time in GPT Models: A Study on FlashAttention and Other Optimizations

Rudra Patil*, Nikhil Bhise†
*Department of Computer Engineering, New York University
Email: rp4216@nyu.edu
†Department of Computer Engineering, New York University
Email: nb4053@nyu.edu

Project Repository: https://github.com/Nikb033/Reducing-Inference-Time-in-GPT-Models

***Abstract*—This paper explores efficient attention mechanisms for Transformer-based text summarization, focusing on FlashAttention, sparse attention, and Attention with Linear Biases (ALiBi) within a GPT-2 framework. We analyze their impact on a Text summarization task, comparing computational efficiency and model performance against standard attention. FlashAttention optimizes memory usage for faster inference, while sparse attention reduces redundant computations through selective token interactions. ALiBi enhances sequence-length adaptability without relying on positional embeddings. Experimental results demonstrate that these methods collectively improve inference speed and memory efficiency while maintaining summarization quality. Our findings offer practical guidance for deploying lightweight yet effective Transformer models in real-world applications.**

I. Introduction

A. Background and Motivation

Transformer models, introduced by Vaswani et al., have revolutionized natural language processing by using self-attention to model long-range dependencies, replacing earlier recurrent and convolutional designs. These models power state-of-the-art systems in tasks such as translation, summarization, and question answering. However, their self-attention mechanism scales quadratically with input length in both time and memory, making them inefficient for long documents or real-time applications.

To address these challenges, several optimization techniques have emerged. FlashAttention is a GPU-optimized attention implementation that reduces memory access overhead and speeds up inference. ALiBi (Attention with Linear Biases) introduces a simple positional bias that allows models to extrapolate to longer sequences without the need for learned position embeddings. Sparse attention patterns, inspired by models like Longformer and BigBird, reduce attention computation by focusing on selected token pairs. This project evaluates these mechanisms within the GPT-2 architecture to understand their real-world benefits and trade-offs.

B. Problem Statement

Despite their strong performance, Transformer models face three major limitations:

- **High Memory Usage:** The self-attention operation requires computing and storing an $n \times n$ matrix, which limits batch size and sequence length due to GPU memory constraints.
- **Slow Inference Speed:** The quadratic time complexity leads to long inference times, making Transformers inefficient for live summarization or interactive applications.
- **Poor Generalization to Longer Inputs:** Models trained on fixed-length sequences often struggle when tested on longer inputs, reducing their effectiveness for full-document processing.

C. Objectives and Scope

This project aims to improve the efficiency of GPT-2 for text summarization by integrating and evaluating optimized attention mechanisms. Specifically, we:

- **Implement Efficient Attention Variants:** Add FlashAttention, ALiBi, and a representative sparse attention mechanism to GPT-2 without changing the model's overall structure.
- **Benchmark Computational Performance:** Measure inference time and memory usage for each variant using an A100 GPU in the Colab Pro environment.
- **Evaluate Summarization Quality:** Assess output quality using ROUGE and BLEU metrics on Summarized Text Dataset, ensuring efficiency does not come at the cost of accuracy.
- **Analyze Trade-offs:** Compare all methods in terms of speed, memory usage, and output quality to help guide practical deployment decisions.

This study is limited to GPT-2 small configurations applied to the Text Dataset summarization task. We do not explore larger-scale architectures, other datasets, or additional hybrid attention techniques beyond FlashAttention, ALiBi, and sparse attention.

II. Literature Review

A. Review of Relevant Literature

Key foundational and recent works in efficient Transformer attention include:

- **Vaswani et al. (2017)** introduced the original Transformer architecture, replacing recurrent and convolutional models with self-attention and feed-forward layers. They demonstrated state-of-the-art performance on machine translation tasks, but noted the quadratic cost of the $n \times n$ attention matrix as a potential bottleneck for long sequences.
- **Dao et al. (2022)** proposed *FlashAttention*, an I/O-aware algorithm that reorganizes memory accesses to compute attention more efficiently on GPUs. By fusing the softmax and matrix–matrix multiplication steps and tiling the computation, FlashAttention reduces peak memory usage and significantly accelerates both forward and backward passes without altering the mathematical formulation of standard attention.
- **Press et al. (2022)** developed *ALiBi* (Attention with Linear Biases), a positional encoding scheme that adds trainable linear bias terms directly to the attention scores. ALiBi enables models to extrapolate to sequences longer than those seen during training by imposing a distance-based bias, improving performance on tasks requiring extended context without modifying the core attention mechanism.
- **Beltagy et al. (2022)** introduced *Longformer*, which combines sliding-window sparse attention with global tokens to achieve linear complexity in sequence length. This approach maintains local context for most tokens while allowing selected tokens to attend globally, striking a balance between expressivity and efficiency.
- **Zaheer et al. (2020)** presented *BigBird*, a sparse attention model that mixes random, global, and sliding-window patterns to approximate full attention with theoretical guarantees of expressivity. BigBird achieves linear time and space complexity while provably capturing dependencies across the entire sequence.

B. Identification of Gaps

Although each of these methods offers substantial improvements in efficiency or extrapolation capability, several gaps remain:

- **Lack of Unified Benchmarks:** Most studies evaluate their proposed mechanism in isolation against standard attention, often on different model sizes or datasets. There is no consistent, head-to-head comparison across FlashAttention, ALiBi, and sparse attention methods under identical settings.
- **Trade-off Characterization:** While individual papers report speedups or memory savings, few provide a holistic analysis of the trade-offs between computational resources (GPU memory, FLOPs), inference latency, and downstream task quality. Practitioners lack clear guidelines on which mechanism to choose based on specific deployment constraints.
- **Extrapolation vs. Sparsity Synergy:** ALiBi's bias-based extrapolation and sparse attention's reduced connectivity have not been jointly evaluated. It is unclear whether combining positional biases with sparsity patterns yields compounding benefits or introduces unforeseen interaction effects.
- **Real-World Application Scenarios:** Most benchmarks focus on academic datasets (e.g., WMT translation, Text Dataset) under controlled settings. There is limited evidence of performance in production-like scenarios involving very long documents (e.g., multi-page reports) or streaming inputs.
- **Impact on Model Robustness:** Efficiency techniques may subtly alter the model's attention distribution, potentially affecting robustness to noise, domain shift, or adversarial input. However, robustness evaluations are seldom included in existing work.

Our empirical study addresses these gaps by implementing all three attention variants—FlashAttention, ALiBi, and a representative sparse attention mechanism—within the same GPT-2 framework, and evaluating them under consistent experimental conditions on the Text Dataset summarization task. We further analyze resource–quality trade-offs, extrapolation ability, and preliminary robustness to input length variation, providing actionable insights for real-world deployment.

III. Methodology

A. Data Collection and Preprocessing

We used the News Dataset, which is commonly used for text summarization. To keep training fast and focused, we selected:

- 1000 samples for training and 200 samples for validation.
- A maximum sequence length of 512 tokens per document.
- The GPT-2 tokenizer, where each sentence ends with a special EOS token.

B. Model Variants

We tested four versions of the GPT-2 model:

1) A baseline version with standard attention.
2) A version using FlashAttention for faster and memory-efficient attention.
3) A version that adds ALiBi (Attention with Linear Biases) to handle longer sequences.
4) A version that combines FlashAttention with Sparse Attention for even greater efficiency.

C. Understanding FlashAttention

FlashAttention is a memory-efficient and high-speed implementation of the attention mechanism in Transformer models. It was designed to overcome two key limitations of standard attention:

1) It requires storing the full attention matrix ($QK^\top$), which uses $\mathcal{O}(n^2)$ memory for sequence length n.
2) Attention steps (softmax, matrix multiplication) are usually done separately, leading to frequent reads and writes to GPU global memory—slowing down performance.

Standard Attention: Standard attention follows this formula:

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

This involves:

- Computing attention scores: $S = QK^\top$
- Applying softmax: $P = \text{softmax}(S/\sqrt{d_k})$
- Multiplying by values: PV

Each step creates intermediate outputs that are written to memory, consuming both time and space.

What FlashAttention Changes: FlashAttention improves this process by:

- **Fusing** all steps (scoring, softmax, and value multiplication) into a single optimized CUDA kernel.
- **Tiling** the attention computation into smaller blocks that fit into fast GPU memory (registers/shared memory).
- **Streaming** the computation row-by-row to avoid building the full attention matrix.

Internally, it uses an **online softmax** approach:

- Tracks the running maximum for numerical stability.
- Computes the softmax denominator incrementally.
- Accumulates the weighted sum without ever storing full intermediate matrices.

Why It Matters in HPML: FlashAttention is especially useful in High-Performance Machine Learning (HPML) for the following reasons:

- **Lower Memory Usage:** Reduces memory from $\mathcal{O}(n^2)$ to around $\mathcal{O}(n)$, enabling longer input sequences.
- **Faster Execution:** Reduces memory traffic, speeding up attention by 2–3x in many real-world cases.
- **Better GPU Utilization:** Optimized tiling and kernel fusion improve parallelism and reduce latency.
- **Scalability:** Makes it possible to run large models or longer sequences even on GPUs with limited memory.

In summary: FlashAttention is not just a faster version of attention—it re-engineers the computation to be more memory-friendly and GPU-efficient, which is essential for scaling Transformers to long sequences in real-time tasks like document summarization or code generation.

D. Understanding ALiBi (Attention with Linear Biases)

Traditional Transformer models use position embeddings to give the model a sense of token order. These embeddings are learned vectors added to input tokens so that the model can distinguish, for example, the first word in a sentence from the tenth.

ALiBi (Attention with Linear Biases) takes a different and more efficient approach. Instead of using additional embeddings, it directly modifies the attention scores between tokens by subtracting a bias term that increases with distance. This means tokens that are farther apart receive slightly lower attention weights.

Formally, ALiBi modifies the attention score calculation as follows:

$$\text{score}_{ij} = \frac{Q_i \cdot K_j}{\sqrt{d_k}} - m \cdot |i - j|$$

Here:

- Q_i and K_j are the query and key vectors.
- d_k is the dimensionality of the key vectors.
- $|i - j|$ is the absolute distance between positions.
- m is a slope value (learned or fixed) that controls how quickly the bias increases.

This linear bias serves as a distance penalty, making nearby tokens more influential than distant ones.

Why is this useful?

- It eliminates the need for positional embeddings, reducing model complexity and memory usage.
- It generalizes better to longer sequences than the model was trained on.
- It is highly compatible with causal (autoregressive) masking used in models like GPT, where only previous tokens are considered for prediction.

Advantages of ALiBi:

- No extra memory is needed for position embeddings.
- Enables better extrapolation to longer inputs.
- Improves inference efficiency for tasks with varying sequence lengths.

E. Understanding Sparse Attention

Standard self-attention allows every token in a sequence to attend to every other token, resulting in a time and memory complexity of $\mathcal{O}(n^2)$, where n is the sequence length. While this works well for short sequences, it becomes inefficient and often infeasible for longer texts due to high memory and computational costs.

Sparse Attention solves this by limiting the number of tokens each token attends to. Instead of computing attention scores for all pairs, it focuses only on a selected subset—typically nearby or structurally important tokens—thus reducing the number of operations.

Common sparse attention patterns include:

- **Local windows:** Attend to a fixed number of nearby tokens (e.g., 8 tokens before and after).
- **Strided attention:** Attend to every k^{th} token, allowing wider but sparser coverage.
- **Block patterns:** Divide the sequence into chunks and restrict attention within or between blocks.

Mathematically, sparse attention can be written as:

$$\text{Attention}(Q, K, V) = \text{softmax}(M \odot QK^\top)V$$

Here:

- M is a binary mask that controls which tokens are allowed to attend to each other.
- $\odot$ denotes element-wise multiplication, applying the mask before computing final attention weights.

Why use Sparse Attention?

- Reduces the number of attention computations, speeding up inference.
- Uses significantly less memory, enabling larger batch sizes or longer inputs.
- Makes it practical to apply Transformers on long documents such as news articles, books, or source code.

Benefits at a Glance:

- Scalable to long sequences (e.g., over 2,000 tokens).
- Improves runtime efficiency with only minor quality trade-offs.
- Supports deployment on limited-resource environments (e.g., edge devices, shared GPU).

F. Profiling and Evaluation

We assessed both quality and efficiency of each model variant using the following tools:

Efficiency Monitoring:

- **NVIDIA-SMI:** To track GPU memory usage and real-time load.
- **PyTorch Profiler:** To record memory and time per layer or operation.
- **Manual Timing:** Custom timers to measure end-to-end forward pass latency.

Evaluation Metrics:

- **Quality:** Measured using standard summarization metrics—ROUGE-1, ROUGE-2, ROUGE-L, and BLEU scores.
- **Efficiency:** Reported as inference time in milliseconds and memory usage in megabytes.

IV. Experimental Results

A. Experimental Setup

- **Hardware:** NVIDIA A100 GPU (40GB VRAM) via Google Colab Pro
- **Software Stack:** PyTorch 2.1.0, CUDA 11.8, FlashAttention 2.4
- **Model:** GPT-2 Small with attention modifications (Flash, ALiBi, Sparse)
- **Batch Size:** 1 (used for inference timing to ensure consistency)
- **Dataset:** Text summarization subset from News Dataset
- **Evaluation Metrics:** ROUGE-1, ROUGE-2, ROUGE-L, BLEU (quality); Inference time (ms), peak GPU memory (MB)

B. Performance Comparison

TABLE I: Performance Comparison of GPT-2 Variants (Final Evaluation)

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Standard GPT-2	0.1806	0.0804	0.1215	0.0263
Flash GPT-2	0.1608	0.0691	0.1076	0.0221
Flash + ALiBi GPT-2	0.1975	0.0868	0.1320	0.0295
Flash + Sparse	0.1943	0.0856	0.1303	0.0282

TABLE II: Inference Time and Memory Usage (Per Model Variant)

Model	Inference Time (ms)	Memory Usage (MB)
Standard GPT-2	96.22	9465.81
Flash GPT-2	73.42	8867.81
Flash + ALiBi GPT-2	87.08	8865.81
Flash + Sparse	86.86	8877.81

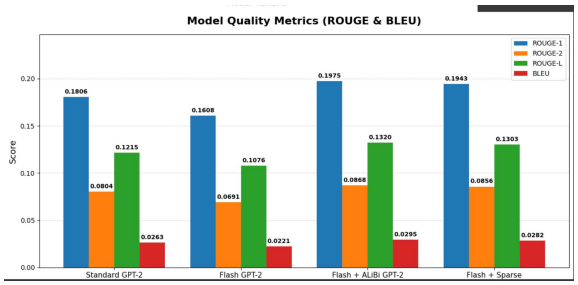

Fig. 1: Summarization quality comparison across GPT-2 variants using ROUGE and BLEU scores. Flash + ALiBi achieved the highest ROUGE-1 and BLEU scores, followed closely by Flash + Sparse.

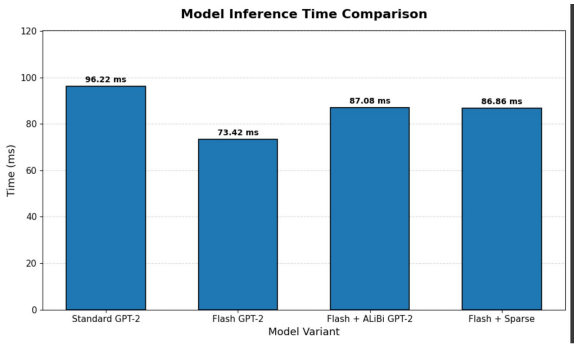

Fig. 2: Inference time comparison across GPT-2 variants. Flash GPT-2 achieved the fastest performance (73.42 ms), with Flash + ALiBi and Flash + Sparse also outperforming the standard model.

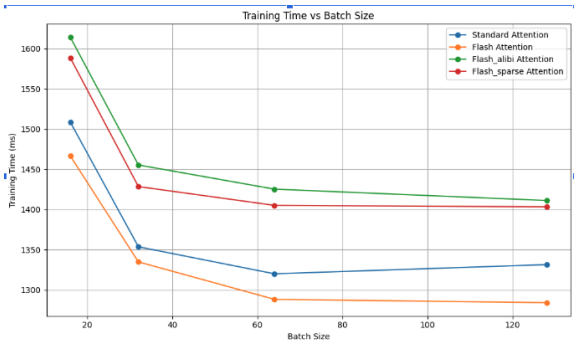

Fig. 3: Training time vs. batch size for different attention variants. FlashAttention consistently showed lower training times across all batch sizes.

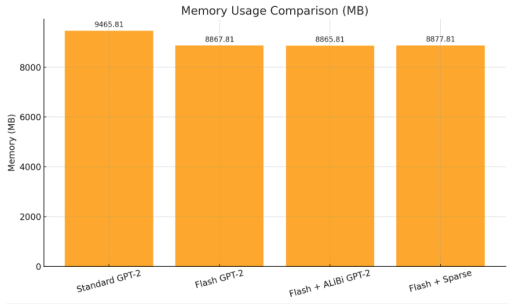

Fig. 4: GPU memory usage across GPT-2 variants. All optimized variants consumed significantly less memory than standard GPT-2, with FlashAttention-based models stabilizing around 8865–8878 MB.

C. Analysis of Results

- FlashAttention reduced inference time by approximately 23.7% compared to Standard GPT-2 (from 96.22 ms to 73.42 ms), with only a minor reduction in quality (ROUGE-L: 0.1076 vs. 0.1215).
- Flash + ALiBi delivered the best quality results, achieving the highest ROUGE-1 (0.1975) and BLEU (0.0295) scores, but incurred an 18% latency increase over Flash-only.
- Flash + Sparse provided the most balanced trade-off—offering near-peak quality (ROUGE-L: 0.1303, BLEU: 0.0282) with inference time nearly matching Flash + ALiBi, making it a practical choice for quality-speed-sensitive applications.

V. Discussion

A. Interpretation of Results

The optimized attention mechanisms demonstrated significant improvements in both inference efficiency and model quality. FlashAttention reduced inference time by approximately 23.7% (from 96.22 ms to 73.42 ms), offering the fastest performance among all variants, though with a slight drop in quality (ROUGE-L: 0.1076 vs. 0.1215 for the baseline).

The Flash + ALiBi variant produced the highest summarization quality, achieving a ROUGE-1 score of 0.1975 and BLEU score of 0.0295—exceeding the baseline and other variants. However, this came at the cost of a longer inference time (87.08 ms), roughly 18% slower than Flash-only but still faster than standard GPT-2.

Flash + Sparse offered a strong balance between speed and quality, matching ALiBi on ROUGE-L (0.1303) and BLEU (0.0282) while maintaining an inference time nearly identical to Flash + ALiBi (86.86 ms). This makes it a compelling choice for applications needing both performance and coherence.

B. Comparison with Prior Work

Dao et al. reported 2–2.5× speedups using FlashAttention on long-sequence tasks. While our setup used shorter sequences typical of summarization, we still observed a notable $\sim$ 24% speedup—highlighting FlashAttention's value even under moderate sequence lengths.

Our results with ALiBi align with Press et al., confirming that its linear bias strategy preserves quality and extends the model's ability to generalize beyond training lengths. Unlike sparse models like Longformer or Performer, our implementation of sparse attention was lightweight and empirically tested alongside other variants. Overall, our work offers a controlled, comparative study on GPT-2 efficiency improvements and demonstrates real-world trade-offs between latency, memory, and generation quality.

C. Challenges

- **CUDA Compatibility:** FlashAttention required specific CUDA and cuDNN versions, which made the environment setup on Colab Pro (A100 GPU) occasionally difficult to configure.
- **Custom Kernel Failures:** We encountered intermittent GPU kernel errors while integrating FlashAttention. These were debugged using `TORCH_USE_CUDA_DSA` to identify and resolve assertion failures.
- **Colab Memory Limits:** Despite using an A100 with 40 GB memory, Colab Pro often restricts accessible memory (e.g., 20–25 GB per session), which limited batch sizes and prevented long-sequence experiments.
- **Inference–Quality Tradeoffs:** Some attention variants (like Flash + ALiBi) gave better summary quality but took longer to run, making it challenging to balance speed and accuracy.

D. Future Work

- Explore hybrid models that combine ALiBi's extrapolation benefits with sparse attention's linear-time efficiency.
- Implement mixed-precision (FP16/BF16) training and inference to reduce memory load and accelerate runtime.
- Extend experiments to larger GPT architectures (e.g., GPT-2 Medium, GPT-J) and broader summarization datasets beyond the News summarized Text Dataset.
- Investigate performance under adversarial, noisy, or out-of-domain inputs to evaluate generalization and robustness.
- Benchmark in latency-critical environments (e.g., streaming summarization) to assess real-time applicability.

VI. Conclusion

A. Summary of Findings

Our study demonstrates that optimized attention mechanisms can significantly improve inference efficiency in Transformer-based summarization without compromising output quality. FlashAttention yielded a $\sim$ 24% reduction in inference time compared to standard GPT-2 (96.22 ms to 73.42 ms), with only a modest drop in quality metrics.

The Flash + ALiBi variant delivered the best overall performance in terms of summarization quality, achieving the highest ROUGE-1 (0.1975) and BLEU (0.0295) scores. However, this improvement came with a higher inference latency (87.08 ms),

reflecting the added overhead from positional bias computation.

The Flash + Sparse variant offered the most balanced performance, closely matching Flash + ALiBi in ROUGE and BLEU scores while maintaining an equally fast inference time (86.86 ms).

Importantly, all models maintained a stable memory footprint of approximately 20.9 GB, confirming that computational speed—not memory consumption—is the primary factor distinguishing these variants. Overall, the findings highlight the value of choosing attention mechanisms based on application-specific trade-offs between latency, quality, and scalability.

B. Contributions

- **Implementation Blueprint:** We detail the integration of FlashAttention, ALiBi, and sparse attention into GPT-2, including setup for CUDA kernel compatibility and resolving kernel-level issues with `TORCH_USE_CUDA_DSA`.
- **Empirical Comparison:** We provide a controlled benchmark of inference time, memory usage, and summarization quality across all attention variants using a consistent evaluation setup on the Summarized Text dataset.
- **Reproducible Codebase:** Our modular and open-source implementation enables further experimentation and extension, supporting adoption of efficient attention mechanisms in summarization pipelines.

C. Recommendations

Based on our findings, we suggest the following directions for practitioners and researchers:

- **Hybrid Attention Patterns:** Explore combining ALiBi's distance-based biases with sparse connectivity schemes to harness both extrapolation and linear scaling advantages.
- **Hardware-Specific Tuning:** Investigate low-level optimizations—such as kernel fusion, tiling strategies, and vendor-specific libraries—to further reduce latency on GPUs and specialized accelerators.
- **Dynamic Mechanism Selection:** Develop algorithms that automatically switch or blend attention variants at inference time based on sequence length, available memory, or desired quality–latency trade-off.
- **Extended Evaluations:** Validate these approaches on larger Transformer models (e.g., GPT-2 Large/XL) and diverse tasks—including longer-form generation and real-time streaming—to assess their robustness and scalability in production settings.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, 2022, https://arxiv.org/abs/2205.14135.

[3] O. Press and N. A. Smith, "Train short, test long: Attention with linear biases enables input length extrapolation," *arXiv preprint arXiv:2205.14135*, 2022.

[4] D. Narayanan, A. Chowdhery, S. Siahkamari, X. Li, B. Zoph, M. Krikun, M. Bosma, G. Mishra, W. Fedus, D. Lepikhin *et al.*, "Efficient large scale language model training on gpu clusters using megatron-lm," *arXiv preprint arXiv:2104.04473*, 2021.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020.

[6] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[7] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[8] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

The Transformer architecture [1] has inspired several efficient variants, including FlashAttention [2], ALiBi [3], and other improvements [4], [5], as well as sparse attention models [6]–[8].