

FlashAttention: An IO-Aware Approach to Fast and Memory-Efficient Transformers

The quadratic time and memory complexity of the self-attention mechanism in Transformers presents a major bottleneck, hindering their application to long sequences. While approximate attention methods aim to reduce computational complexity (FLOPs), they often fail to deliver practical wall-clock speedups because they overlook the primary performance constraint on modern GPUs: memory access (IO).

The core insight presented is that a missing principle in designing efficient algorithms is **IO-awareness** the careful management of data movement between slow, high-capacity GPU High Bandwidth Memory (HBM) and fast, small on-chip SRAM.

FlashAttention is a new, IO-aware exact attention algorithm that fundamentally restructures the computation to dramatically reduce the number of HBM reads and writes. By leveraging two key technique **tiling** and **recomputation**—FlashAttention computes exact attention without ever materializing the large $N \times N$ attention matrix in HBM. This results in an algorithm that is both significantly faster and more memory-efficient, scaling linearly with sequence length in memory usage.

Empirical validation demonstrates that FlashAttention delivers substantial end-to-end training speedups, outperforming the MLPerf record for BERT-large by 15% and accelerating GPT-2 training by up to 3 \times . By enabling longer context, it yields higher-quality models, improving perplexity on GPT-2 and achieving new state-of-the-art results on long-document classification.

Critically, FlashAttention enables the first Transformer models to achieve better-than-chance performance on the challenging Path-X (16K sequence length) and Path-256 (64K sequence length) benchmarks. An extension, **Block-sparse FlashAttention**, further improves speed and is faster than any existing approximate attention method tested.

The Core Problem: The IO Bottleneck in Standard Attention

Standard self-attention has a time and memory complexity of $O(N^2)$, where N is the sequence length. This quadratic scaling makes processing long sequences computationally prohibitive. The standard implementation explicitly materializes the large $N \times N$ attention and softmax matrices (S and P) in GPU HBM.

On modern hardware, the speed of computation has significantly outpaced memory bandwidth. Consequently, many operations in deep learning, including key parts of the attention mechanism like softmax and elementwise operations, are **memory-bound**. The runtime of these operations is dictated not by the number of arithmetic calculations (FLOPs), but by the time spent reading and writing data to and from HBM.

Many approximate attention methods have been proposed to address this issue by reducing FLOPs to linear or near-linear complexity. However, they frequently fail to achieve wall-clock speedups because they do not account for the significant overhead of memory access. They

introduce complex data access patterns that can be inefficient, neutralizing their theoretical computational gains.

FlashAttention: An IO-Aware Algorithm

FlashAttention is an exact attention algorithm designed to minimize HBM access by restructuring the computation. Its main objective is to avoid reading and writing the large $N \times N$ attention matrix to and from HBM. This is achieved through a combination of tiling, recomputation, and kernel fusion.

Methodology

- **Tiling:** The algorithm splits the input matrices (Q , K , V) into smaller blocks. It then loads blocks of K and V from HBM into the much faster on-chip SRAM. In an inner loop, it iterates through blocks of Q , loading them into SRAM and computing the attention output with respect to the blocks of K and V already in SRAM. The softmax operation is computed incrementally across these blocks using a numerically stable online scheme, which maintains running normalization statistics to produce the correct final output without needing the full attention matrix.
- **Recomputation:** To maintain memory efficiency during the backward pass, FlashAttention avoids storing the $N \times N$ intermediate attention matrices. Instead, it stores only the final output (O) and the small softmax normalization statistics (ℓ, m) from the forward pass. During backpropagation, it recomputes the necessary attention matrix blocks on-the-fly in fast SRAM. While this increases the number of FLOPs, it is significantly faster than the standard approach of reading the entire intermediate matrix from slow HBM. This technique can be viewed as a form of selective gradient checkpointing that trades increased computation for reduced memory IO, leading to a net speedup.
- **Kernel Fusion:** All attention operations—matrix multiplication, masking, softmax, and dropout are fused into a single CUDA kernel. This ensures that inputs are loaded from HBM only once, and intermediate results are kept within registers and SRAM, eliminating the need for multiple round-trips to HBM that would occur in a standard, multi-kernel implementation.

Theoretical Performance and Complexity Analysis

FlashAttention's IO-aware design leads to a provable reduction in HBM accesses compared to standard attention.

Metric	Standard Attention	FlashAttention
FLOPs	$O(N^2d)$	$O(N^2d)$
HBM Accesses	$\Theta(Nd + N^2)$	$\Theta(N^2d^2M^{-1})$
Extra Memory	$O(N^2)$	$O(N)$

(Where N is sequence length, d is head dimension, and M is SRAM size)

The IO complexity of FlashAttention, $\Theta(N^2d^2M^{-1})$, is significantly lower than the $\Theta(N^2)$ term of standard attention for typical hardware configurations. A formal analysis provides a lower

bound, showing that no exact attention algorithm can asymptotically improve on this number of HBM accesses for all possible SRAM sizes.

Experiments confirm that HBM access is the primary determinant of runtime. As shown in the table below (for GPT-2 medium, N=1024), FlashAttention has a higher FLOP count due to recomputation in the backward pass, but its massive reduction in HBM access leads to a much faster execution time.

Metric	Standard Attention	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

Extension: Block-Sparse FlashAttention

FlashAttention's principles can be extended to approximate attention methods to overcome their IO overheads. **Block-sparse FlashAttention** implements a sparse attention algorithm where computation is only performed for a predefined set of non-zero blocks in the attention matrix. The algorithm is identical to FlashAttention but simply skips the computation and memory access for the zero-blocks.

This results in an improved IO complexity of $\Theta(Nd + N^2d^2M^{-1}s)$, where s is the fraction of non-zero blocks. This provides a direct speedup proportional to the sparsity, making it faster than dense FlashAttention and, empirically, faster than all other existing approximate attention methods.

Empirical Validation and Key Results

FlashAttention was evaluated on model training speed, model quality from longer context, and raw attention layer performance.

Faster Model Training

- **BERT-large:** Achieved a 15% faster training time to reach the target accuracy compared to the NVIDIA implementation that set the MLPerf 1.1 training speed record.

BERT Implementation Training time (minutes)

Nvidia MLPerf 1.1 20.0 ± 1.5

FlashAttention (ours) 17.4 ± 1.4

- **GPT-2:** Delivered significant end-to-end speedups over highly optimized baselines while achieving identical perplexity.

Model implementations	Training time (speedup)
-----------------------	-------------------------

GPT-2 medium - Huggingface 21.0 days (1.0 \times)

GPT-2 medium - Megatron-LM 11.5 days (1.8 \times)

GPT-2 medium - FlashAttention 6.9 days (3.0 \times)

- **Long-Range Arena (LRA):** A vanilla Transformer using FlashAttention achieved a $2.4\times$ average speedup over standard attention. Block-sparse FlashAttention was even faster, with a $2.8\times$ speedup.

Higher-Quality Models via Longer Context

The efficiency of FlashAttention enables training with much longer sequences, leading to improved model quality and new capabilities.

- **Language Modeling:** A GPT-2 small model trained with a 4K context length using FlashAttention was 30% faster to train than a Megatron-LM model with a 1K context, while achieving 0.7 better perplexity.

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0 \times)
GPT-2 small - FlashAttention 4k		17.5	3.6 days (1.3\times)

- **Long Document Classification:** Increasing sequence length improved micro-F1 scores significantly on two datasets with long documents, yielding a 4.3 point lift on MIMIC-III (16K vs 512) and an 8.5 point lift on ECtHR (8K vs 512).

- **Path-X & Path-256:** On these challenging long-context reasoning tasks, where previous Transformers failed, FlashAttention and its sparse variant were the first to achieve better-than-random accuracy.

Model	Path-X (seq len 16K)	Path-256 (seq len 64K)
Previous Transformers	~7% (random)	~7% (random)
FlashAttention	61.4%	~7%
Block-sparse FlashAttention	56.0%	63.1%

Attention Layer Benchmarks

- **Runtime:** FlashAttention is up to $3\times$ faster than standard PyTorch attention for common sequence lengths (up to 2K). While some approximate attention methods become faster at very long sequence lengths (crossover at 1K), **Block-sparse FlashAttention is faster than all tested exact, approximate, and sparse attention baselines across all sequence lengths.**

- **Memory Footprint:** The memory usage of FlashAttention and Block-sparse FlashAttention grows linearly with sequence length, making them up to $20\times$ more memory-efficient than standard attention and more efficient than most approximate baselines.

Limitations and Future Directions

- **Implementation Complexity:** The current implementation requires writing low-level CUDA code, which demands significant engineering effort and may not be portable across different GPU architectures. A high-level language or compiler that can generate IO-aware implementations is a key area for future work.

- **Generalizing IO-Awareness:** The principles of IO-aware design can be applied beyond attention to other memory-intensive components of deep learning models, such as sparse MLP layers.

- **Multi-GPU Systems:** Extending this analysis to multi-GPU settings, which introduces another layer to the memory hierarchy (inter-GPU communication), could unlock further performance gains for large-scale model training.

How a Decades-Old Computing Trick Makes AI Models 3x Faster

Transformer models have revolutionized artificial intelligence, powering everything from large language models to image generation. Yet for all their power, they have a critical weakness: the computational and memory costs of their core 'self-attention' mechanism grow quadratically with the length of the input. This makes processing long sequences of text, high-resolution images, or other complex data prohibitively slow and expensive.

For years, researchers have tried to solve this by developing 'approximate' attention methods, which trade model quality for a reduction in complexity. The problem is that these methods often fail to deliver real-world speedups on modern hardware and can degrade the model's performance.

The breakthrough, as presented in the research behind FlashAttention, comes from a surprising realization. The true bottleneck isn't just the sheer number of calculations (FLOPs) a GPU has to perform. The real performance killer is the traffic between two specific tiers of GPU memory: the constant, slow data transfer between the GPU's massive but relatively slow High Bandwidth Memory (HBM) and its tiny, ultra-fast on-chip SRAM. This article explores the powerful and counter-intuitive takeaways from FlashAttention, an algorithm that solves this problem by being "IO-aware."

1. More Math, Less Time: The Paradox of Faster Performance

The standard approach to calculating attention requires the creation of a massive intermediate matrix (the $N \times N$ attention matrix, where N is the sequence length). Because this matrix is too large to fit in the GPU's fastest memory, it must be written out to the slower HBM before it can be used in the next step. This round-trip to slow memory is a major I/O cost that stalls the entire process.

FlashAttention uses a clever alternative: it completely avoids writing and reading this massive matrix. Instead, it breaks the computation into smaller pieces and, for the backward pass (used during model training), it simply *recomputes* the necessary parts of the attention matrix on the fly.

This leads to a central paradox, highlighted by the paper's performance data. Even though FlashAttention performs *more* floating-point operations (75.2 GFLOPs) than

standard attention (66.6 GFLOPs) during the combined forward and backward pass, it is dramatically faster—completing the operation in 7.3 milliseconds versus standard attention's 41.7 milliseconds. This trade-off is effective because on modern GPUs, where compute speed has dramatically outpaced memory speed, the penalty for a few extra calculations is minuscule compared to the massive delay of waiting for data to be retrieved from slow HBM. This proves that the raw computational load isn't the whole story; for modern AI workloads, avoiding slow memory access is far more important for improving wall-clock speed.

2. The Real Bottleneck: It's Not Compute, It's Memory I/O

The key to FlashAttention's success lies in being "IO-aware." To understand this, one must consider the GPU memory hierarchy: there is a small amount of extremely fast on-chip SRAM (with a bandwidth of \sim 19 TB/s) and a large amount of relatively slow High Bandwidth Memory, or HBM (with a bandwidth of \sim 1.5 TB/s). The SRAM is over an order of magnitude faster, but also orders of magnitude smaller. As the authors state, the missing principle in prior work was accounting for this hardware reality.

We argue that a missing principle is making attention algorithms IO-aware—accounting for reads and writes between levels of GPU memory.

FlashAttention's "tiling" method is designed specifically for this hierarchy. It breaks the large attention computation into smaller blocks that are sized to fit entirely within the GPU's fast SRAM. By processing these blocks one at a time, it performs all the intermediate steps within the fast memory, avoiding the need to read and write the huge $N \times N$ matrix to and from the slow HBM. This simple change massively reduces the number of HBM accesses required, cutting the data read and written from 40.3 GB in the standard implementation to just 4.4 GB. Essentially, tiling turns a single, massive memory operation that spills over into slow HBM into a series of smaller, self-contained operations that each execute entirely within the ultra-fast SRAM, dramatically reducing the round-trips to main memory.

3. A Free Lunch: Speed Without the Quality Trade-off

Most attempts to optimize the attention mechanism fall into the category of "approximate attention." These methods achieve speedups by changing the underlying math, often by sparsifying or using low-rank approximations of the attention matrix. While this can reduce computational complexity, it almost always comes at the cost of model quality and accuracy.

FlashAttention is different. It is an *exact* attention algorithm. Because it doesn't change the core computation only how that computation is executed on the hardware

it produces a mathematically identical output to the standard implementation. This is a critical distinction because it allows researchers to adopt FlashAttention as a drop-in replacement to accelerate existing models without introducing approximation errors, which would otherwise complicate model evaluation and comparison.

This is what makes the algorithm so impactful. It provides a massive boost in speed and memory efficiency without any downside in terms of accuracy or model quality. It is a rare "free lunch" in machine learning, where performance is improved without any trade-offs.

4. Unlocking New Frontiers: From Faster Training to New AI Capabilities

The practical impact of this I/O-aware approach is not just incremental. By fundamentally improving efficiency, FlashAttention unlocks new capabilities that were previously out of reach. The source document highlights several concrete achievements:

- **Faster Training:** It delivered a **15% end-to-end wall-clock speedup** on BERT-large training compared to the MLPerf 1.1 record, a **3x speedup** on training GPT-2, and a **2.4x speedup** on the Long-Range Arena benchmark.
- **Higher Quality Models:** By enabling the use of longer context, models become more powerful. A GPT-2 model trained with FlashAttention achieved a **0.7 improvement in perplexity**, and models for long-document classification saw a **6.4 point lift** in performance.
- **Solving Unsolvable Problems:** The most powerful result was on the challenging Path-X (16K sequence length) and Path-256 (64K sequence length) benchmarks. Before FlashAttention, all Transformer models had failed on these tasks, achieving only random-chance performance. FlashAttention enabled the **first-ever Transformer model to achieve better-than-chance performance** on the Path-X benchmark, and its block-sparse variant achieved the same breakthrough on the even more challenging Path-256 benchmark. These results show that optimizing how algorithms interact with hardware is not just about saving time or money; it can be the key to allowing models to tackle problems that were previously intractable.

Conclusion: Looking at Old Problems in a New Light

The core lesson from FlashAttention is that in an era where GPU compute speed has outpaced memory speed, optimizing for memory I/O—not just raw computation—is a critical and often overlooked principle for building efficient AI systems. By applying

two well-established computing techniques tiling and recomputation the researchers were able to remove a fundamental bottleneck that had plagued an entire class of AI models.

FlashAttention solved a major AI bottleneck by reconsidering a fundamental assumption. What other performance walls could we break through by questioning the metrics we've been taught to focus on?

FlashAttention Study Guide

1. What is the primary performance bottleneck of standard self-attention that FlashAttention is designed to address?
 2. Explain the concept of an "IO-aware" algorithm and its relevance to modern GPU performance.
 3. What are the two principal techniques used by FlashAttention to reduce memory accesses and avoid storing the intermediate attention matrix?
 4. How does FlashAttention's memory requirement scale with sequence length compared to the standard attention implementation?
 5. Describe the roles of GPU High Bandwidth Memory (HBM) and on-chip SRAM within the FlashAttention algorithm's execution.
 6. FlashAttention's backward pass involves more FLOPs than standard attention due to recomputation. Why does it still achieve a faster wall-clock time?
 7. What is kernel fusion, and how is this optimization technique utilized in the implementation of FlashAttention?
 8. How does block-sparse FlashAttention differ from the standard FlashAttention algorithm, and what is its main advantage?
 9. What new capabilities for Transformer models were demonstrated on the Path-X and Path-256 benchmarks by using FlashAttention?
 10. State the IO complexity for both standard attention and FlashAttention, as defined in the paper.
-

Answer Key

1. The primary bottleneck is that standard attention's time and memory complexity are quadratic in the sequence length (N). This is because it must materialize and write the large $N \times N$ attention matrix to and from the relatively slow GPU High Bandwidth Memory (HBM), making it slow and memory-hungry for long sequences.
2. An IO-aware algorithm is one that carefully accounts for reads and writes between different levels of memory, such as fast on-chip SRAM and slower HBM. This is crucial for modern GPUs because compute speed has outpaced memory speed, meaning many operations are bottlenecked by memory access (IO) rather than computation.
3. FlashAttention uses tiling and recomputation. Tiling involves splitting the input matrices into blocks, loading them into fast SRAM, and incrementally performing the softmax reduction. Recomputation is used in the backward pass to reconstruct the attention matrix on-chip using statistics saved from the forward pass, thereby avoiding the need to store the large intermediate $N \times N$ matrix.
4. FlashAttention's memory usage scales linearly with sequence length, requiring only $O(N)$ additional memory. In contrast, standard attention implementation has a memory footprint that scales quadratically, $O(N^2)$, because it must store the intermediate attention matrix.
5. In FlashAttention, the large input matrices (Q , K , V) and final output matrix (O) reside in the slower HBM. The algorithm iteratively loads small blocks of these matrices into the much faster on-chip SRAM, performs the core attention computations entirely within SRAM, and then writes the updated output block back to HBM.
6. Although recomputation in the backward pass increases the number of floating-point operations (FLOPs), it is faster because it avoids the standard approach of reading the massive intermediate attention matrix from slow HBM. The time saved from the massively reduced HBM access far outweighs the time spent on the extra on-chip computation.
7. Kernel fusion is a technique that combines multiple operations into a single GPU kernel to minimize memory traffic. FlashAttention fuses all attention operations—matrix multiplication, softmax, optional masking and dropout—into one kernel, which prevents the intermediate matrices from being repeatedly written to and read from HBM.
8. Block-sparse FlashAttention extends the algorithm by only computing over a predefined block-sparse mask of the attention matrix, skipping the zero blocks. Its main advantage is that it is even faster and has a better IO complexity than dense FlashAttention, with the speedup being proportional to the sparsity ratio.
9. FlashAttention enabled the first Transformer models to achieve better-than-chance performance on these challenging long-context tasks. Specifically, a Transformer with FlashAttention achieved 61.4% accuracy on Path-X (sequence length 16K), and a block-sparse FlashAttention model achieved 63.1% accuracy on Path-256 (sequence length 64K).

10. For a sequence length N , head dimension d , and SRAM size M , the IO complexity of standard attention is $\Theta(Nd + N^2)$. The IO complexity of FlashAttention is $\Theta(N^2d^2M^{-1})$, which represents a significant reduction in HBM accesses for typical hardware values.
-

Essay Questions

1. Discuss the principle of "IO-awareness" as it relates to modern GPU architecture and performance bottlenecks. Using the comparison between standard attention and FlashAttention, explain why focusing on FLOP reduction alone can be an insufficient or even misleading metric for wall-clock speedup.
 2. Elaborate on the implementation of tiling and recomputation within the FlashAttention algorithm. How do these two techniques synergize to overcome the primary challenges of (i) computing the softmax reduction without access to the entire input and (ii) executing the backward pass without storing the $O(N^2)$ intermediate attention matrix?
 3. Analyze the performance trade-offs inherent in FlashAttention's design. Compare its efficiency against both standard attention and various approximate attention methods (e.g., Linformer, Performer) across the dimensions of FLOPs, HBM accesses, memory footprint, and wall-clock runtime.
 4. The paper presents block-sparse FlashAttention as a useful extension. Describe the motivation for creating a sparse version, the mechanism by which block-sparsity is implemented in the algorithm, and how its IO complexity and empirical runtime scale relative to the dense FlashAttention implementation.
 5. Evaluate the empirical impact of FlashAttention on Transformer model training and capabilities, as presented in the paper's experiments. Discuss its contributions in three key areas: accelerating training time for models like BERT and GPT-2, improving model quality by enabling longer context lengths, and solving previously intractable long-sequence benchmarks.
-

Glossary of Key Terms

Term	Definition
Arithmetic Intensity	The ratio of arithmetic operations to bytes of memory access. It is used to classify operations as compute-bound or memory-bound.
Block-sparse Attention	An approximate attention method where computation is restricted to a predefined block-sparse pattern within the attention matrix, skipping computations for blocks designated as zero.
Compute-bound	A characteristic of an operation where the total time taken is determined primarily by the number of arithmetic operations, with time for memory access being significantly smaller.
FlashAttention	An IO-aware, exact attention algorithm that uses tiling and recomputation to reduce the number of memory read/write operations between GPU HBM and SRAM, resulting in faster execution and linear memory usage.

HBM (High Bandwidth Memory)	The main, large-capacity GPU memory (e.g., 40-80GB on an A100 GPU). It has high bandwidth (1.5-2.0 TB/s) but is significantly slower than on-chip SRAM.
IO-aware	A quality of an algorithm that is designed to account for and minimize the cost of reads and writes (Input/Output) between different levels of a memory hierarchy, such as fast SRAM and slower HBM.
IO Complexity	A measure of an algorithm's performance based on the number of data transfers (HBM accesses) it requires. FlashAttention's analysis focuses on minimizing this complexity.
Kernel Fusion	An optimization technique where multiple sequential operations are merged into a single GPU kernel. This reduces overhead by loading input from HBM only once and avoiding writing intermediate results back to HBM.
Memory-bound	A characteristic of an operation where the total time taken is determined primarily by memory access speed, with time spent on computation being significantly smaller. Many Transformer operations, like softmax and layer norm, are memory-bound.
Recomputation	A technique used in FlashAttention's backward pass. Instead of storing the large intermediate attention matrix, it is recomputed on-the-fly in fast SRAM using the original inputs and normalization statistics saved from the forward pass.
SRAM (Static Random-Access Memory)	Extremely fast, on-chip GPU memory. It is an order of magnitude faster than HBM but is also orders of magnitude smaller in size (e.g., 192KB per streaming multiprocessor on an A100 GPU).
Standard Attention	The conventional implementation of self-attention, which materializes the full $N \times N$ attention matrix in HBM, leading to quadratic time and memory complexity.
Tiling	A technique of splitting large inputs into smaller blocks or "tiles." In FlashAttention, blocks of Q, K, and V are loaded from HBM into SRAM to compute attention incrementally, avoiding the materialization of the entire $N \times N$ matrix.