

Cylon: Asynchronous Linear Attention

Alexander Waitz¹, William Hu¹, Benjamin Spector¹, Atri Rudra²,
Christopher Ré¹, Simran Arora³

¹Stanford University ²University at Buffalo, SUNY ³Caltech
waitz@cs.stanford.edu

Abstract

Sequence models face stark tradeoffs between recall quality and memory efficiency. The ability to use information over long sequences (i.e. recall) is critical for sequence modeling tasks ranging from information extraction to reasoning. Prior work has shown that in theory, *linear* attention models with sufficient recurrent state sizes can expand the Pareto frontier of the recall-memory tradeoff space beyond alternative architectures such as softmax attention and state space models. However, it is difficult to scale the linear attention state size due to hardware bottlenecks. I/O-aware algorithms store the linear attention states in thread registers, however state sizes beyond ≈ 3 megabytes exhaust register memory and trigger expensive register spills. In this work, we introduce CYLON, a hardware-aware modification to linear attention architectures which partitions the recurrent state across the registers of multiple GPU processors and asynchronously combines the partitions. When applying CYLON to popular architectures, such as Hedgehog, Mamba-2, and Gated Deltanet, we unlock up to $6\times$ higher forward pass throughput and $10\times$ higher backward pass throughput compared to prior linear attention algorithms on Hopper and Blackwell GPUs. Moreover, CYLON preserves recall quality compared to non-partitioned baselines. Finally, CYLON makes large states available to model designers by unlocking sizes (≥ 131 MB) which are not achievable by the existing linear attention kernels.

1 Introduction

Sequence models face a fundamental tradeoff between recall quality—the ability to manage information over long sequences—and memory efficiency (Cho et al., 2014; Arora et al., 2024). While there has been significant exploration into architectures that require complex initializations and inductive biases over the past few years (von Oswald et al., 2025; Poli et al., 2023; Gu & Dao, 2023), the bitter lesson is that a model’s memory consumption is the primary factor governing its quality (Arora et al., 2023). Prior work shows that using linear attention (Katharopoulos et al., 2020) with large recurrent state sizes expands the recall-memory frontier beyond prior architectures such as state space models (Gu et al., 2021) and softmax attention (Vaswani et al., 2017).

Applications land at different points on the tradeoff space: for example, low-latency workloads often want to prioritize extreme efficiency over recall (Merrill et al., 2026; Kimi et al., 2025; Qwen, 2025; Cartesia, 2025; Gu et al., 2025; Liquid, 2025). Compressed and sparse attention have emerged as widely adopted methods in open-source models for navigating this tradeoff space (GLM-5-Team et al., 2026; DeepSeek-AI et al., 2026). By tuning the compression and sparsity factors, practitioners can flexibly trade off memory and recall (DeepSeek-AI et al., 2024; Lai et al., 2026; Yuan et al., 2025; Child et al., 2019).

Linear attention has also become a popular architectural choice for open-source frontier models (Kimi et al., 2026; Qwen Team, 2026). However, even as increasingly large models adopt linear attention, the state sizes used by these models remain relatively small (2MB for Kimi Linear and Qwen3-Next, 2.6MB for Mamba2-2.7B), locking models to the extreme-efficiency end of the spectrum. No prior work has studied how to use linear attention

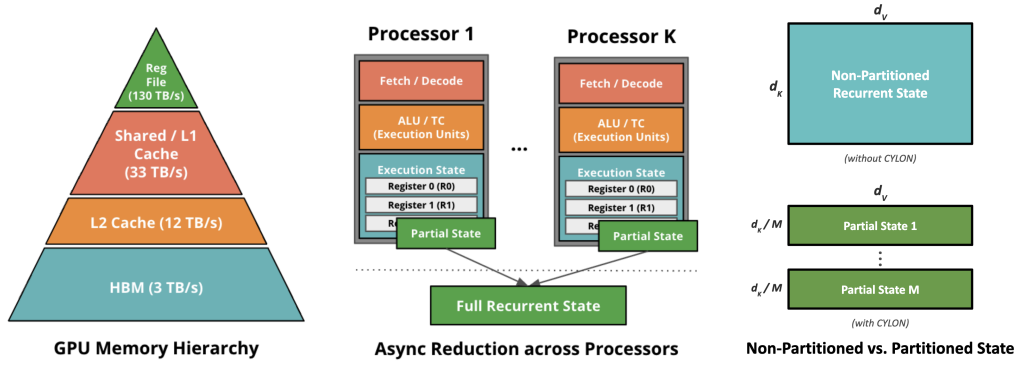


Figure 1: Despite substantial effort into architecture design, the bitter lesson is that a model’s memory consumption is the primary factor governing its quality (Arora et al., 2023). Linear attention expands the frontier between quality and efficiency beyond prior architectures (SSMs, attention). **(Left)** Hardware memory limits prevent foundation model designers from flexibly scaling the state size, however. **(Middle)** CYLON addresses the bottleneck by partitioning the state across the registers of many processors and asynchronously combining them using modern GPU features. **(Right)** Algorithmically, CYLON is equivalent to expanding recurrent state by increasing the size of the feature dimension.

with larger state sizes or how quality scales with state size. We ask: *why is linear attention constrained to small state sizes and can we modify existing algorithms to use larger state sizes with improved efficiency?*

To understand why today’s linear attention models use small state sizes, we start by profiling a suite of popular linear attention kernels (Yang & Zhang, 2024). Linear attention replaces softmax with a simple dot product over query and key vectors. While softmax attention scales quadratically with sequence length, linear attention admits a sub-quadratic *chunkwise parallel* algorithm (Hua et al., 2022): sequences are divided into chunks (e.g. of 64 tokens), and the output is computed using recurrent computations across chunks and parallel computations within chunks. The fastest kernels persistently store the recurrent state in thread registers, the fastest segment of GPU memory (Spector et al., 2025). However, register files are limited to 256 KB per processor (NVIDIA, 2022), so states beyond a few megabytes tend to trigger register spills resulting in $\approx 4\times$ slower memory accesses. Our analysis reveals that the key challenge is not raw compute but the cost of moving and aggregating state across the GPU memory hierarchy.

We observe that current architectures miss a fundamental trend in AI accelerators that can help address this limit: accelerators are increasingly asynchronous machines. Our insight is that the register bottleneck can be addressed by partitioning large linear attention states across the fast memory of *multiple thread blocks*, and then asynchronously combining the partial results in the GPU’s global memory. Unfortunately, old-school atomic operations are bottlenecked by address generation, use of the generic (vs. async) proxy, and poor coalescing, which hurts memory throughput (Table 2). Excitingly, modern GPUs are prioritizing explicit hardware for bulk, pipelined asynchronous instructions such as Warp-group matrix multiply (WGMMMA) and Tensor Memory Acceleration (TMA) on Hopper and Blackwell.

Inspired by the asynchrony of modern GPUs, we propose a drop-in modification to linear attention architectures called CYLON, designed to unlock new operating points in the recall-memory tradeoff space. We partition the recurrent state across m streaming multiprocessors (SM) and each SM maintains its partition in registers, processing the sequence independently with the chunkwise algorithm. A global asynchronous reduction is used to combine the partitioned outputs. CYLON is designed to maximize locality while enabling large effective state sizes, by using three classes of asynchronous operations:

- **Asynchronous multiplication:** Tensor cores execute matmuls using inputs in shared (or tensor) memory, reducing register pressure and allowing larger chunks per block.

- **Asynchronous I/O:** TMA streams each SM’s partition between HBM and shared memory, with warp specialization to overlap data movement and computation (Bauer et al., 2011).
- **Asynchronous reductions:** Global reduction instructions (e.g., sum) aggregate the outputs of different SMs without explicit synchronization, avoiding HBM round-trips.

Overall, we demonstrate across three linear attention backbones (Hedgehog, Mamba-2, Gated Deltanet) that CYLON achieves up to $6\times$ greater forward pass throughput and $10\times$ greater backward pass throughput compared to widely used linear attention kernels on Hopper and Blackwell GPUs. Moreover, these models match the quality of non-CYLON linear attention on a suite of downstream recall-intensive and natural language understanding tasks, thereby expanding the Pareto frontier between quality and efficiency (Figure 1). In summary, our contributions include: (1) analysis of the bottlenecks of linear attention, (2) the CYLON partitioning strategy, and (3) an I/O-aware implementation of CYLON.

As AI moves towards a heterogenous and multi-silicon hardware landscape (NVIDIA, 2026b; 2025; AMD, 2026; Anthropic, 2026; OpenAI, 2026; Etched, 2026; d Matrix, 2025), we hope CYLON encourages further exploration of silicon-native model architectures.

2 Background

We provide background on GPU hardware in Section 2.1, background on efficient ML architectures in Section 2.2, and a profiling analysis of existing linear attention kernels in Section 2.3.

2.1 GPU Fundamentals.

ML workloads are executed on GPUs as *kernels*, which are programs run by thousands of threads in parallel. In NVIDIA’s terminology, GPUs consist of *streaming multiprocessors* (SMs), each responsible for running the *thread blocks* that make up a kernel. At the hardware level, blocks are decomposed into *warps* – groups of 32 threads that execute in lockstep. A typical kernel loads data, performs computation, and writes results.

Memory hierarchy. GPUs are designed with a layered memory hierarchy (Figure 1), each level trading off capacity for speed; (1) *Global memory (HBM)*: Large but relatively slow, shared across all SMs (e.g., 80 GB, 3 TB/s bandwidth on NVIDIA H100), (2) *Shared memory (SRAM)*: On-chip, fast, and scoped to a thread block within an SM (e.g., 256 KB, 33 TB/s). (3) *Registers*: Fast and small memory, private to individual threads (e.g., 65K registers per SM, 130 TB/s). Efficient kernel design carefully stages data through this hierarchy to minimize costly HBM accesses.

Execution units. GPUs expose a diverse set of hardware execution units: (1) *Tensor Cores* are specialized units for dense matrix multiplications, delivering extraordinary throughput (approximately 2 PFLOP/s on B200). (2) *Arithmetic Logic Units and Special Function Units* are general-purpose compute units for scalar, vector, and transcendental operations, much slower in comparison (approximately 80 TFLOP/s on B200). (3) Processors have load/store units handle I/O. The performance gap between Tensor Cores and general ALU/SFU pipelines has widened across GPU generations, making tensor cores the critical path for accelerating modern ML workloads. Different warps can *occupy* different execution units, which helps avoid over-utilization of any one unit and improves throughput.

Hardware asynchrony Beyond occupancy, recent GPU generations introduce dedicated support for bulk, pipelined asynchronous execution. An asynchronous instruction can run in the background while threads perform other instructions. For example, Hopper and Blackwell architectures expose the Tensor Memory Accelerator (TMA) dedicated hardware for asynchrony:

- **TMA I/O:** TMA issues bulk data transfers between HBM and SRAM asynchronously. TMA performs address calculations, freeing registers and enabling producer-consumer specialization.
- **TMA reductions:** TMA can perform global reductions (e.g., sum, max, min) while SMs store to HBM, allowing reduction latency to be hidden behind ongoing computation.

- **Asynchronous MMA:** Instructions that allow Tensor Cores to consume operands directly from shared memory or tensor memory rather than registers. In practice, async MMAs are designed to work in tandem with TMA: TMA streams large tiles into SRAM asynchronously, and the tensor core consumes them efficiently, reducing register pressure and enabling larger tile sizes.

2.2 Efficient Architectures Tradeoff State Size and Quality

Attention Vaswani et al. (2017) define queries, keys, and values $q, k, v \in \mathbb{R}^{N \times d}$ from input $u \in \mathbb{R}^{N \times d}$ via weight matrices $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$. The attention mechanism outputs

$$a_{n,i} = \frac{\exp(q_n^\top k_i / \sqrt{d})}{\sum_{j=1}^n \exp(q_n^\top k_j / \sqrt{d})}, \quad y_n = \sum_{i=1}^n a_{n,i} v_i, \quad (1)$$

where N is the sequence length and d is the head dimension. Since each query q_n must interact with all prior keys k_i , the overall runtime is quadratic in sequence length, $\mathcal{O}(N^2 d)$, while the memory cost during inference per token generation is $\mathcal{O}(Nd)$ due to the KV-cache.

Linear attention. Katharopoulos et al. (2020) replace the softmax kernel with a feature map $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^D$, yielding:

$$\hat{a}_{n,i} = \frac{\phi(q_n)^\top \phi(k_i)}{\sum_{j=1}^n \phi(q_n)^\top \phi(k_j)}, \quad \hat{y}_n = \sum_{i=1}^n \hat{a}_{n,i} v_i. \quad (2)$$

Unlike softmax attention, the key-value terms can be pre-aggregated:

$$\hat{y}_n = \frac{\phi(q_n)^\top \left(\sum_{i=1}^n \phi(k_i) v_i^\top \right)}{\phi(q_n)^\top \sum_{i=1}^n \phi(k_i)}. \quad (3)$$

This leads to a recurrent update rule:

$$\hat{y}_n = \frac{\phi(q_n)^\top s_n}{\phi(q_n)^\top z_n}, \quad s_n = s_{n-1} + \phi(k_n) v_n^\top, \quad z_n = z_{n-1} + \phi(k_n), \quad (4)$$

where $s_n \in \mathbb{R}^{D \times d}$ is the recurrent state and $z_n \in \mathbb{R}^D$ is a normalization accumulator. Recent work replaces z_n with layer or RMS normalization (Sun et al., 2023; Yang et al., 2023). Linear attention runs in $\mathcal{O}(NdD)$ time during training and $\mathcal{O}(Dd)$ time per recurrent step during inference, with constant memory $\mathcal{O}(Dd)$ independent of sequence length.

State size and recall. Recent linear attention work focuses on how to better compress input sequences into the fixed-size recurrent state (e.g., Mamba-2, Deltanet, RetNet (Dao & Gu, 2024; Yang et al., 2025b; Sun et al., 2023)). Increasing state size improves quality by allowing the model to retain more information in its working memory during inference, but also incurs higher compute and memory costs, reducing efficiency. State size is especially important in *recall tasks*, or tasks that require a model to manage and use long-term information, a capability that underpins reasoning, code generation, and dialogue. However, the behavior of linear attention models on memory-intensive tasks across state size scales has not yet been studied.

2.3 Profiling linear attention

We profile representative linear attention kernels using the widely used Flash Linear Attention and Mamba-2 repositories on NVIDIA H100 GPUs (Yang & Zhang, 2024; Dao & Gu, 2024). We tune the Triton kernels and collect profiles using NVIDIA Nsight Compute (NCU). Experiments are run at sequence length 4096, batch size 8, and model dimension 2048 across different state sizes. We measure tensor core utilization, HBM throughput, and long scoreboard stalls (which correlate with stalls on HBM/L2 or register spills). Representative results are reported in Table 1 for models with 32 heads (small state size) and 1 head (large state size).

Linear attention admits three distinct factorizations, each with different hardware tradeoffs:

1. **Parallel:** Parallelizes over sequence, multiplying $\mathbf{QK}^\top$ first (as in softmax attention, Eq. 1). Cost scales quadratically, $\mathcal{O}(N^2d)$, making this form prohibitive for long sequences.
2. **Recurrent:** Pre-aggregates keys and values (Eq. 4) into a recurrent update rule. This minimizes FLOPs, $\mathcal{O}(NdD)$, but does not use tensor cores, limiting throughput.
3. **Chunkwise:** Interpolates between parallel and recurrent. Within chunks of size C , the parallel form uses tensor cores; across $\frac{N}{C}$ chunks, the recurrent form avoids quadratic compute.

Table 1 highlights these trade-offs: the parallel kernel wastes compute on quadratic recomputation; the recurrent kernel avoids this but forgoes tensor cores; and the chunkwise kernel is most efficient until large state sizes cause register spilling to L1 (~ 40 cycles, $\sim 4\times$ slower).

Impl.	Heads	Duration (ms)	Utilizations	HBM	
			Tensor core (%)	GB/s	Stalls (Cycles)
Parallel	32	2.5	36.2	157	1.5
Parallel	1	13.8	35.0	129	1.5
Recurrent	32	6.0	0.0	88	8.2
Recurrent	1	87.5	0.0	57	5.5
Chunkwise	32	0.6	13.3	1400	5.9
Chunkwise	1	4.8	24.7	1620	7.1

Table 1: NCU profiles for linear attention kernels that use different factorizations.

3 CYLON: Asynchronous linear attention

Our objective is to enable large state sizes in linear attention while retaining hardware efficiency. Existing kernels struggle because recurrent states exceeding a few MBs cannot fit in registers, leading to register spills, shared memory contention, and global synchronization overhead (Section 2.3). CYLON addresses this by partitioning the recurrent state across multiple SMs, ensuring each partition remains fully register-resident and then combining them asynchronously.

3.1 Methodology

Linear attention can be expressed using a single feature map function ϕ , which transforms queries and keys before aggregation:

$$\mathbf{y}_n = \phi(\mathbf{q}_n)^\top \left(\sum_{i=1}^n \phi(\mathbf{k}_i) \mathbf{v}_i^\top \right). \quad (5)$$

CYLON generalizes this formulation by *partitioning* the computation across m streaming multiprocessors (SMs). Each SM is assigned a distinct feature map ϕ_k , producing independent partial states that are later aggregated:

$$\mathbf{y}_n = \sum_{k=1}^m \phi_k(\mathbf{q}_n)^\top \left(\sum_{i=1}^n \phi_k(\mathbf{k}_i) \mathbf{v}_i^\top \right). \quad (6)$$

Following modern linear attention architectures, we replace the normalization term from Equation 4 with an RMS normalization after the linear attention operator.

The linear attention state size is computed as $\frac{d^2}{H}$ for model dimension d and H heads, where modulating H changes the state size. For CYLON, the state size is computed as $m(\frac{d^2}{H})$, where modulating H or m changes the state size.

Feature map functions. Standard linear attention employs a *single* feature map $\phi(\cdot)$. In contrast, CYLON reuses the same $\mathbf{q}, \mathbf{k}, \mathbf{v}$ across SMs, but applies *distinct* $\phi_k(\cdot)$ functions.

Simply duplicating the same map across SMs would yield redundant work; diversity among ϕ_k ensures that each SM contributes unique information. For quality, we implement ϕ_k as a learned MLP following Zhang et al. (2024). In practice, this reduces to a single linear projection followed by a softmax over the feature dimension:

$$\phi_k(x) = \frac{\exp(A_k x + b_k)}{\sum_i \exp((A_k x + b_k)_i)}, \quad (7)$$

where A_k and b_k are learned weights.¹

3.2 Analysis

This section connects CYLON to existing linear attention architectures (Katharopoulos et al., 2020). A natural question is: *does splitting the recurrent state into disjoint partitions reduce the expressivity of linear attention?* We next investigate whether the partitioned representation, represented using feature maps $\phi_1, \dots, \phi_m$, can recover the same computations as a single unified feature map, which we denote by Φ .

Theorem 1 (Unbounded Equivalence). *Let d, m, D be positive integers. Given a unified feature map $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{mD}$, there exists a collection of smaller maps $\phi_1, \dots, \phi_m : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that:*

$$\frac{\Phi(q_i)^\top \sum_{j \leq i} (\Phi(k_j) v_j^\top)}{\Phi(q_i)^\top \sum_{j \leq i} \Phi(k_j)} = \frac{\sum_{k=1}^m \phi_k(q_i)^\top \sum_{j \leq i} (\phi_k(k_j) v_j^\top)}{\sum_{k=1}^m \phi_k(q_i)^\top \sum_{j \leq i} \phi_k(k_j)}. \quad (8)$$

A single feature map can be decomposed into disjoint partitions without loss of expressivity.

Theorem 2 (Elementwise Nonlinear Maps). *Suppose $\Phi(x) = \sigma(Ax + b)$ with $A \in \mathbb{R}^{Dm \times d}$, $b \in \mathbb{R}^{Dm}$, and σ an elementwise activation function. Let $C(\cdot)$ denote the number of arithmetic operations required by a function. Then there exist $\phi_1, \dots, \phi_m$ satisfying Equation 8 such that $C(\Phi) = \sum_{k=1}^m C(\phi_k)$. In other words, the decomposition introduces no additional computational cost across all feature maps compared to the single, unified map.*

Together, these results establish that partitioning the state across SMs preserves the expressivity of linear attention. It directly connects to modern linear attention architectures such as Mamba-2 and Gated Deltanet which employ elementwise feature maps (SiLU)². To demonstrate that CYLON can match the representational capacity of non-partitioned models, we later present empirical results showing that CYLON models achieve comparable perplexity to baselines despite using distributed state (Section 4).

3.3 Hardware-Aware Algorithms

CYLON improves the hardware-efficiency of linear attention by combining three systems principles: (1) *asynchronous execution*, (2) *work partitioning across SMs*, and (3) *register-locality of state*. Our design takes advantage of features such as Tensor Memory Accelerator (TMA) instructions and tensor cores (TC).

CYLON kernel. Given Equation 6, a straightforward implementation requires SMs to repeatedly communicate through high-bandwidth memory (HBM). Each SM must load the output tensor from HBM, add its local result, and write out the new tensor to HBM for the next SM. This introduces long-latency roundtrips and inter-SM synchronization overhead, producing up to a $16\times$ slowdown compared to our final implementation.

Instead, CYLON partitions the recurrent state across SMs, keeps each partition fully resident in registers, and uses asynchronous reductions to combine results. Pseudocode for the core

¹In linear attention variants, such as Mamba-2 and Gated Deltanet, explicit feature maps conflict with the existing initialization (Dao & Gu, 2024). For these settings, we instead discard the learned component in $\phi(\cdot)$ and assign distinct projections of q and k across SMs, which remains efficient due to Mamba-2’s Multi Value Attention design (see Appendix B.4).

²Nevertheless, some architectures employ design decisions which do not cleanly partition in practice, such as Hedgehog’s softmax activation and Gated Deltanet’s state update rule.

kernel is provided in Algorithm 2, where $\mathbf{local}_{KV}$ denotes the cumulative register-resident state maintained by each SM. We also include the reference chunk-wise algorithm that is used in prior linear attention kernels for reference (Hua et al., 2022; Yang et al., 2023; Arora et al., 2024). The key operation is the reduction step, which aggregates results across SMs.

Algorithm 1 Chunk-wise (Hua et al., 2022)

Require: Queries, keys, values $Q, K, V \in \mathbb{R}^{N \times d}$ and feature map ϕ

- 1: Init. recurrent states $\mathbf{KV} \leftarrow 0$ in registers
- 2: **for** $t = 0, \dots, N/64$ **do**
- 3: Load tile Q_t, K_t, V_t
- 4: $X_t \leftarrow \phi(Q_t)\phi(K_t)^\top$
- 5: Apply causal mask to X_t
- 6: $O_t \leftarrow X_t V_t + \phi(Q_t)\mathbf{KV}$
- 7: $\mathbf{KV} \leftarrow \mathbf{KV} + \phi(K_t)^\top V_t$

Algorithm 2 CYLON async chunk-wise

Require: Queries, keys, values $Q, K, V \in \mathbb{R}^{N \times d}$ and feature maps $\phi_1, \dots, \phi_m$

- 1: Init. local states $\mathbf{local}_{KV} \leftarrow 0$ in registers
- 2: **for** $t = 0, \dots, N/64$ **do**
- 3: Load tile Q_t, K_t, V_t
- 4: $X_t \leftarrow \phi_k(Q_t)\phi_k(K_t)^\top$
- 5: Apply causal mask to X_t
- 6: $O_t \leftarrow X_t V_t + \phi_k(Q_t)\mathbf{local}_{KV}$
- 7: **Async. Reduce** O_t across SMs
- 8: $\mathbf{local}_{KV} \leftarrow \mathbf{local}_{KV} + \phi_k(K_t)^\top V_t$

Reduction Mechanisms. A challenge in CYLON is efficiently aggregating partitioned states across SMs without becoming bottlenecked by memory synchronization. While traditional atomic operations (NVIDIA, 2020; AMD, 2025a) enable concurrent updates through relaxed memory semantics, they suffer from address calculation overheads and poor memory coalescing, while also making it difficult to implement fine-grained pipelining (e.g., they use the generic, not async, proxy).

Modern GPUs expose explicit asynchronous reduction hardware through Tensor Memory Accelerator (TMA) instructions, which we leverage in CYLON. Unlike atomic operations that require address generation and use generic memory proxies, TMA reductions operate as bulk, pipelined operations – warps issue reductions (add, max, min) that execute asynchronously in dedicated hardware units while computation proceeds uninterrupted. Table 2 quantifies this advantage: TMA reductions achieve 8-11% higher throughput and 12-15% lower latency compared to atomic operations. The performance gap widens with larger feature maps (4 vs. 16). Compared to fully synchronous reductions, TMA achieves over $10\times$ higher throughput and $2\times$ lower latency.

Mechanism	Feat. Maps	Latency (ms/iter)	Throughput (GB/s)	Notes
Sync reductions	4	5.3	82.6	Barrier-synchronized global memory access
	8	15.8	28.2	
Atomic operations	4	2.5	856.9	Serialized global memory access
	16	9.3	845.4	
Async reductions	4	2.2	932.3	Overlaps reduction with compute
	16	7.9	940.9	

Table 2: Microbenchmark comparison of reduction mechanisms for 4 and 16 feature maps.

Portability. CYLON is a partitioning strategy that is portable across linear attention architectures, GPU generations, and kernel implementations. Linear attention variants such as Mamba-2 and Gated Deltanet can be CYLON-partitioned to more efficiently support large state sizes. In our experiments, we compare CYLON-partitioned models to their non-partitioned counterparts. Notably, CYLON is not limited to bespoke kernels: we apply it directly to widely used open-source implementations – Flash Linear Attention (Yang & Zhang, 2024) (Triton) and FlashQLA (Zhang et al., 2026) (TileLang) – and demonstrate speedups across these CUDA, Triton, and TileLang backends. Finally, we show in Table 2 that atomic addition can be used as a lower performance alternative to TMA reductions, enabling backwards compatibility with pre-Hopper GPUs.

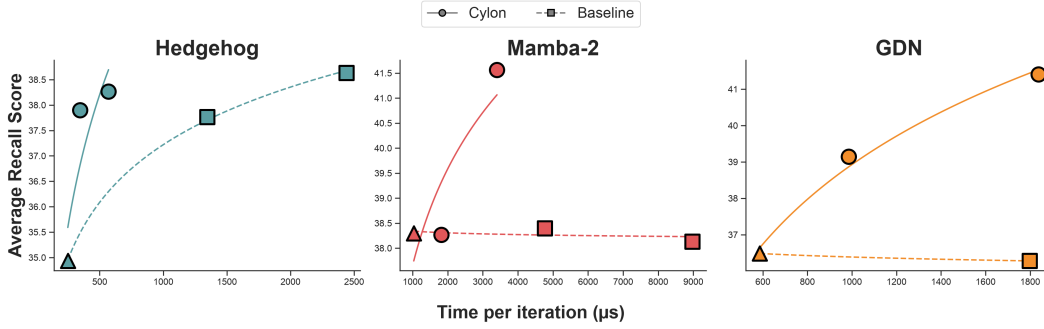


Figure 2: **Quality-speed tradeoff.** We demonstrate that CYLON represents an expansion of the quality-speed Pareto frontier when applied across linear attention architectures, as it achieves higher quality levels at greater speeds. In Tables 3 and 4, we present the model recall quality and speed results separately, as functions of recurrent state size.

4 Experiments

In this section, we present results to support the following claims:

1. **Language modeling overall.** We evaluate architectures in pretraining on the Pile dataset (Gao et al., 2020) and on standard language understanding (NLU) benchmarks. We find that CYLON-partitioned models match or exceed the quality of the non-partitioned (default) linear attention models across NLU tasks.
2. **Language modeling recall.** We show that CYLON-partitioned models match or exceed the recall quality of the baseline linear attention models across a range of state sizes (scaling state size correlates with recall quality).
3. **Forward and backward pass throughput.** Our CYLON kernels offers up to $6\times$ higher forward throughput and $10\times$ higher backward throughput compared to the popular SoTA linear attention kernels and unlocks larger state sizes ($\geq 131\text{MB}$) than prior kernels. These results hold on both Hopper and Blackwell GPUs.

Experimental setup. We evaluate models at the 1.3B scale using 50B tokens of training on the Pile dataset (Gao et al., 2020) (tokenized with the GPT-2 BPE tokenizer (Radford et al., 2019)). We first pretrain and evaluate Transformer++ (rotary encodings (Su et al., 2023)) and three representative variants of linear attention models: Hedgehog (Zhang et al., 2024), Mamba-2 (Dao & Gu, 2024), and Gated Deltanet (Yang et al., 2025a). We train each linear attention architecture across three state sizes. We then apply CYLON to Hedgehog, Mamba-2, and Gated Deltanet using $m \in \{1, 2, 4\}$ state partitions, and compare the CYLON-model to the baseline model that uses the unified state. We run evaluations on downstream tasks using the LM Eval Harness (Gao et al., 2023).

4.1 Language Modeling Evaluations

We measure the perplexity of each pretrained model on a heldout test set of 10M tokens and provide the results in Table 3. We also evaluate these models on a suite of natural language understanding (NLU) tasks that are commonly used in architecture evaluations (ARC easy/challenge, PIQA, WinoGrande, HellaSwag). Quality on these tasks is reported in Table 3 (“LM-Evals”). We find that larger state sizes tend to result in lower perplexity scores, and that the CYLON models compete with the perplexity scores of the unified state models at each state size. We also observe that all model variants perform similarly on the NLU task suite. This is expected since the NLU tasks are relatively short context and thus do not implicate state size.

4.2 Recall Evaluations

We evaluate our pre-trained models on a suite of downstream recall-intensive tasks proposed in prior work (Arora et al., 2024), including SWDE and FDA (Wu et al., 2021; Deng et al.,

Architecture	Method	Params/ Tokens	State Sz. MB/layer	LM Pile Ppl. ↓	Recall Avg. Acc. ↑	NLU Avg. Acc. ↑
Transformer	—	1.3B/50B	33.5	6.6	62.7	47.6
Hedgehog	Non-partitioned	1.2B/50B	1.0	6.9	34.9	47.5
	CYLON ($m=2$)	1.2B/50B	2.0	6.9	37.9	47.4
	Non-partitioned	1.2B/50B	2.0	6.9	37.8	47.4
	CYLON ($m=4$)	1.2B/50B	4.0	6.9	38.3	47.3
	Non-partitioned	1.2B/50B	4.0	6.8	38.6	46.7
Mamba-2	Non-partitioned	1.4B/50B	2.1	6.7	38.3	47.6
	CYLON ($m=2$)	1.4B/50B	4.2	6.7	38.2	47.3
	Non-partitioned	1.4B/50B	4.2	6.7	38.4	47.0
	CYLON ($m=4$)	1.4B/50B	8.4	6.7	41.6	48.6
	Non-partitioned	1.4B/50B	8.4	6.7	38.1	47.4
Gated Deltanet	Non-partitioned	1.2B/50B	1.0	6.9	36.5	46.6
	CYLON ($m=2$)	1.4B/50B	2.0	6.9	39.2	46.5
	Non-partitioned	1.4B/50B	2.0	6.8	36.3	47.0
	CYLON ($m=4$)	1.7B/50B	4.0	6.9	41.4	47.4
	Non-partitioned	1.7B/50B	4.0	—	—	—

Table 3: **Evaluation of pre-trained language models.** We train models of different state sizes (reported in MB per layer) using default linear attention and CYLON-partitioned linear attention. Within each architecture, we compare CYLON-partitioned models (with m partitions) against default (non-partitioned) linear attention at matched state sizes. We report overall language modeling (LM) perplexity on a heldout test set of The Pile, and downstream scores on both recall-intensive (SWDE, FDA, SQUAD) and NLU tasks (LM-Evals). For each pair of models (CYLON and non-partitioned) at the same state size, we **bold** the scores of the better performing model. Per-task scores are reported in Section B.1. Missing measurements indicate configurations not supported by the given algorithm’s kernel.

2022), two popular information extraction benchmarks, and SQUAD (Rajpurkar et al., 2018), a popular QA benchmark. Recall-intensive tasks require the model to remember information provided in the prompt context, which is a capability that benefits from using larger state sizes (Arora et al., 2024).

In Table 3, we observe that models with larger state sizes demonstrate improved recall quality, both for the models with CYLON partitioned and default, non-partitioned recurrent states. We validate that the average scores of the CYLON models preserve or improve upon the scores of the non-partitioned state models averaged across state sizes and tasks: across the 3 recall task scores (which are presented as an average in Table 3), we see a 50% win-rate for CYLON models versus default Hedgehog models, 83% win-rate for CYLON models versus default Mamba-2 models, and 67% win-rate for CYLON models versus default Gated Deltanet models.

4.3 Efficiency Evaluations

We benchmark our CYLON kernels on both NVIDIA Hopper and Blackwell GPUs. In Tables 4 and 5, we compare to the widely used reference linear attentions (Flash linear attention (Yang & Zhang, 2024), Mamba-2 (Dao & Gu, 2024)). We benchmark kernels at sequence length 4096 tokens, model dimension 2048, and batch size 8 reporting the average of 100 iterations after 20 warmup iterations. We compute model state sizes using the equations in Appendix A.

Table 4 shows that CYLON achieves up to a $6\times$ speedup over chunkwise linear attention on both H100 and B200 GPUs and up to a $2.6\times$ speedup over Mamba-2 on the H100 GPU. Table 5 demonstrates up to a $10\times$ speedup for the linear attention backward pass on the H100 GPU. We observe that the reference kernels are brittle, failing to run (e.g., due to running out of memory) without error at the largest state sizes that we evaluate (e.g.,

(a) Linear Attention (H100)								
State Size (MB)	≤ 2	2-4	4-8	8-16	16-32	32-64	64-128	≥ 128
CYLON (CUDA)	249	345	569	1.2k	2.3k	4.8k	9.5k	19.1k
CYLON (Triton)	1.3k	1.7k	2.3k	3.6k	6.2k	11.4k	21.7k	—
Non-part. (Triton)	925	1.3k	2.4k	4.6k	12.6k	28.3k	—	—
CYLON (Tri.) Speedup	0.7×	0.8×	1.0×	1.3×	2.1×	2.5×	—	—
CYLON (CUDA) Speedup	3.7×	3.9×	4.3×	4.0×	5.5×	6.0×	—	—

(b) Linear Attention (B200)								
State Size (MB)	≤ 2	2-4	4-8	8-16	16-32	32-64	64-128	≥ 128
CYLON	115	195	364	645	1.3k	2.7k	5.4k	11k
Non-partitioned	560	864	1.4k	2.8k	7.8k	15.6k	31.0k	62.3k
CYLON speedup	4.9×	4.4×	3.8×	4.3×	6.0×	5.8×	5.7×	5.7×

(c) Mamba-2 (H100)								
State Size (MB)	≤ 2	2-4	4-8	8-16	16-32	32-64	64-128	≥ 128
CYLON	708	1.0k	1.8k	3.4k	6.5k	12.8k	25.3k	50.3k
Non-partitioned	1.4k	2.4k	4.8k	9.0k	—	—	—	—
Speedup	2.0×	2.3×	2.6×	2.6×	—	—	—	—

(d) Gated Deltanet (H100)								
State Size (MB)	≤ 2	2-4	4-8	8-16	16-32	32-64	64-128	≥ 128
CYLON	586	985	1.8k	3.6k	7.2k	14.5k	29.3k	58.7k
Non-partitioned	1.2k	1.8k	—	—	—	—	—	—
Speedup	2.0×	1.8×	—	—	—	—	—	—

Table 4: Forward pass performance across state sizes on NVIDIA Hopper and Blackwell GPUs, measured in microseconds per iteration. We report time per iteration for each method at different recurrent state sizes. Missing measurements indicate configurations that exceed resource limits or are unsupported. CYLON offers up to 6× higher throughput and unlocks large state sizes (≥ 131 MB).

≥ 131 MB). Most notable, existing Gated Deltanet kernels enforce a hard restriction on the feature dimension, capping our state size at 2MB as reflected in Tables 3 and 4. This points to how CYLON unlocks a new operating point for AI: state size is critical to model quality, and CYLON allows model designers to use larger state sizes with significantly improved efficiency compared to prior methods.

Model / State Size	≤ 2	2-4	4-8	8-16	16-32	32-64	64-128	≥ 128
CYLON	410	583	1.1k	2.3k	4.6k	9.5k	18.7k	37.0k
Non-partitioned	1.7k	2.9k	5.6k	11.9k	46.1k	92.2k	—	—
CYLON Speedup	4.1×	5.0×	5.1×	5.2×	10.0×	9.7×	—	—

Table 5: Backward pass performance for linear attention across state sizes on NVIDIA Hopper, measured in microseconds per iteration. CYLON offers up to 10× higher throughput and unlocks large state sizes (≥ 64 MB).

5 Discussion & Conclusion

This work tackles a long-standing bottleneck in linear attention: scaling recurrent states without collapsing throughput. Large states are essential for recall – a key skill for reasoning and long-context modeling – but linear attention suffers severe slowdowns as state scales. We propose CYLON, a hardware-aware method for partitioning recurrent states across multiprocessors and combining them asynchronously. On Hopper and Blackwell GPUs, which embody the shift toward asynchronous execution, CYLON achieves up to 6× higher forward pass throughput and 10× higher backward pass throughput at large state sizes compared to widely used kernels. CYLON unlocks larger recurrent states (≥ 131 MB) than supported by the baseline kernels. CYLON consistently preserves quality on downstream tasks, expanding the Pareto frontier of the recall–efficiency tradeoff. Overall, we show that CYLON makes large-state linear attention practical on modern GPUs, providing a path to efficient and scalable long-context models.

Limitations. Although we alleviate local register pressure, our strategy remains bounded by the total register memory available on a GPU. Additionally, CYLON’s speedups are most pronounced for memory-bound kernels; the gains are smaller for more compute-intensive kernels such as Gated Deltanet.

Future trends. CYLON’s asynchronous design principles align with emerging hardware trends. AMD’s upcoming GFX1250 chip will introduce an asynchronous memory accelerator analogous to TMA (AMD, 2025b). NVIDIA’s sixth-generation NVLink for the Rubin architecture emphasizes accelerated, asynchronous in-network compute for operations such as reductions (NVIDIA, 2026b). These trends suggest that asynchrony will be a critical feature of utilizing modern hardware.

Reproducibility statement

We provide all necessary pointers to replicate our results. The model and algorithmic details are specified in Section 3, with additional experimental details such as the exact hyperparameters used in Appendix A. We also list compute resources and the code we used for training in Appendix A. For theoretical results, we include complete proofs of the claims in Appendix C.

Acknowledgements

The authors thank Yasa Baig, Vivien Cheng, Catherine Deng, Andy Dimnaku, Owen Dugan, Dan Fu, Roberto Garcia, Albert Gu, Neel Guha, Junmiao Hu, Hermann Kumbong, Jerry Liu, Hayden Prairie, Stuart Sul, and Michael Zhang for their helpful feedback and discussion.

We gratefully acknowledge the support of NIH under Nos. U54EB020405 (Mobilize) and R01GM124443 (SimTK); NSF under Nos. CCF2247015 (Hardware-Aware), CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), 1937301 (RTML), 1900638 (Open Federated Virtual Assistants), and 2028602 (PPoSS); DARPA under No. HR00112520038 (Fallingwater); US DEVCOM ARL under Nos. W911NF-23-2-0184 (Long-context) and W911NF-21-2-0251 (Interactive Human-AI Teaming); NSTXL under No. N00164-23-9-G057-01 (Energy-Efficient AI Hardware); ONR under Nos. N000142312633 (Deep Signal Processing), N000142212426 (Misinformation Analysis), N000142012480 (Non-Euclidean Geometry), and N000142012275 (Data Programming); Stanford HAI under Nos. 247183, 215955, and 345077 (Evo); Google DeepMind; Google Research; Google Cloud; NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, Salesforce, Total, the Laude Institute, Prime Intellect, Anthropic, Together AI, the HAI-GCP Cloud Credits for Research program, the Stanford Data Science Initiative (SDSI), the Stanford Marlowe Computing Platform, and members of the Stanford DAWN project: Meta, Google, and VMWare; and members of the Stanford SEAMS project: IBM and Felicis. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of NIH, ONR, DARPA, or the U.S. Government.

References

- Josh Alman and Zhao Song. Fast attention requires bounded entries, 2023. URL <https://arxiv.org/abs/2302.13214>.
- AMD. Cdna4 instruction set architecture, 2025a. URL <https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instruction-set-architectures/amd-instinct-cdna4-instruction-set-architecture.pdf>.
- AMD. [amd] add initial support for tdm on gfx1250. <https://github.com/triton-lang/triton/pull/8333>, 2025b. Triton pull request #8333.
- AMD. Amd helios rack-scale solution, 2026. URL <https://www.amd.com/content/dam/amd/en/documents/solutions/ai/amd-helios-bro.pdf>.
- Anthropic. Anthropic expands partnership with google and broadcom for multiple gigawatts of next-generation compute, 2026. URL <https://www.anthropic.com/news/google-broadcom-partnership-compute>.
- Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *The Eleventh International Conference on Learning Representations*, 2023.
- Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *International Conference on Machine Learning*, 2024.

Michael Bauer, Henry Cook, and Brucek Khailany. Cudadma: Optimizing gpu memory bandwidth via warp specialization. *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

Cartesia. Sonic, 2025. URL <https://cartesia.ai/sonic>.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

d Matrix. Scaling ai the right way: Introducing our rack-level inference solution, 2025. URL <https://www.d-matrix.ai/scaling-ai-the-right-way-introducing-our-rack-level-inference-solution/>.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *International Conference on Machine Learning (ICML)*, 2024.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojuan Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yudian Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL <https://arxiv.org/abs/2405.04434>.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiabin Xu,

Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuan Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yudian Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL <https://arxiv.org/abs/2606.19348>.

Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. Dom-Im: Learning generalizable representations for html documents. 2022.

Etched. Frontier inference clusters, 2026. URL <https://www.etched.com/progress/frontier-inference-clusters>.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL <https://zenodo.org/records/10256836>.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin,

- Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiabin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yueshan Bi, Yuhao Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL <https://arxiv.org/abs/2602.15763>.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *International Conference on Learning Representations (ICLR)*, 2021.
- Yuxian Gu, Qinghao Hu, Shang Yang, Haocheng Xi, Junyu Chen, Song Han, and Han Cai. Jet-nemotron: Efficient language model with post neural architecture search, 2025. URL <https://arxiv.org/abs/2508.15884>.
- Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International Conference on Machine Learning*, pp. 9099–9117. PMLR, 2022.
- A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020. URL <https://arxiv.org/abs/2006.16236>.
- Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *34th International Conference on Algorithmic Learning Theory*, volume 201, pp. 1–23, 2023.
- Kimi, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chen-zhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran

He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiayi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Mingqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tianian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapai Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qian Kang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Wei Yu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL <https://arxiv.org/abs/2607.24653>.

Team Kimi, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiayi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun

- Du. Kimi linear: An expressive, efficient attention architecture, 2025. URL <https://arxiv.org/abs/2510.26692>.
- Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. Minimax sparse attention, 2026. URL <https://arxiv.org/abs/2606.13392>.
- Liquid. Introducing lfm2: The fastest on-device foundation models on the market, 2025. URL <https://www.liquid.ai/blog/liquid-foundation-models-v2-our-second-series-of-generative-ai-models>.
- William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Chuan Li, Kyle Lo, Saumya Malik, DJ Matusz, Benjamin Minixhofer, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A. Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. Olmo Hybrid: From theory to practice. Technical report, Allen Institute for AI, 2026. URL <https://allenai.org/papers/olmo-hybrid>.
- NVIDIA. Nvidia A100 tensor core GPU architecture, 2020.
- NVIDIA. Nvidia H100 tensor core GPU architecture, 2022.
- NVIDIA. Nvidia blackwell architecture technical brief, 2025. URL <https://resources.nvidia.com/en-us-blackwell-architecture/>.
- NVIDIA. Nemotron 3 super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2026a. URL <https://research.nvidia.com/labs/nemotron/files/NVIDIA-Nemotron-3-Super-Technical-Report.pdf>.
- NVIDIA. Inside the nvidia rubin platform: Six new chips, one ai supercomputer, 2026b. URL <https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer/>.
- OpenAI. Openai and broadcom unveil llm-optimized inference chip, 2026. URL <https://openai.com/index/openai-broadcom-jalapeno-inference-chip/>.
- Michael Poli, Jue Wang, Stefano Massaroli, Jeffrey Quesnelle, Ryan Carlow, Eric Nguyen, and Armin Thomas. StripedHyena: Moving Beyond Transformers with Hybrid Signal Processing Models. 12 2023. doi: 10.57967/hf/1595. URL <https://github.com/togethercomputer/stripedhyena>.
- Qwen. Qwen3-next: Towards ultimate training & inference efficiency, 2025. URL <https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd>.
- Qwen Team. Qwen3.8: A new bar for coding and cowork, August 2026. URL <https://qwen.ai/blog?id=qwen3.8>.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for squad. *ACL*, 2018.
- Benjamin Frederick Spector, Simran Arora, Aaryan Singhal, Arjun Parthasarathy, Daniel Y Fu, and Christopher Re. Thunderkittens: Simple, fast, and $\text{\textit{Adorable}}$ kernels. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=0fJfVOSUra>.
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.
- Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.
- Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Rif A. Saurous, Guillaume Lajoie, Charlotte Frenkel, Razvan Pascanu, Blaise Agüera y Arcas, and João Sacramento. Mesanet: Sequence modeling by locally optimal test-time training, 2025.
- Eric Wu, Kevin Wu, Roxana Daneshjou, David Ouyang, Daniel Ho, and James Zou. How medical ai devices are evaluated: limitations and recommendations from an analysis of fda approvals. *Nature Medicine*, 27:1–3, 04 2021.
- Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL <https://github.com/sustcsonglin/flash-linear-attention>.
- Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *International Conference on Machine Learning*, 2023.
- Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025a. URL <https://arxiv.org/abs/2412.06464>.
- Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025b. URL <https://arxiv.org/abs/2406.06484>.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL <https://arxiv.org/abs/2502.11089>.
- Chengruidong Zhang, Xi Lin, Huiqiang Jiang, Zekun Wang, Xiao Li, Yizhong Cao, Bohan Zhuang, Rui Men, Jianwei Zhang, Bo Zheng, Junyang Lin, Dayiheng Liu, and Jingren Zhou. Flashqla: Flash qwen linear attention. <https://github.com/QwenLM/FlashQLA>, 2026.
- Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. *International Conference on Learning Representations (ICLR)*, 2024.

Our appendix is organized as follows:

1. Appendix A contains additional experimental details.
2. Appendix B contains additional experimental results.
3. Appendix C contains proofs for the theoretical analysis.

LLM Usage

Aside from revision and programming assistance, we used LLMs for typesetting tables and math into LaTeX. Additionally, we used LLMs for revising and clarifying mathematical reasoning.

A Experiment details

We use H100, A100, and B200 80GB NVIDIA GPUs to run all experiments. We use training infrastructure closely adapted from the FlashAttention code base: <https://github.com/Dao-AI-Lab/flash-attention/tree/main> for all pretraining runs (Dao et al., 2022). Our pretraining data is pulled from The Pile corpus (Gao et al., 2020) and is tokenized using the GPT2BPETokenizer (Radford et al., 2019). This section provides additional details for our model hyperparameters and configurations.

A.1 Model configurations

Mamba-2 (Dao & Gu, 2024) We train using the implementation from the provided reference at <https://github.com/state-spaces/mamba>. We compute the state size for feature dimension d_{state} , model dimension d , value heads H , and expansion factor E as:

$$\text{sizeof}(s_i) = E \times d_{\text{state}} \times d$$

Note that Mamba-2 uses Multi Value Attention. We use the hyperparameters from Table 6.

Table 6: Mamba-2 Training Settings

Optimizer	AdamW
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$
Optimizer eps	$1e - 8$
Precision	BFloat16
Warmup	1%
Learning rate decay	Cosine
Learning rate (min, base)	$8e-5, 8e-4$
Global batch size	256
Weight decay	0.1
Num Layers	46
Hidden Size	2048
RMSNorm	True
Norm Epsilon	$1e - 5$
Dt State	16
Dt (Min, Max)	(0.001, 0.1)
Dt Init. Strategy	Random
Dt Init. Floor	$1e - 4$
Dt Scale	1.0
Dt Softplus	True
Projection Expansion Factor	2

Hedgehog (Zhang et al., 2024) We use the implementation of Hedgehog from the provided reference at <https://github.com/HazyResearch/lolcats>. We compute the state size for feature dimension d_{feature} , model dimension d , and value heads H as:

$$\text{sizeof}(s_i) = \frac{d_{\text{feature}} \times d}{H}$$

We train using the Based model backbone (Arora et al., 2024) from the provided reference at <https://github.com/HazyResearch/based>, which includes a hybrid of short convolution BaseConv (Arora et al., 2023), sliding window attention, and linear attention layers. This backbone provides a simple way to improve quality over pure linear attention. We use the hyperparameters from Table 7.

Table 7: Hedgehog Training Settings

Optimizer	Adam
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$
Optimizer eps	$1e - 8$
Precision	BFloat16
Warmup	1%
Learning rate decay	Cosine
Learning rate (min, base)	$8e-5, 8e-4$
Global batch size	256
Weight decay	0.1
Num Layers	26
Hidden Size	2048
Num Heads	16
Head Dim	128
RMSNorm	True
Norm Epsilon	$1e - 5$
Projection Expansion Factor	4
Short Conv Filter Size	4

Gated Deltanet (Yang et al., 2025a) We train using the implementation from the provided reference at <https://github.com/fla-org/flash-linear-attention>. We compute the state size for feature dimension model dimension d and value heads H as:

$$\text{sizeof}(s_i) = \frac{d^2}{H}$$

Table 8: Gated Deltanet Training Settings

Optimizer	AdamW
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$
Optimizer eps	$1e - 8$
Precision	BFloat16
Warmup	1%
Learning rate decay	Cosine
Learning rate (min, base)	$8e-5, 8e-4$
Global batch size	256
Weight decay	0.1
Num Layers	26
Hidden Size	2048
Num Heads	16
Head Dim	128
Num V Heads	16
RMSNorm	True
Norm Epsilon	$1e - 6$
MLP Expansion Factor	2
Short Conv Filter Size	4

A.2 State sizes of open-source models

Throughout the paper, we discuss the state size of different popular linear attention models. Here we provide the basis for our computations:

Model	Heads	$d_{q/k}$	d_v / d_{state}	dtype	Size
Olmo Hybrid 7B (Merrill et al., 2026)	30	96	192	fp32	~ 2.0 MB
Nemotron 3 Super 102B (NVIDIA, 2026a)	128	64	128	fp32	~ 4.0 MB
Qwen3-Next (Qwen, 2025)	32	128	128	fp32	~ 2.0 MB
Kimi Linear (Kimi et al., 2025)	32	128	128	fp32	~ 2.0 MB
Jet-Nemotron (Gu et al., 2025)	16	128	256	fp32	~ 2.0 MB
GLA (Yang et al., 2023)	8	128	128	fp32	~ 0.5 MB
DeltaNet (Yang et al., 2025b)	20	128	128	fp32	~ 1.3 MB
Mamba2 2.7B (Dao & Gu, 2024)	80	64	128	fp32	~ 2.6 MB

Table 9: Comparison of head configurations, dimensions, and state sizes.

For Mamba2 2.7B, we include the expansion factor of 2 in $d_{q/k}$. Hyperparameters for GLA and Deltanet are taken from FLA Hub checkpoints.

A.3 Evaluation Details

In the following table, we include maximum sequence lengths seen in each evaluation task.

Task	Category	Max Seq. Len.
PiQA (0-shot)	NLU	255
ARC-Easy (0-shot)	NLU	159
ARC-Challenge (0-shot)	NLU	171
HellaSwag (0-shot)	NLU	152
WinoGrande (0-shot)	NLU	39
FDA	Recall	2001
SWDE	Recall	1580
SQuAD-Completion	Recall	1984
NIAH Single 1	RULER	1840
NIAH Single 2	RULER	1939
NIAH Single 3	RULER	1376
NIAH Multikey 1	RULER	1405
NIAH Multikey 2	RULER	1895
NIAH Multikey 3	RULER	1769
NIAH Multiquery	RULER	1442
NIAH Multivalue	RULER	1412
RULER VT	RULER	1776
RULER CWE	RULER	1697
RULER FWE	RULER	2047
RULER QA (HotpotQA)	RULER	2016

Table 10: Maximum sequence length per evaluation task.

B Additional Results

Here we include additional experimental results.

B.1 Full Evaluation Tables

In Table 3, we present an average recall score across three evaluations (FDA, SWDE, SQUAD). In Table 11 and Table 12 we provide a breakdown of each tasks scores. In Table 13, we provide evaluation scores on RULER, an additional suite of recall benchmarks. Due to the noisy signal in these task scores, we omit them from the body of our work.

Architecture	Method	Params/ Tokens	State Sz./ MB/lyr	NLU				
				PiQA Acc. ↑	ARC-e Acc. ↑	ARC-c Acc. ↑	HellaSwag Acc. ↑	Winogrande Acc. ↑
Transformer	—	1.3B/50B	33.5	69.7	53.2	23.9	36.7	54.8
Hedgehog	Non-partitioned	1.2B/50B	1.0	69.3	54.4	22.9	36.1	54.5
	CYLON ($m=2$)	1.2B/50B	2.0	68.6	55.5	24.3	36.1	52.6
	Non-partitioned	1.2B/50B	2.0	69.4	54.7	23.5	36.2	53.4
	CYLON ($m=4$)	1.2B/50B	4.0	70.0	55.3	24.2	36.2	51.1
	Non-partitioned	1.2B/50B	4.0	69.6	53.7	22.5	35.9	51.9
Mamba-2	Non-partitioned	1.4B/50B	2.1	69.3	55.0	23.4	36.6	53.5
	CYLON ($m=2$)	1.4B/50B	4.2	68.8	54.7	23.4	36.5	53.2
	Non-partitioned	1.4B/50B	4.2	68.2	54.9	23.2	36.4	52.2
	CYLON ($m=4$)	1.4B/50B	8.4	70.7	55.9	25.3	36.5	54.6
	Non-partitioned	1.4B/50B	8.4	69.6	54.9	22.7	35.9	53.9
Gated Deltanet	Non-partitioned	1.2B/50B	1.0	69.2	52.1	22.6	35.1	53.8
	CYLON ($m=2$)	1.4B/50B	2.0	67.7	53.8	23.3	35.4	52.4
	Non-partitioned	1.4B/50B	2.0	69.4	54.0	21.6	35.8	54.1
	CYLON ($m=4$)	1.7B/50B	4.0	68.6	54.8	22.7	35.6	55.5
	Non-partitioned	1.7B/50B	4.0	—	—	—	—	—

Table 11: **Per-task NLU breakdown.** Individual zero-shot accuracy on PiQA, ARC-Easy, ARC-Challenge, HellaSwag, and Winogrande. For each matched pair, we **bold** the better score.

Architecture	Method	Params/ Tokens	State Sz./ MB/lyr	Info. Extraction				QA	
				FDA Acc. ↑	FDA F1 ↑	SWDE Acc. ↑	SWDE F1 ↑	SQUAD Acc. ↑	SQUAD F1 ↑
Transformer	—	1.3B/50B	33.5	76.0	82.5	73.0	75.8	39.0	38.4
Hedgehog	Non-partitioned	1.2B/50B	1.0	31.7	42.7	37.3	39.8	35.8	32.9
	CYLON ($m=2$)	1.2B/50B	2.0	27.5	39.5	51.3	54.9	34.9	32.8
	Non-partitioned	1.2B/50B	2.0	31.8	45.8	47.2	49.6	34.3	32.0
	CYLON ($m=4$)	1.2B/50B	4.0	29.2	40.3	46.8	50.3	38.8	35.9
	Non-partitioned	1.2B/50B	4.0	31.3	45.6	50.7	54.0	33.9	31.6
Mamba-2	Non-partitioned	1.4B/50B	2.1	31.3	40.3	49.8	53.1	33.8	31.9
	CYLON ($m=2$)	1.4B/50B	4.2	29.7	34.7	50.7	51.0	34.4	32.2
	Non-partitioned	1.4B/50B	4.2	27.7	38.3	54.6	58.1	32.9	30.9
	CYLON ($m=4$)	1.4B/50B	8.4	37.0	46.6	51.4	54.0	36.3	34.6
	Non-partitioned	1.4B/50B	8.4	33.9	44.6	47.6	51.0	32.9	31.1
Gated Deltanet	Non-partitioned	1.2B/50B	1.0	27.3	38.8	50.1	53.7	32.1	31.4
	CYLON ($m=2$)	1.4B/50B	2.0	33.2	45.2	49.7	52.6	34.6	33.1
	Non-partitioned	1.4B/50B	2.0	26.3	39.0	52.9	56.2	29.6	29.1
	CYLON ($m=4$)	1.7B/50B	4.0	38.4	50.5	53.0	56.3	32.8	32.3
	Non-partitioned	1.7B/50B	4.0	—	—	—	—	—	—

Table 12: **Per-task information extraction and QA breakdown.** Individual accuracy on FDA, SWDE, and SQUAD. For each matched pair, we **bold** the better score.

Architecture	Method	Params/ Tokens	State Sz./ MB/lyr	RULER											
				NIAH Multikey 1	NIAH Multikey 2	NIAH Multikey 3	NIAH Single 1	NIAH Single 2	NIAH Single 3	NIAH Multiquery	NIAH Multivalue	VT	CWE	FWE	QA Hotpot
Transformer	—	1.3B/50B	33.5	73.2	9.2	17.6	100.0	99.6	99.8	59.6	59.8	28.9	18.1	17.5	0.4
Hedgehog	Non-partitioned	1.2B/50B	1.0	30.4	0.2	0.0	99.4	33.0	4.8	23.8	21.7	2.0	0.1	26.7	16.2
	CYLON ($m=2$)	1.2B/50B	2.0	30.2	0.2	0.0	98.6	93.4	97.0	30.2	0.2	0.0	0.2	45.2	15.2
	Non-partitioned	1.2B/50B	2.0	30.2	0.6	0.0	98.6	97.8	98.8	37.2	26.7	5.7	0.1	7.7	16.6
	CYLON ($m=4$)	1.2B/50B	4.0	30.4	0.2	0.2	100.0	89.2	92.2	25.9	11.7	2.6	0.1	29.5	15.0
	Non-partitioned	1.2B/50B	4.0	32.0	0.2	0.2	99.6	87.8	76.4	33.6	25.8	2.4	5.9	30.1	18.6
Mamba-2	Non-partitioned	1.4B/50B	2.1	30.6	0.2	0.0	99.8	81.8	86.8	27.4	18.9	5.6	0.3	3.6	15.0
	CYLON ($m=2$)	1.4B/50B	4.2	26.8	1.0	0.0	98.4	64.4	93.0	38.4	32.1	2.8	2.8	50.6	14.6
	Non-partitioned	1.4B/50B	4.2	29.2	1.0	0.0	100.0	72.6	99.2	17.8	24.3	2.5	1.6	20.9	16.4
	CYLON ($m=4$)	1.4B/50B	8.4	34.4	1.0	0.0	100.0	75.0	95.0	38.5	36.9	14.7	0.3	6.3	14.8
	Non-partitioned	1.4B/50B	8.4	24.4	0.8	0.0	99.6	88.4	95.2	34.5	26.8	4.6	21.5	35.3	17.2
Gated Deltanet	Non-partitioned	1.2B/50B	1.0	27.0	1.0	0.0	100.0	47.8	85.4	3.2	2.6	12.5	1.3	40.0	14.6
	CYLON ($m=2$)	1.4B/50B	2.0	23.4	0.8	0.0	100.0	45.8	81.4	24.0	37.9	1.6	1.0	41.7	16.6
	Non-partitioned	1.4B/50B	2.0	26.2	1.6	0.0	100.0	24.0	61.6	1.0	4.6	5.1	0.8	28.3	14.0
	CYLON ($m=4$)	1.7B/50B	4.0	33.2	0.6	0.6	99.8	58.0	86.4	8.6	8.5	2.9	2.0	45.8	18.8
	Non-partitioned	1.7B/50B	4.0	—	—	—	—	—	—	—	—	—	—	—	—

Table 13: **Per-task RULER breakdown.** Individual accuracy across the 12 RULER subtasks. For each matched pair, we **bold** the better score.

B.2 Extended Benchmarking Figures

Below we provide additional benchmarking figures for our linear attention kernels, swept across batch sizes and sequence lengths.

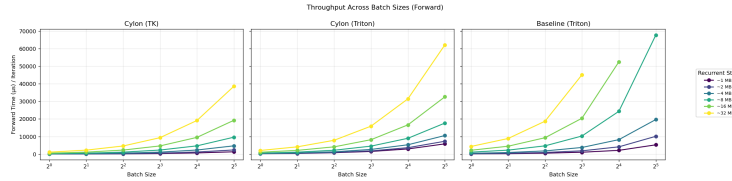


Figure 3: Linear attention forward kernel benchmark results, swept across batch size.

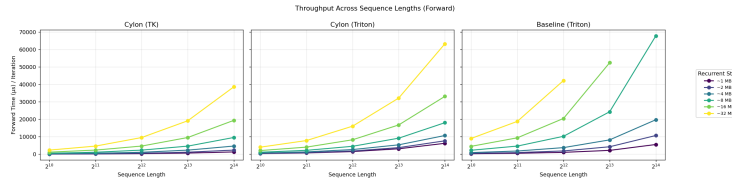


Figure 4: Linear attention forward kernel benchmark results, swept across sequence length.

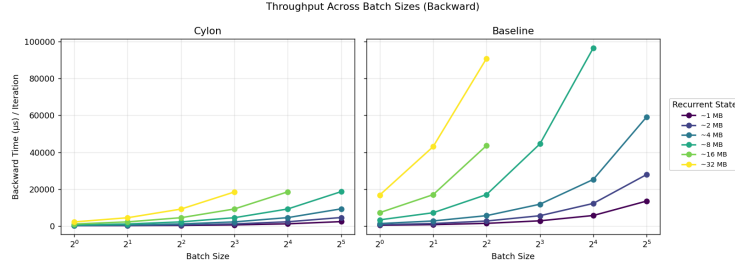


Figure 5: Linear attention backward kernel benchmark results, swept across batch size.

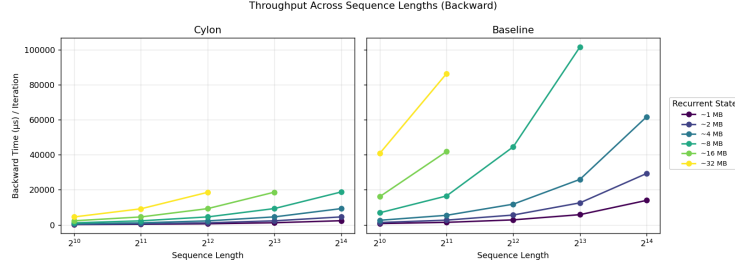


Figure 6: Linear attention backward kernel benchmark results, swept across sequence length.

B.3 Extended Profiling Figures

We report additional profiling results in Table 14 to characterize the impact of asynchronous reductions.

Kernel	Feature Maps	HBM Throughput (Gb/s)	Occupancy (%)	Tensor-Core Utilization (%)	Energy (J)
Linear Attention	1	893.95	12.46	29.44	1.37
CYLON	1	834.47	12.36	24.04	1.41
CYLON	2	919.29	12.36	24.09	2.83
CYLON	4	926.48	12.36	24.13	5.63
CYLON	8	932.30	12.36	24.15	11.56
CYLON	16	940.88	12.36	24.16	23.74

Table 14: **Additional profiling metrics.** We compare our baseline Hedgehog linear attention kernel with CYLON. These kernels only differ by CYLON’s use of asynchronous reduction operations in comparison to an asynchronous store.

In Table 4, we benchmark CYLON with batch size 8, 16 heads, and sequence length 4096. As we increase the number of feature maps, the throughput decreases, as expected. However, to what extent is this attributable to TMA asynchronous reduction overheads? To isolate this quantity, we benchmark our kernel running on only one SM v all SMs, each reducing to the same output. We report these results in Table 15.

B.4 Architecture Ablations

Mamba2 Feature Activation. We apply several non-linear element-wise maps to the BC and C tensors, as is ablated in the original Mamba-2 paper (Dao & Gu, 2024). In addition, we apply a learned Hedgehog feature map (Zhang et al., 2024). We find that our fixed non-linear maps do not impact the model’s perplexity, confirming the results from the original Mamba-2 paper, but degrades its performance on recall evaluations by up to 7%. Moreover, we find that including a learned Hedgehog feature map degrades model performance by nearly 1 point of perplexity and recall evaluations by up to 7%.

SMs Resident	Micro Sec / Iter
1	346.16
2	339.62
4	326.78
8	335.23
16	332.23
32	322.10
64	322.89
114	341.51

Table 15: **TMA Overhead.** We benchmark our CYLON kernel with batch size 1, head count 1, and sequence length 4096. We sweep the number of feature maps from 1 to 114, such that the number of thread blocks never exceeds the GPU’s SM count.

B.5 Cost Model Experiments

In Figure 7, we include a sweep of runtimes across state sizes (via head dimension ablation) and sequences lengths for the Flash Linear Attention parallel, chunked, and fused chunked Triton kernels. We show that at large state sizes and sequence lengths, the chunked kernel is fastest.

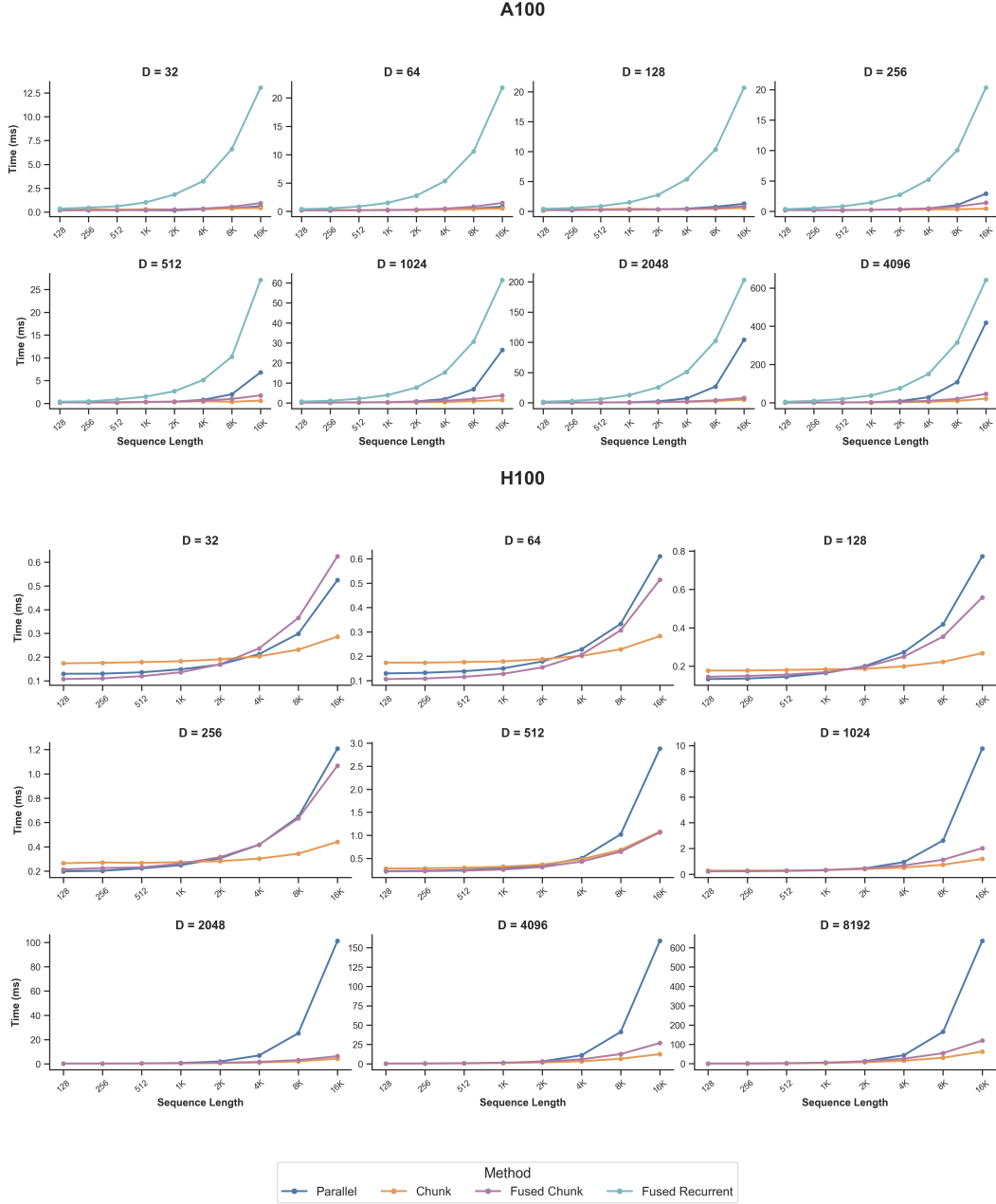


Figure 7: Sweep across prefill times. Each plot is set to a fixed head dimension. For both A100 and H100 sweeps, we start with a head dimension of 32 and increase by a factor of 2x in the plots left-to-right. For the A100, we end at a head dimension of 4096 in the lower right. For the H100, we end at a head dimension of 8192 in the lower left.

B.6 Training Stability

In Figure 8, we plot feature map gradient statistics to demonstrate training stability across models with multiple feature maps.

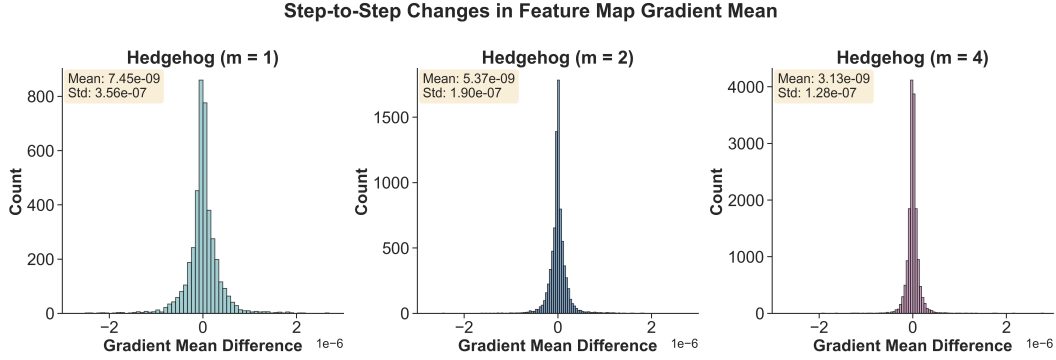


Figure 8: Training Stability. During training, we measure each parameter tensor’s gradient mean, max, and min at a regular interval of 26 steps. We compute the inter-step differences in gradient mean across all feature maps’ linear projections. We plot a distribution of these differences for three Hedgehog 1B parameter models trained on 10B tokens from The Pile.

C Theoretical analysis

C.1 Partitioned Feature Map Equivalence

A natural question is: *how does partitioning the state into disjoint components impact expressivity?* More formally, we study whether the behavior of linear attention with a single, non-partitioned feature map Φ can be reproduced with m independent feature maps $\phi_1, \dots, \phi_m$. In particular, we will assume a fixed Φ with some set of properties and then show that we can prove the existence of $\phi_1, \dots, \phi_m$ which satisfies the following

$$\frac{\Phi(q_i)^\top \sum_j^i (\Phi(k_j) v_j^\top)}{\Phi(q_i)^\top \sum_j^i \Phi(k_j)} = \frac{\sum_k^m \phi_k(q_i)^\top \sum_j^i (\phi_k(k_j) v_j^\top)}{\sum_k^m \phi_k(q_i)^\top \sum_j^i \phi_k(k_j)} \quad (9)$$

where $q_i, k_j, v_j \in \mathbb{R}^d$, $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{mD}$, and $\phi_k : \mathbb{R}^d \rightarrow \mathbb{R}^D$ for all k such that $1 \leq k \leq m$.

C.1.1 Unbounded Equivalence

We first prove Equation 9 without making any assumptions regarding the structure of Φ or placing significant restrictions on $\phi_1, \dots, \phi_m$.

Theorem C.1 (Unbounded equivalence). *For any feature map $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{mD}$, there exist feature maps $\phi_1, \dots, \phi_m : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that Equation 9 holds.*

Proof. Let us construct $\phi_1, \dots, \phi_m$. Let $\phi_i(x) = \Phi(x)[(i-1) \cdot D : i \cdot D]$ ³. Given this construction, we know that $\Phi(x) = [\phi_1(x)^\top, \dots, \phi_m(x)^\top]^\top$ is true. Thus, we can derive the following.

$$\frac{\Phi(q_i)^\top \sum_j^n (\Phi(k_j) v_j^\top)}{\Phi(q_i)^\top \sum_j^n \Phi(k_j)} = \frac{[\phi_1(q_i)^\top, \dots, \phi_m(q_i)^\top] \sum_j^n ([\phi_1(k_j)^\top, \dots, \phi_m(k_j)^\top]^\top v_j^\top)}{[\phi_1(q_i)^\top, \dots, \phi_m(q_i)^\top] \sum_j^n [\phi_1(k_j)^\top, \dots, \phi_m(k_j)^\top]^\top} \quad (10)$$

$$= \frac{\sum_j^n [\phi_1(q_i)^\top, \dots, \phi_m(q_i)^\top] [\phi_1(k_j)^\top, \dots, \phi_m(k_j)^\top]^\top v_j^\top}{\sum_j^n [\phi_1(q_i)^\top, \dots, \phi_m(q_i)^\top] [\phi_1(k_j)^\top, \dots, \phi_m(k_j)^\top]^\top} \quad (11)$$

$$= \frac{\sum_j^n \sum_k^m (\phi_k(q_i)^\top \phi_k(k_j)) v_j^\top}{\sum_j^n \sum_k^m (\phi_k(q_i)^\top \phi_k(k_j))} \quad (12)$$

$$= \frac{\sum_k^m \phi_k(q_i)^\top \sum_j^n \phi_k(k_j) v_j^\top}{\sum_k^m \phi_k(q_i)^\top \sum_j^n \phi_k(k_j)} \quad (13)$$

Thus, we have demonstrated Equation 9 in a minimal setting. $\square$

One may notice that this is unsatisfying, however, since $\phi_1, \dots, \phi_m$ are unrestricted and thus enjoy unlimited computational resources. In the following sections, we further characterize our claims to be more meaningful in our applied setting, where we require $\phi_1, \dots, \phi_m$ to be individually "cheaper" than Φ .

C.1.2 Elementwise Nonlinear Maps

Let us introduce structure to Φ . Pick some $A \in \mathbb{R}^{Dm \times d}$ and $b \in \mathbb{R}^{Dm}$ and let σ be some non-linear function. We define the function $\Phi(x) = \sigma(Ax + b)$ for any $x \in \mathbb{R}^d$. In this section,

³We borrow the $A[x : y]$ operator from Python, where it denotes a "slice" of tensor A ranging from position x to y in its first dimension

we will prove Equation 9 assuming that σ is an elementwise function⁴. More formally, for any $z \in \mathbb{R}^a$ for some arbitrary non-zero integer a , $\sigma(z)[i] = \sigma(z[i])$ is true for some index $1 \leq i \leq a$. Moreover, let $C(\cdot)$ denote the number of primitive operations required by a function. Primitive operation include arithmetic operations, exponentiation/logarithms, ReLU, and other elementwise operations. Let us place an additional restriction on $\phi_1, \dots, \phi_m$ such that $C(\Phi) = \sum_{k=1}^m C(\phi_k)$ must be true.

Theorem C.2 (Elementwise Nonlinear Maps). *Let $\Phi(x) = \sigma(Ax + b)$ where σ is elementwise. There exist $\phi_1, \dots, \phi_m : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that Equation 9 holds and $C(\Phi) = \sum_{k=1}^m C(\phi_k)$.*

Proof. We construct $\phi_1, \dots, \phi_m$ such that

$$\Phi(x) = [\phi_1(x)^\top, \dots, \phi_m(x)^\top]^\top$$

is true. If we can demonstrate that our construction satisfies this equality, then Equation 9 follows from Theorem C.1.

We produce this construction by decomposing A and b along their second and first dimensions, respectively, to produce $A_1, \dots, A_m \in \mathbb{R}^{D \times d}$ and $b_1, \dots, b_m \in \mathbb{R}^D$. Then, we define $\phi_i(x) = \sigma(A_i x + b_i)$ where σ is the same elementwise operation from ϕ . Now, we can easily derive the following. Recall that since σ is element-wise, we can write $\sigma(z) = [\sigma(z_1), \dots, \sigma(z_a)]$ if $z = [z_1, \dots, z_a]$.

$$\phi(x) = \sigma(Ax + b) \tag{14}$$

$$= \sigma([A_1^\top, \dots, A_m^\top]^\top x + [b_1^\top, \dots, b_m^\top]^\top) \tag{15}$$

$$= \sigma([(A_1 x + b_1)^\top, \dots, (A_m x + b_m)^\top]^\top) \tag{16}$$

$$= [\sigma(A_1 x + b_1)^\top, \dots, \sigma(A_m x + b_m)^\top]^\top \tag{17}$$

$$= [\phi_1(x)^\top, \dots, \phi_m(x)^\top]^\top \tag{18}$$

This proves the decomposition. It remains to verify the operation count. We know that $C(\Phi) = dDm + (d-1)Dm + Dm + Dm = (2d+1)Dm$. The first two terms account for computing Ax , the third term accounts for adding bias, and the final term accounts for computing σ . Moreover, by similar logic we know $C(\phi_k) = dD + (d-1)D + D + D = (2d+1)D$ for all $1 \leq k \leq m$. Thus, we know that $\sum_{k=1}^m C(\phi_k) = \sum_{k=1}^m (2d+1)D = (2d+1)Dm$, which is equal to $C(\Phi) = (2d+1)Dm$, as required. $\square$

Thus we have demonstrated a simple construction for partitioning Φ , showed equivalence between the partitioned and non-partitioned linear attention expressions, and proved that they use equal numbers of operations to compute.

C.1.3 Polynomial Expansion.

Proposition C.3 (Polynomial expansion as affine map). *Let $y = (Ax + b)^2$ for any $A \in \mathbb{R}^{Dm \times d}$, $b \in \mathbb{R}^{Dm}$, and $x \in \mathbb{R}^d$. There exists $\bar{A} \in \mathbb{R}^{Dm \times d^2 + d + 1}$, $\bar{b} \in \mathbb{R}^{Dm}$, and $\bar{x} \in \mathbb{R}^{d^2 + d + 1}$ such that $y = \bar{A}\bar{x} + \bar{b}$.*

Proof. Let us expand our expression as follows.

$$y_i = (\sum_j A_{ij} x_j + b_i)^2 \tag{19}$$

$$= \sum_j \sum_k A_{ij} A_{ik} x_j x_k + 2b_i \sum_j A_{ij} x_j + b_i^2 \tag{20}$$

⁴In practice, we set σ to be some non-linearity such as the exponential or ReLU

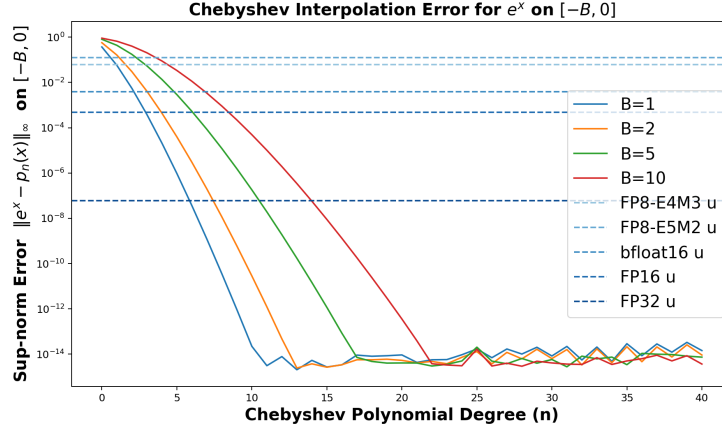


Figure 9: Sup-norm error $\|e^x - p_n(x)\|_\infty$ on $[-B, 0]$ for Chebyshev polynomials of degree n . Curves show different B values; dashed horizontal lines mark half-ULP thresholds for FP8, bfloat16, FP16, and FP32.

From this, we can produce the following construction. Let “vec” below denote the “flattening” of a matrix to a vector.

$$\bar{A}_i = [A_{i1}A_{i1}, A_{i1}A_{i2}, \dots, A_{i1}A_{id}, A_{i2}A_{i1}, \dots, A_{id}A_{id} \mid 2b_iA_{i1}, 2b_iA_{i2}, \dots, 2b_iA_{id} \mid 0] \quad (21)$$

$$\bar{b} = [b_1^2, b_2^2, \dots, b_{D_m}^2]^\top \quad (22)$$

$$\bar{x} = [\text{vec}(xx^\top)^\top, x^\top, 1]^\top \quad (23)$$

If we substitute our constructions into Equation 20, we produce our desired equality $y = \bar{A}\bar{x} + \bar{b}$, thus proving that a $\bar{A}, \bar{b}, \bar{x}$ exist which satisfy our claim. $\square$

Relationship to softmax attention. We present our polynomial expansion result to theoretically frame the relationship between softmax attention and CYLON. Recall that softmax attention kernels require quadratic compute and centralized work within a single SM, where kernels are typically bottlenecked on the special functional units that compute the exp (Spector et al., 2025). While the quadratic compute cannot be avoided (Alman & Song, 2023), CYLON suggests an alternate path to softmax attention: the exponential function can be approximated to within machine precision using low-degree polynomial expansions (e.g., Chebyshev or Taylor series) (Zhang et al., 2024; Keles et al., 2023; Arora et al., 2024). Each SM is assigned a subset of polynomial terms in the expansion of $\exp(x)$ and runs the chunkwise algorithm for the term. The partial results are then combined via asynchronous global reductions (TMA). This assigns the quadratic cost of softmax to a massively parallel grid of SMs. Motivating this feasibility, for common AI precision-levels (e.g., FP8), $\approx 3 - 5$ terms suffice for machine precision (Figure 9). This observation contextualizes CYLON’s efficient expansion of feature dimension as an approach to approximating softmax and thus unifying softmax and linear attention in practice.

C.2 Reduction Order Error

Let m be the number of feature maps we’re using i.e. the number of independent linear attention partitions. Let t be some token position. Then, we write the output of our partitioned linear attention block at position t as

$$o_t = \sum_{i=1}^m o_t^{(i)}$$

In practice, $o_t, o_t^{(i)}$ are represented in bf16 and reduced in a non-deterministic order. Thus some error is incurred via variance of reduction order. Here we study these effects.

C.2.1 Floating Point Basics

Let function $\text{fl} : \mathbb{R} \rightarrow \mathbb{R}$ be a function which "rounds" real numbers to their nearest floating point representation. The behavior of this function depends on one's choice of floating point representation. For any given representation format, we define the *unit roundoff* u , which bounds the relative rounding error of a single arithmetic operation. For bf16 (7 stored mantissa bits), $u = 2^{-8} \approx 0.0039$. Note that u is related to the more commonly cited machine epsilon ϵ_{mach} (the gap between 1 and the next representable float) by $u = \epsilon_{\text{mach}}/2$ under round-to-nearest.

For some operation $\circ \in \{+, -, \times, /\}$ and floating point values a and b , we know

$$(a \circ b)(1 - \delta) \leq \text{fl}(a \circ b) \leq (a \circ b)(1 + \delta) \quad (24)$$

Intuitively, the rounding error incurred by using floating point representations is bounded by a $1 \pm \delta$ coefficient since any error greater than this in magnitude could be captured by the floating point representation and thus would not be rounded away.

C.2.2 Sequential Summation Analysis

Preliminaries. Pick n floating point values $x_1, \dots, x_n$. While there are many summation orders, we only consider the left-to-right case to match our summation model (e.g. pairwise summation is not relevant).

First, we present a formal statement of "sequential summation."

Algorithm 3 Sequential Summation

```

1: Input:  $x_1, \dots, x_n$ .
2:  $\hat{S}_0 \leftarrow 0$ 
3: for  $k = 1$  to  $n$  do
4:    $\hat{S}_k \leftarrow \text{fl}(\hat{S}_{k-1} + x_k)$ 
5: return  $\hat{S}_n$ 

```

In this algorithm we know that $\hat{S}_n$ has the following structure.

$$\hat{S}_n = \text{fl}(x_n + \hat{S}_{n-1}) \quad (25)$$

$$= \text{fl}(x_n + \text{fl}(x_{n-1} + \dots)) \quad (26)$$

$$= x_n(1 + \delta_n) + x_{n-1}(1 + \delta_n)(1 + \delta_{n-1}) + \dots \quad (27)$$

$$= x_n \prod_{j=n}^n (1 + \delta_j) + x_{n-1} \prod_{j=n-1}^n (1 + \delta_j) + \dots \quad (28)$$

$$= \sum_{i=1}^n x_i \prod_{j=i}^n (1 + \delta_j) \quad (29)$$

Recall that $|\delta_i| < u$ where typically $u \ll 1$. Thus, we can approximate $\prod_j (1 + \delta_j)$ by ignoring degree-2 terms or above. In other words, $\prod_j (1 + \delta_j) \approx 1 + \sum_j \delta_j$. This simplifies our analysis.

$$\hat{S}_n = \sum_{i=1}^n x_i \prod_{j=i}^n (1 + \delta_j) \quad (30)$$

$$\approx \sum_{i=1}^n x_i \left(1 + \sum_{j=i}^n \delta_j \right) \quad (31)$$

Now, let us write an expression for error where $S_i = \sum_{j=i}^n x_j$:

$$S - \hat{S} \approx \sum_{i=1}^n x_i - \sum_{i=1}^n x_i \left(1 + \sum_{j=i}^n \delta_j \right) \quad (32)$$

$$= - \sum_{i=1}^n x_i \sum_{j=i}^n \delta_j \quad (33)$$

$$= - \sum_{i=1}^n \delta_i \sum_{j=1}^i x_j \quad (34)$$

$$= - \sum_i^n \delta_i S_i \quad (35)$$

We may also solely consider the magnitude of the error.

$$|S - \hat{S}| \approx \left| \sum_i^n \delta_i S_i \right| \quad (36)$$

$$\leq \sum_i^n |\delta_i S_i| \quad (37)$$

$$\leq u \sum_i^n |S_i| \quad (38)$$

Finally, we may further expand this bound, permitting more detailed analysis.

$$|S - \hat{S}| \lesssim u \sum_i^n |S_i| \quad (39)$$

$$\leq u \sum_i^n \sum_j^i |x_j| \quad (40)$$

$$= u \sum_i^n (n - i + 1) |x_i| \quad (41)$$

C.2.3 Best, Worst, and Expected Ordering

Best/Worst Case Ordering. Let us consider orderings of $x_1, \dots, x_n$ which maximize and minimize error.

Now let us we consider the case where all x_i share the same sign (e.g. are all positive or all negative). We first establish a swap lemma, then use it to prove that descending and ascending orders are extremal.

Lemma C.4 (Single-sign swap). *For some $x_1, \dots, x_n$ where $x_i > 0$ which is not sorted in descending order, pick j, k such that $j > k$ and $x_j > x_k$. Swapping x_j and x_k in position will increase our upper-bound for $|S - \hat{S}|$.*

Proof. First, let us expand the upper bound for $|S - \hat{S}|$.

$$|S - \hat{S}| \leq \sum_{i=1}^n |x_i|(n-i+1)u \quad (42)$$

$$= x_j(n-j+1)u + x_k(n-k+1)u + \sum_{\substack{i=1 \\ i \neq j,k}}^n x_i(n-i+1)u \quad (43)$$

If we swap the positions of value j and k , our bound will become:

$$|S - \hat{S}'| \leq x_j(n-k+1)u + x_k(n-j+1)u + \sum_{\substack{i=1 \\ i \neq j,k}}^n x_i(n-i+1)u \quad (44)$$

To show that this bound is greater, we must show that the following is true.

$$x_j(n-j+1)u + x_k(n-k+1)u < x_j(n-k+1)u + x_k(n-j+1)u \quad (45)$$

To do so, let us first make the following definitions:

$$\Delta_j = x_j(n-k+1)u - x_j(n-j+1)u \quad (46)$$

$$= x_j(j-k)u \quad (47)$$

$$\Delta_k = x_k(n-j+1)u - x_k(n-k+1)u \quad (48)$$

$$= x_k(k-j)u \quad (49)$$

From this we can derive the following.

$$\Delta_j + \Delta_k = x_j(j-k)u + x_k(k-j)u \quad (50)$$

$$= (x_j - x_k)(j-k)u \quad (51)$$

Since $j-k > 0$ and $x_j > x_k$, this implies that $\Delta_j + \Delta_k > 0$ as required, demonstrating 45 to be true. $\square$

Theorem C.5 (Single-sign extremal orderings). *Sorting $x_1, \dots, x_n$ by $|x_i|$ in descending order maximizes the upper bound for $|S - \hat{S}|$. Sorting in ascending order minimizes it.*

Proof. Given Lemma C.4, this follows directly. For any ordering $x_1, \dots, x_n$ which is not in descending order, we can sequentially make swaps to increase the upper bound until our ordering is in descending order.

Moreover, a similar argument enables induction the other way. By making swaps which make the ordering more ascending rather than descending, the error upper bound decreases. We omit this proof for brevity. $\square$

Expected Ordering. Now that we have constructed the best and worst case bounds, let us consider the expected error bound when there is random ordering.

Proposition C.6 (Expected error bound under random ordering). *Under a uniformly random permutation π ,*

$$\mathbb{E}[|S - \hat{S}|] \leq \frac{u(n+1)}{2} \sum_{i=1}^n |x_i|. \quad (52)$$

Proof. First, let us expand our existing upper bound for $|S - \hat{S}|$.

$$\mathbb{E}[|S - \hat{S}|] \leq \mathbb{E} \left[\sum_{i=1}^n |x_{\pi(i)}| (n - i + 1) u \right] \quad (53)$$

$$= \sum_{\pi} \frac{1}{n!} \sum_{i=1}^n |x_{\pi(i)}| (n - i + 1) u. \quad (54)$$

We know that there exists $n!$ permutations, which π is uniformly and randomly drawn from. Thus each $|x_i|$ appears in each position exactly $(n - 1)!$ times across all $n!$ permutations. Thus we can consider the overall contribution of $|x_i|$ to the expected bound:

$$\frac{(n - 1)!}{n!} \cdot u \cdot |x_i| \sum_{k=1}^n k = \frac{1}{n} \cdot u \cdot |x_i| \cdot \frac{n(n + 1)}{2} \quad (55)$$

$$= \frac{u \cdot |x_i| \cdot (n + 1)}{2}. \quad (56)$$

We have effectively swapped the order of summation in Equation 54. Summing over all i gives the result.

$$\sum_{i=1}^n \frac{u \cdot |x_i| \cdot (n + 1)}{2} = \frac{u(n + 1)}{2} \sum_{i=1}^n |x_i| \quad (57)$$

□

Now let us compute the variance of this bound. Define $w_i = (n - i + 1)$. We thus refer to our upper bound as $B(\pi) = u \sum_{i=1}^n w_i |x_{\pi(i)}|$.

Proposition C.7 (Variance of error bound under random ordering). *Under a uniformly random permutation π , the variance and standard deviation of the bound $B(\pi) = u \sum_{i=1}^n w_i |x_{\pi(i)}|$ are*

$$\text{Var}_{\pi}(B) = \frac{u^2 \sigma_{|x|}^2 n^2 (n + 1)}{12}, \quad (58)$$

$$\sigma_B = \frac{u \sigma_{|x|} n \sqrt{n + 1}}{2\sqrt{3}}, \quad (59)$$

where $\sigma_{|x|}^2 = \frac{1}{n} \sum_{i=1}^n (|x_i| - \overline{|x|})^2$ and $\overline{|x|} = \frac{1}{n} \sum_{i=1}^n |x_i|$.

Proof. For a uniformly random permutation π , we know the following standard identities:

$$\text{Var}(|x_{\pi(i)}|) = \sigma_{|x|}^2, \quad (60)$$

$$\text{Cov}(|x_{\pi(i)}|, |x_{\pi(j)}|) = -\frac{\sigma_{|x|}^2}{n - 1} \quad (i \neq j). \quad (61)$$

Expanding the variance of a weighted sum under random permutation:

$$\begin{aligned}
\text{Var}_\pi(B) &= u^2 \left[\sigma_{|x|}^2 \sum_{i=1}^n w_i^2 - \frac{\sigma_{|x|}^2}{n-1} \sum_{\substack{i,j=1 \\ i \neq j}}^n w_i w_j \right] \\
&= u^2 \sigma_{|x|}^2 \left[\sum_{i=1}^n w_i^2 - \frac{1}{n-1} \left(\left(\sum_{i=1}^n w_i \right)^2 - \sum_{i=1}^n w_i^2 \right) \right] \\
&= \frac{u^2 \sigma_{|x|}^2}{n-1} \left[n \sum_{i=1}^n w_i^2 - \left(\sum_{i=1}^n w_i \right)^2 \right]. \tag{62}
\end{aligned}$$

Substituting $\sum_{i=1}^n w_i = \frac{n(n+1)}{2}$ and $\sum_{i=1}^n w_i^2 = \frac{n(n+1)(2n+1)}{6}$, the bracketed expression simplifies:

$$\begin{aligned}
n \cdot \frac{n(n+1)(2n+1)}{6} - \frac{n^2(n+1)^2}{4} &= \frac{n^2(n+1)}{12} [2(2n+1) - 3(n+1)] \\
&= \frac{n^2(n+1)(n-1)}{12}. \tag{63}
\end{aligned}$$

Dividing by $(n-1)$ and simplifying gives the result. $\square$

C.2.4 Empirical Results

To better understand our derived bounds, we empirically compute the error incurred by sequential summation in a pretrained model using $m = 4$ partitions. We pass 32 sequences of 2048 tokens each from Wikitext through this model. At each CYLON layer, we compute the error incurred by bf16 representation under all $4! = 24$ possible orderings. In Figure 10 we present histograms of relative error per each layer. We find that although relative error exceeds 10% in some cases, the majority of reductions (99%) fall under 3% relative error.

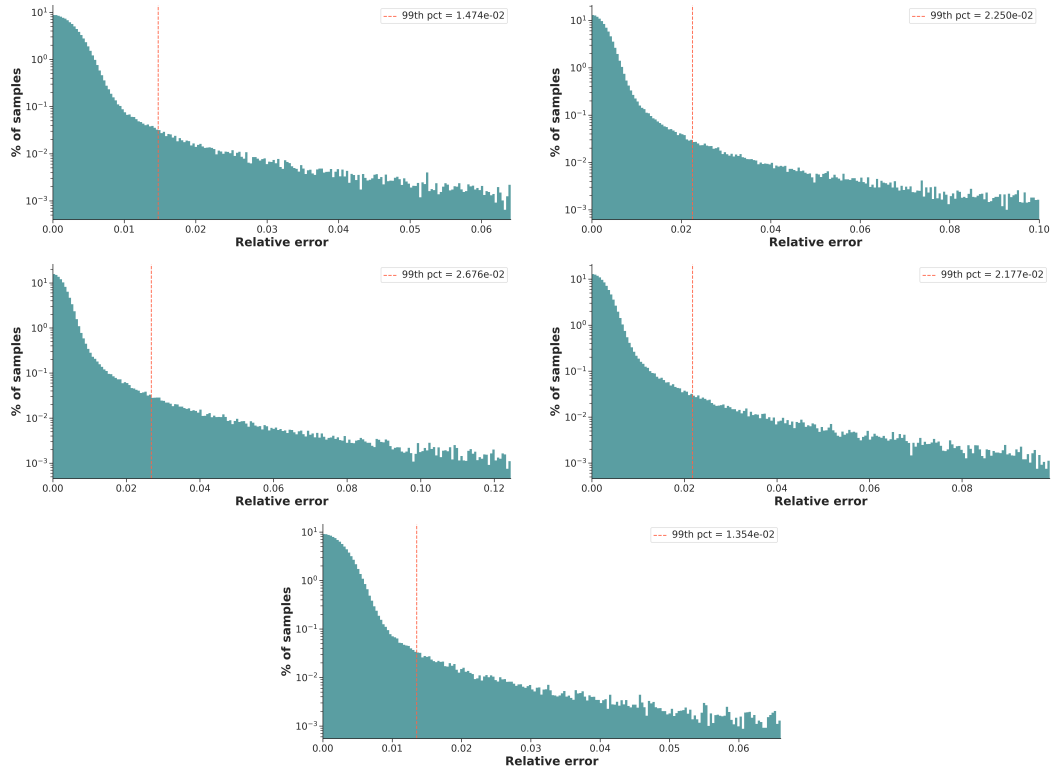


Figure 10: Distribution of bf16 summation-order error relative to fp64 reference, across all $4! = 24$ feature map orderings. The dashed line marks the 99th percentile of relative error.