GLIDE: GUIDED LAYERWISE HYBRID ATTENTION FOR EFFICIENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

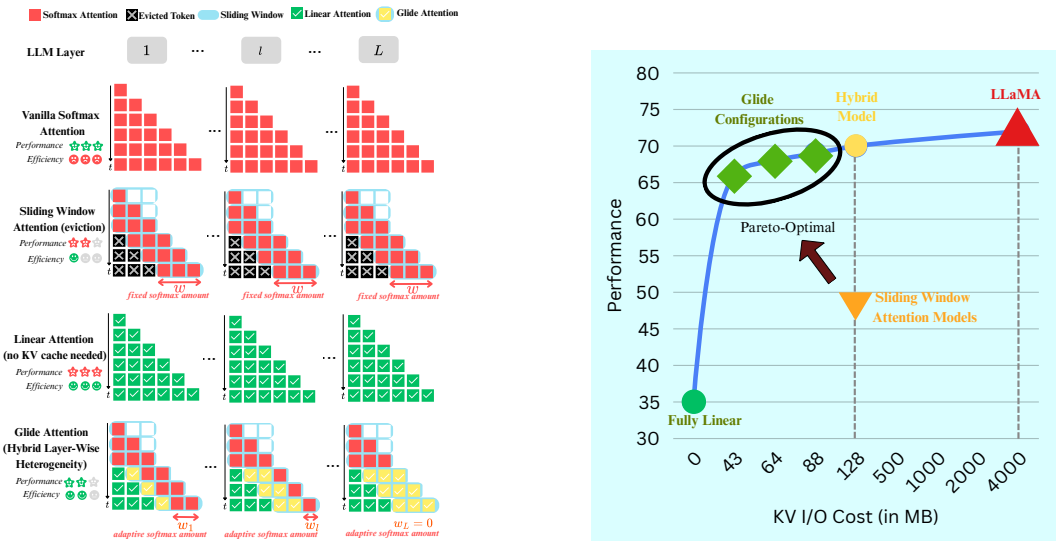
As Large Language Models scale to increasingly long contexts, the memory I/O and computational overhead of the Key-Value (KV) cache during decoding emerges as the primary throughput bottleneck. To address this, we propose **GLIDE**, a Guided Layerwise Hybrid Attention that strategically integrates sliding-window softmax attention with linear recurrent aggregation. **GLIDE** is motivated by layer-wise heterogeneity: early layers exhibit high sensitivity to softmax removal, while deeper layers demonstrate redundancy and tolerate aggressive replacement by linear alternatives. Leveraging this insight, **GLIDE** introduces a layer-wise adaptive mechanism wherein each layer is balanced with fully variable configurations between efficient linear recurrence and full-accuracy softmax. Employing front-loading softmax retention strategies, **GLIDE** non-uniformly compresses the softmax allocation in the model, reducing *aggregate KV cache I/O* while retaining model accuracy in sensitive layers. We demonstrate **GLIDE**'s framework adaptability and superior performance-efficiency tradeoffs with multiple linear and hybrid attention models, reducing end-to-end latency for long-context generation without compromising quality.

1 INTRODUCTION

Long-context Large Language Model (LLM) inference is increasingly limited not by compute alone, but by data movement during autoregressive decoding. As the KV-cache grows with sequence length, each generated token demands retrieval of an expanding volume of cached key-value states, shifting execution from compute-bound to memory-bandwidth-bound regimes (Feng et al., 2024). This bottleneck is particularly acute in long-context generation, where KV-cache growth inflates both memory footprint and off-chip data transfer, degrading hardware utilization and overall throughput. Such constraints pose a critical barrier to scalable deployment on memory-constrained accelerators and in latency-sensitive serving environments (LI et al., 2025). This motivates a central question: *How can long-context decoding achieve lower latency and reduced memory I/O while maintaining generation quality?*

Recent work has approached efficient long-context inference by optimizing KV-cache management through two main strategies: a) *Eviction-based* methods reduce memory and compute by selectively discarding cached tokens deemed less relevant (Zhang et al., 2023; Cai et al., 2024; Li et al., 2024), while b) *Retention-based* methods preserve contextual information through hybrid attention mechanisms that combine local softmax attention with efficient recurrent-style aggregation. Retention-based hybrid architectures such as LoLCats (Zhang et al., 2025), Liger (Lan et al., 2025) and Lizard (Nguyen et al., 2026) (see Section 2) offer a particularly promising design point: they apply parameter-efficient fine-tuning (PEFT) to distill pretrained softmax attention into linear recurrent forms, improving decoding efficiency without explicitly evicting context from the KV-cache. However, existing retention-based methods apply a uniform hybridization policy across all layers, replacing softmax attention with linear alternatives at fixed positions throughout the network. This one-size-fits-all strategy overlooks a critical observation: attention layers vary significantly in their sensitivity to linearization. Early layers with foundational token representations may rely more heavily on precise softmax attention, while deeper layers operating on increasingly abstract features may tolerate aggressive approximation with minimal performance degradation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) **Various Attention Mechanisms** (b) **Performance-Efficiency Pareto Analysis**

Figure 1: **Layer-Wise Adaptive Attention Strategy of GLIDE. (a) Attention Mechanisms: GLIDE** strategically transitions from softmax attention (red) in early layers to linear attention (green) in deeper layers, concentrating expensive computation where most critical, unlike vanilla softmax, sliding window attention (SWA), or pure linear approaches. **(b) Pareto Analysis: GLIDE** configurations achieve $45\times - 62\times$ lower KV cache I/O than baseline LLaMA while retaining $92\% - 96\%$ accuracy, outperforming uniform alternatives (SWA, Hybrid Model) by exploiting layer-wise heterogeneity.

Can a non-uniform, layer-wise hybrid attention policy guided by depth-dependent sensitivity achieve a superior efficiency–accuracy Pareto frontier compared to uniform hybridization?

As a first step towards answering this question, we conduct an empirical study in which we systematically vary the ratio of softmax-to-linear attention across transformer layers and measure the resulting impact on downstream task accuracy. Our analysis reveals a clear layerwise heterogeneity in softmax importance: earlier layers responsible for establishing low-level token representations and local syntactic structure benefit significantly from retaining softmax-based attention, whereas deeper layers, which operate on more abstract semantic features, tolerate aggressive linearization with minimal impact on generation quality. This finding suggests that uniform softmax allocation across layers is fundamentally suboptimal, leaving substantial efficiency gains unrealized. Building on this insight, we introduce **GLIDE** (**G**uided **L**ayerwise Hybr**i**d Attention for Efficient **D**ecoding), a retention-based architecture that allocates softmax attention non-uniformly across layers according to their measured sensitivity.

As illustrated in Fig. 1, **GLIDE** exploits layer-wise sensitivity patterns to achieve superior efficiency-accuracy trade-offs compared to various attention schemes. Fig. 1(a) contrasts four attention mechanisms: vanilla softmax attention (red) applies full quadratic attention uniformly across all layers, achieving optimal performance but incurring prohibitive memory costs; sliding window attention (red with evicted tokens) reduces memory by discarding distant context, sacrificing accuracy; linear attention (green) eliminates KV cache overhead entirely but suffers severe degradation; and **GLIDE** strategically transitions from softmax attention in early layers to linear attention in deeper layers, concentrating expensive computation where most critical. This layer-wise allocation directly reduces KV cache memory traffic during autoregressive decoding, the dominant bottleneck in long-context generation, while preserving representational fidelity in sensitive layers. Fig. 1(b) demonstrates the resulting Pareto frontier: **GLIDE** configurations occupy the favorable top-left region, achieving $45\times - 62\times$ lower KV cache I/O than baseline while retaining $92\% - 96\%$ accuracy, substantially outperforming uniform alternatives including hybrid models and sliding window attention which fail to exploit layer-wise heterogeneity. By enabling strategic allocation of softmax versus linear attention across depth, **GLIDE** unlocks previously inaccessible operating points that balance memory bandwidth, latency, and task performance. Our main contributions are as follows:

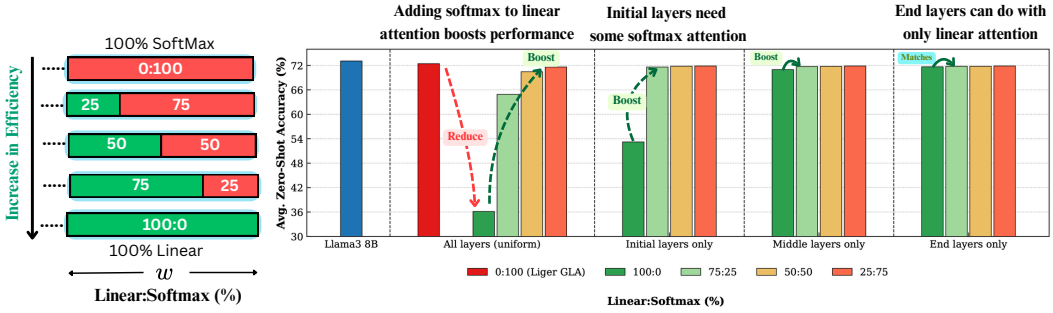
- We conduct a systematic study of hybrid attention sensitivity across transformer depth, revealing that softmax attention importance is strongly layer-dependent: early layers are markedly more sensitive to linearization, with full linearization causing catastrophic accuracy collapse (36% avg.), while late layers tolerate complete linearization with minimal degradation (Section 3.1).
- We introduce **GLIDE**, a guided layerwise hybrid attention framework that allocates softmax attention non-uniformly across transformer blocks based on observed layer-wise sensitivity. GLIDE partitions the network into early, middle, and late blocks, assigning full softmax ($\delta = 0$) to early layers, partial linearization ($\delta = \alpha \cdot w$, $\alpha \in [0, 1]$) to middle layers, and full linearization ($\delta = w$) to late layers. This reduces the δ-configuration search space to a single scalar α that continuously interpolates between full softmax and full linearization (Section 3.2).
- We demonstrate that **GLIDE** integrates seamlessly with existing retention-based architectures (Liger, LoLCats, Lizard), achieving up to **62**$\times$ reduction in KV-cache I/O and **3.3**$\times$ decoding speedup while retaining 92–94% of baseline accuracy across six reasoning benchmarks on Llama-3-8B and Mistral-7B (Section 4).

2 BACKGROUND

Policies of Sliding Window Attention. Sliding Window Attention (SWA) (Beltagy et al., 2020) restricts $\mathbf{q}_i$ to the w most recent key–value pairs, reducing the per-token memory and compute complexity to $O(w)$ with $w < n$. *However, SWA follows a first-in-first-out eviction policy, whereby older tokens are discarded as new ones enter the window, potentially removing semantically important context and degrading recall over long-range dependencies.* Prior work addressing this limitation falls into two families. *Eviction-based* policies keep the cache bounded and spend the budget where it matters, under a fixed budget in SnapKV (Li et al., 2024), H_2O (Zhang et al., 2023) and StreamingLLM (Xiao et al., 2024), or adaptively across heads and layers in AdaKV (Feng et al., 2024), MSWA (Xu et al., 2025) and PyramidKV (Cai et al., 2024). *Although saliency-based eviction mitigates the limitations of recency-based policies, it still discards cached entries and incurs noticeable performance degradation. Retention-based* policies absorb the evicted tokens into a constant-size linear recurrent state, as in LESS (Dong et al., 2024), BASED (Arora et al., 2024), Infini-Transformer (Munkhdalai et al., 2024) and EdgeInfinite (Chen et al., 2025). *Despite the performance enhancement via a constant recurrent state, these policies often require extensive re-training.* Appendix C states the SWA formulation and discusses both families method by method.

Parameter-Efficient Hybrid Architecture. The inefficiency introduced by the complex training procedures in prior hybrid architectures (Arora et al., 2024; Dong et al., 2024) can be mitigated by adopting parameter-efficient fine-tuning (PEFT) strategies. In particular, Low-Rank Adaptation (LoRA) (Hu et al., 2022) enables efficient adaptation by injecting trainable low-rank matrices into frozen pretrained weights, dramatically reducing memory and training cost. LoLCats (Zhang et al., 2025) employs this LoRA-based approach to distill pretrained softmax attention into learnable ϕ-kernel linear attention, while Liger (Lan et al., 2025) applies the same strategy for non-learnable ϕ-kernel linear attention, establishing a practical path for hybrid architecture adaptation without full retraining. However, despite prior works on adaptive KV-cache budgeting (Feng et al., 2024; Xu et al., 2025; Cai et al., 2024), we observe that most hybrid architectures still employ a uniform hybridization strategy across layers. *This raises a fundamental question: is uniform layer-wise hybridization optimal, or can non-uniform δ-allocation exploit depth-dependent linearization tolerance?*

Hybrid Attention Failure Modes. Hybrid architectures pairing a local softmax window with a linear recurrent state degrade at long contexts via two mechanisms. First, accumulating all key–value pairs into a fixed $d \times d$ recurrent state $\mathbf{S}_i$ dilutes individual token representations as sequence length grows, an effect termed *information dilution* (Lu et al., 2025). Second, because the linear and softmax terms lack a shared partition function, probability instability compounds across layers. Lizard (Nguyen et al., 2026) mitigates both by introducing a learnable attention sink for unified normalization and a linear gate (Yang et al., 2024) that overwrites (rather than purely accumulates) recurrent entries to preserve KV distinguishability. Related works alternatively interleave sparse full-softmax layers to recover recall at a minor KV cache cost. *Because these architectural interventions are orthogonal to layer-wise intra-window ratios,* **GLIDE***'s δ-allocation seamlessly integrates atop these backbones.*



(a) **GLIDE Configuration** (b) **All Layers/Isolated Sensitivity Pair-wise Analysis**

Figure 2: **Analysis of Zero-Shot Accuracy Under Varying Softmax Attention Intensity (Non Fine-Tuned).** (a) The δ parameter controls the Linear:Softmax ratio within each layer's sliding window w, trading efficiency for representational fidelity. (b) Layer-wise sensitivity analysis across Early, Middle, and Late layer groups, evaluated on LM-Eval benchmarks (PiQA, ARC-e, ARC-c, HellaSwag, WinoG) with $w = 64$ and $d = 4096$. GLA (Gated Linear Attention) replaces softmax within the window using a fixed-size recurrent state, preserving context beyond the sliding window; the red bar ($\delta = w$) denotes full linearization.

3 METHODOLOGY

3.1 MOTIVATING OBSERVATIONS

While selective softmax augmentation mitigates quality loss in linearized attention (Zhang et al., 2025), its optimal allocation remains unsolved. Balancing this trade-off, where excessive softmax reintroduces KV cache bottlenecks and insufficient allocation degrades expressivity, raises a fundamental question: *What is the minimal softmax budget required to preserve downstream performance, and is this requirement depth-dependent?*

Key Observations

1. **Depth-Dependent Sensitivity:** Early layers require full softmax, while deeper layers tolerate aggressive linearization without accuracy degradation.
2. **Linear Memory Scaling:** Replacing softmax with linear attention yields a proportional reduction in per-token KV-cache memory bandwidth.
3. **Latency Acceleration:** Mitigating I/O overhead via selective linearization bypasses the memory wall, delivering substantial inter-token decode speedups.

Accuracy Recovery from Linearization. We analyze zero-shot accuracy in a fully linearized LLaMA-3 8B Liger model (Lan et al., 2025) as softmax attention is progressively reintroduced without fine-tuning. Accuracy recovers from 36% (fully linear) to 72% (fully softmax) but exhibits diminishing returns (Fig. 2a-b), indicating a sparse softmax budget suffices (Mistral backbone results show similar generalizability; Appendix C). To determine if this budget can be allocated selectively, we hypothesize that Transformers exhibit depth-dependent sensitivity to linearization. Testing a pairwise isolated linearization setup, linearizing specific layer pairs while keeping others as softmax, confirms this: linearizing early layers $\{1, 2\}$ causes a catastrophic accuracy drop to 36%, whereas linearizing middle $\{16, 17\}$ or late $\{31, 32\}$ layers matches the full-softmax baseline (Fig. 2b). This depth-dependent tolerance forms the foundation of **GLIDE**, which allocates softmax to early-stage bottlenecks and maximizes linearization in deeper layers to reduce KV-cache overhead while preserving model utility.

Low KV Cache I/O with Reduced FLOPs. We quantify the memory efficiency of hybrid attention by varying the uniform ratio of linear to softmax layers. Because linear layers maintain constant-size recurrent states, per-token KV-cache I/O scales linearly with the proportion of softmax layers (Fig. 4). At a 32,000-token context length, the LLaMA-3 8B full-softmax baseline (0:100 linear:softmax) requires $\sim$4000 MB per token.

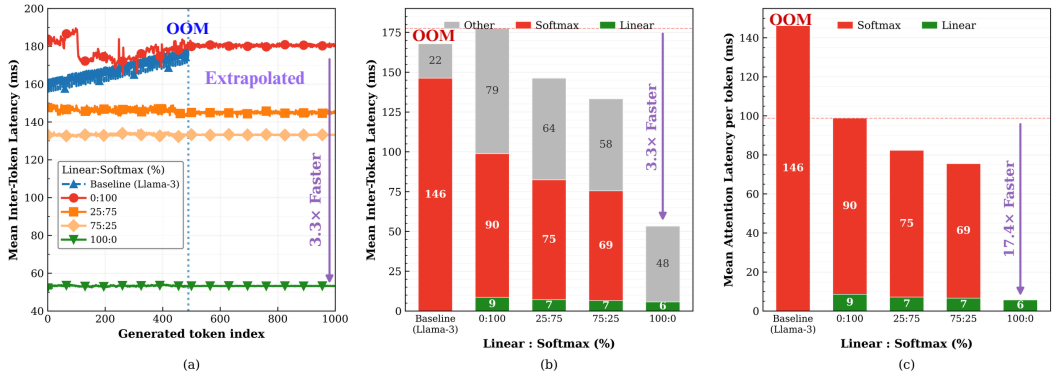


Figure 3: **Attention Latency Analysis Across Linearization Ratios and Baseline.** Profiling Llama-3-8B decoding from 20K-token prefill across uniform linear:softmax ratios with the base window size $w = 20$K and model dimension $d = 4096$. (a) Baseline encounters OOM mid-sequence while GLIDE configurations complete decoding with stable latency, achieving $3.3\times$ speedup at full linearization. (b) Operator breakdown: softmax (red) dominates baseline cost and triggers OOM at 3K tokens; increasing linear ratio eliminates this bottleneck while non-attention overhead (gray) remains constant. (c) Attention-only latency: linear layers contribute $\sim$7ms regardless of sequence length, yielding $17.4\times$ reduction versus softmax.

Increasing linearization directly mitigates this I/O bottleneck: a 25:75 configuration reduces I/O to 1875 MB, a 75:25 ratio achieves a $4\times$ reduction (625 MB), and a fully linear (100:0) model eliminates historical context I/O entirely. This strictly linear memory scaling reinforces the value of our depth-aware allocation strategy to maximize I/O savings while preserving task accuracy.

Improved Latency. Reduced I/O directly accelerates inference (Fig. 3). Mean inter-token decode latency drops from $\sim$180ms in the full-softmax baseline to $\sim$55ms in the linearized setup, achieving a $3.3\times$ speedup. Breaking down these costs confirms that while linear attention latency remains a constant 7–9ms, softmax overhead dominates execution time. By confining softmax to early-stage bottleneck layers and linearizing deeper redundant layers, **GLIDE** bypasses this memory wall, enabling extended context processing with substantially lower per-token FLOPs and decode latency.

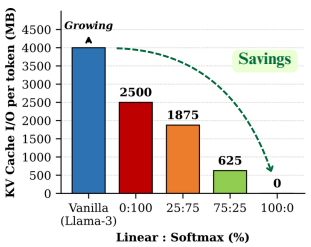


Figure 4: **KV Cache I/O Scales Linearly with Softmax Layer Proportion.** Per-token KV cache traffic for Llama-3-8B at $32,000^{th}$ token generation for the window size $w = 20$k and the model dimension $d = 4096$.

3.2 GLIDE ATTENTION

GLIDE applies linear attention to a δ-sized region inside the sliding window w. A single scalar therefore sets how far each layer is linearized, and Observation 1 licenses raising it with depth. Tokens evicted from the window are absorbed into the linear recurrent memory, which lets the model process extended contexts under a bounded memory footprint. The mechanism is architecture-agnostic. It composes with Taylor-based kernels (Arora et al., 2024; Zhang et al., 2024) and with GLA-style gating (Lan et al., 2025), and it runs on any non-learnable ϕ-kernel, so no additional fine-tuning or learned projection is needed and pretrained transformer weights remain usable as they are. *Given a query q_i at the instance i, the output of* **GLIDE** *attention can be defined as,*

$$O_i^{\text{Glide}} = O_{i-w-\delta}^{\text{linear}} \oplus O_{i-w-\delta+1}^{\text{SWA}}, \tag{1}$$

where the model has L layers with H heads each and per-head dimension $d = d_{\text{model}}/H$, $\mathbf{q}_i \in \mathbb{R}^d$ is the query at generation step i and $\{\mathbf{k}_j, \mathbf{v}_j\}_{j=1}^{i} \subset \mathbb{R}^d$ the cached key–value pairs, superscripts on O_i name the variant that produces it, w denotes the constant KV cache size of Sliding Window Attention (SWA), and δ is the parameter controlling selective KV cache retrieval from that window. We write δ_ℓ for the budget of layer $\ell \in [1, L]$ and reserve $\alpha \in [0, 1]$ for the cache sparsity that sets $\delta = \alpha \cdot w$, keeping it distinct from the attention score α_{ij}. The parameter δ effectively determines the degree of linearization applied at each layer: a larger δ implies greater reliance on linear attention, while $\delta = w$

corresponds to full linearization, completely bypassing KV cache retrieval. In practice, setting $\delta < w$ reduces the effective KV I/O from w to $w' = w - \delta$, enabling fine-grained control over the trade-off between memory bandwidth and attention fidelity. Further breaking down (1), the concentrated local attention computed by Sliding Window Attention (SWA) can be given as,

$$O^{\text{SWA}}_{i-w-\delta+1} = \frac{\sum_{j'=i-w-\delta+1}^{i} \exp(q_i k_{j'}^T) \cdot v_{j'}}{\sum_{j'=i-w-\delta+1}^{i} \exp(q_i k_{j'}^T)}. \tag{2}$$

Tokens evicted from the sliding window KV cache, together with the δ most recent tokens within the window, are handled via linear attention rather than full softmax attention. This design enables efficient global context modeling through a constant-size recurrent state, eliminating the need to store or retrieve explicit key-value pairs for tokens outside the active window. The linear attention component can be formally defined as,

$$O^{\text{Linear}}_{i-w-\delta} = \frac{\sum_{j'=1}^{i-w-\delta} \phi(q_i) \cdot \phi(k_{j'})^T \cdot v_{j'}}{\sum_{j'=1}^{i-w-\delta} \phi(q_i) \cdot \phi(k_{j'})^T}. \tag{3}$$

We use $\phi(\cdot) = \text{softmax}(\cdot)$ as the default kernel, which preserves spectral properties better than polynomial or RBF alternatives. **GLIDE** combines adaptive local sliding-window attention (for fine-grained interactions) with constant-memory global linear recurrence (for full prefix history). The $\oplus$ operator merges these outputs, weighted by their normalizing denominators to ensure proper probability mass allocation. This generic hybridization supports frameworks like LoLCats, Liger, and Lizard (Arora et al., 2024; Zhang et al., 2025; Lan et al., 2025; Nguyen et al., 2026) (Appendix C). It satisfies two boundary conditions: recovering standard causal softmax attention when window size $w \to \infty$, and pure linear attention when $w = 0$.

We formally define the [Linear:Softmax] ratio as the **GLIDE** δ-allocation: (i) **Baseline Hybrid.** $\boldsymbol{\delta} = \{0, \ldots, 0\}$ ($\delta_\ell = 0, \forall \ell$): 100% softmax attention; the KV cache stores only softmax states, (ii) **Baseline Linear.** $\boldsymbol{\delta} = \{w, \ldots, w\}$ ($\delta_\ell = w, \forall \ell$): 100% linear attention; the KV cache stores only compressed recurrent states, and (iii) **GLIDE** configuration denoted $(0, \alpha w, w)$. $\boldsymbol{\delta} = \{0, \ldots, 0 \mid \alpha w, \ldots, \alpha w \mid w, \ldots, w\}$. The cache sparsity parameter $\alpha \in [0, 1]$ assigns varying linear-softmax ratios to different layer blocks. A layer with $\delta_\ell = \alpha w$ processes αw tokens linearly and retrieves the remaining $(1 - \alpha)w$ for softmax, incurring I/O overhead for the latter. For example, $\alpha = 0.5$ allocates pure softmax to early layers, 50/50 mix to middle layers, and pure linear to late layers.

Sub-quadratic Efficiency. During autoregressive decoding, layer ℓ achieves a per-token time and space complexity of $\mathcal{O}((w - \delta_\ell)d + d^2)$, where d is the head dimension. The parameter $\delta_\ell \in [0, w]$ shrinks the quadratic interaction region from w to $(w - \delta_\ell)$, seamlessly interpolating between standard sliding-window ($\delta_\ell = 0$) and fully linear ($\delta_\ell = w$) attention. For an L-layer model with allocations $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_L\}$, the total per-token complexity simplifies to $\mathcal{O}\big((Lw - \sum_{\ell=1}^{L} \delta_\ell)d + Ld^2\big)$. This formulation enables explicit accuracy–efficiency trade-offs: assigning larger δ_ℓ to deeper, linearization-tolerant layers achieves near-minimal complexity while preserving precise attention in critical early bottlenecks.

3.3 SENSITIVITY-GUIDED LAYERWISE ALLOCATION

While **GLIDE** supports fine-grained layer-wise δ_ℓ allocation, fully heterogeneous configurations incur runtime overhead from frequent kernel switching and reduced operator fusion. To mitigate this, we introduce block-wise allocation, where contiguous layer groups share a common δ ($\delta_\ell = \delta_b$ for $\ell \in \mathcal{B}_b$). This coarser granularity captures the depth-dependent sensitivity observed in Section 3.1, allocating smaller δ to early bottlenecks and larger δ to deeper layers, while enabling kernel-aware pipelining and configuration reuse to improve wall-clock efficiency without degrading accuracy. To reduce the search space, we partition the model into early, middle, and late blocks (Fig. 5) using the simplified allocation strategy detailed in Algorithm 1 (Appendix B).

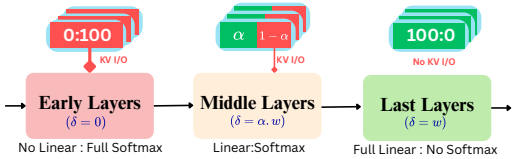


Figure 5: **Proposed δ-Allocation Strategy for GLIDE.** Motivated by Observation 1 (in Section 3.1), block-wise allocation partitions the model into early, middle, and late blocks.

Early layers are front-loaded with full softmax attention ($\delta_{b=1} = 0$) where representational fidelity is critical, while last layers use complete linearization ($\delta_{b=3} = w$) for maximum efficiency. For the middle block ($\delta_{b=2} = \alpha w$), we restrict the search space to $\delta \in \{0, 15w/16, w\}$ to explore performance–efficiency trade-offs.

Speedup. Assuming three equal blocks of $L/3$ layers, the theoretical per-token decode speedup over the hybrid baseline (**GLIDE** (0,0,0)) is:

$$S = \frac{Lwd + Ld^2}{Lwd/3 + Lw(1-\alpha)d/3 + Ld^2} = 1 + \frac{1+\alpha}{(2+3d/w) - \alpha}$$

In long-context regimes ($w \gg d$) where softmax computation dominates, this speedup approaches a $3\times$ maximum and scales directly with cache sparsity α.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate **GLIDE** on LLaMA-3-8B (Grattafiori et al., 2024) and Mistral-7B (Jiang et al., 2023). Models are fine-tuned for two epochs on 50k cleaned Alpaca samples (Taori et al., 2023) using LoRA PEFT to recover accuracy under layer linearization. Our implementation extends Liger, LoLCats, and Lizard (Lan et al., 2025; Zhang et al., 2025; Nguyen et al., 2026) using PyTorch FlexAttention with a FlashAttention backend (Dong et al., 2025). Following the sensitivity hierarchy (Fig. 2b), both $L = 32$ models are partitioned into early (layers 1–11, $\delta = 0$), middle (layers 12–21, $\delta = \alpha \cdot w$), and late (layers 22–32, $\delta = w$) blocks to balance KV cache reduction and task performance. Performance is measured using the LM Evaluation Harness (Gao et al., 2024) across six standard benchmarks, PiQA, ARC-Easy, ARC-Challenge, HellaSwag, WinoGrande, and MMLU, alongside a Needle-in-a-Haystack long-context retrieval task. Detailed hyperparameters and training configurations are provided in Appendix F.

4.2 RESULTS AND DISCUSSION

On the benchmarks and efficiency metrics described above, we compare **GLIDE** configurations against three baselines: vanilla softmax attention, hybrid attention, and pure linear attention. These results demonstrate the Pareto-optimal configurations enabled by the **GLIDE** framework.

Recovering Performance through PEFT. Zero-shot **GLIDE** configurations suffer a 5–10% accuracy drop relative to the hybrid baseline (Fig. 6a-b, distance between $(0,0,0)$ ● and zero-shot $(0,0,w)$ ■). For context, the vanilla softmax baseline (▲) peaks at ∼72% accuracy but incurs a prohibitive 4000 MB/token KV cache I/O cost, whereas the $(0,0,0)$ hybrid baseline retains 69–71% accuracy at just 128 MB/token (a $31\times$ reduction). To recover performance under aggressive linearization, we apply LoRA PEFT (◆). Fine-tuning restores 6–8 points for the $(0,0,w)$ configuration, reaching 68–70% accuracy (96% of baseline) at 88 MB/token (a $45\times$ reduction vs. vanilla). Similarly, the $(0, 15w/16, w)$ configuration yields 66–68% accuracy at just 43 MB/token (a $93\times$ reduction). *Across all allocations, fine-tuned variants consistently outperform zero-shot models at equivalent I/O levels, unlocking a rich Pareto frontier of efficiency–performance trade-offs.*

Performance Trade-offs. Table 1 details fine-tuned accuracy across six benchmarks. The vanilla softmax baseline averages 72.16% (Llama-3-8B) and 72.55% (Mistral-7B) but demands 4000 MB/token KV cache I/O. The $(0,0,0)$ hybrid baseline drops this to 128 MB/token (a $31\times$ reduction) while retaining 96–98% of baseline accuracy (69.12% and 71.31%, respectively). Progressive linearization reveals that mid-layer sparsity incurs negligible penalties once the final block is linearized. For Llama-3-8B, $(0,0,w)$ achieves 67.84% at 88 MB/token, and extending to $(0, w/2, w)$ maintains 67.73%. Aggressive linearization $(0, 15w/16, w)$ yields 66.74% at just 43 MB/token—a $93\times$ efficiency gain preserving 92% of accuracy. Mistral-7B exhibits consistent trends with stronger absolute performance, degrading gracefully by ∼1% per configuration step (69.80%, 69.44%, and 68.23% respectively). Conversely, the fully linear baseline (w, w, w) degrades catastrophically across architectures (∼34% average), with severe reasoning failures on HellaSwag (25.22%) and ARC-C (24.23%). *Overall, fine-tuned* **GLIDE** *configurations unlock 45–93$\times$ efficiency gains while preserving 90–96% of baseline accuracy.*

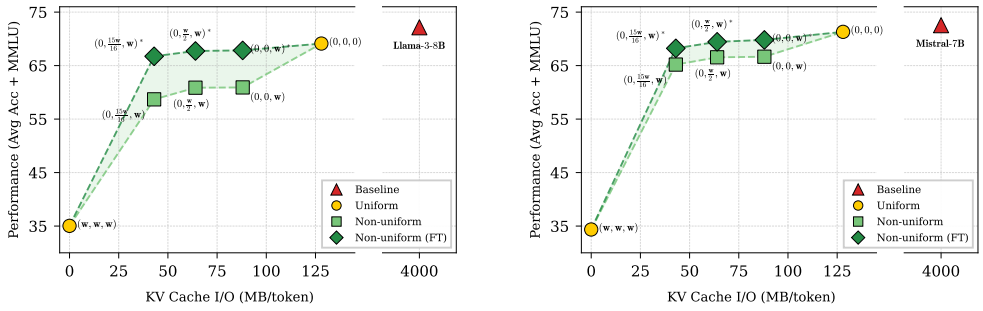
Table 1: **Fine-Tuned Accuracy Across Reasoning and Knowledge Benchmarks.** We evaluate Fine-tuned GLIDE across standard benchmarks: PiQA, ARC-Easy, ARC-Challenge, HellaSwag, and WinoGrande in zero-shot settings, with MMLU evaluated using 5-shot prompting. Each row corresponds to a block-wise δ-configuration ranging from full softmax retention $(0, 0, 0)$ to complete linearization (w, w, w) with the base window size $w = 1024$ with model dimension $d = 4096$. Avg. denotes the mean across all six tasks. Results show that moderate linearization retains competitive performance.

Model	Configuration	PiQA (↑)	ARC-E (↑)	ARC-C (↑)	H.Swag (↑)	WinoG. (↑)	MMLU (↑) (5-shot)	Avg. (↑)	KV I/O (↓) (MB/tok)
Llama-3-8B	Baseline (Vanilla SoftMax)	78.84	80.98	53.92	79.19	73.32	66.68	72.16	4000
	GLIDE $(0, 0, 0) \equiv$ Baseline (Hybrid)	80.20	79.88	51.71	77.41	71.11	54.39	69.12	128
	GLIDE $(0, 0, w)$	78.73	78.58	49.91	76.03	70.09	53.68	67.84	88
	GLIDE $(0, w/2, w)$	78.89	78.11	50.34	75.93	70.48	52.62	67.73	64
	GLIDE $(0, 15w/16, w)$	78.24	77.65	48.55	74.58	70.48	50.93	66.74	43
	GLIDE $(w, w, w) \equiv$ Baseline (Linear)	53.16	27.57	24.23	25.22	50.51	23.09	33.96	0
Mistral-7B	Baseline (Vanilla SoftMax)	80.79	80.22	54.01	81.17	75.30	63.78	72.55	4000
	GLIDE $(0, 0, 0) \equiv$ Baseline (Hybrid)	81.34	81.73	55.12	80.72	72.61	56.32	71.31	128
	GLIDE $(0, 0, w)$	80.63	79.76	51.45	78.00	73.24	55.69	69.80	88
	GLIDE $(0, w/2, w)$	80.30	80.05	51.19	77.98	73.16	53.93	69.44	64
	GLIDE $(0, 15w/16, w)$	80.41	79.97	51.28	76.08	73.16	48.48	68.23	43
	GLIDE $(w, w, w) \equiv$ Baseline (Linear)	53.26	28.49	24.15	25.92	51.14	23.09	34.34	0

Table 2: **Block-wise δ-Allocation Strategies.** End-to-end accumulated latency over sequence length under GLIDE's allocation schemes for Llama-3-8B and Mistral-7B with base window size $w = 1024$ and model dimension $d = 4096$. Configurations exploit layer-wise linearization tolerance, concentrating softmax in early layers. As δ increases, memory bandwidth decreases and inference accelerates.

Configuration	**Llama-3-8B** – E2E Latency (s)				**Mistral-7B** – E2E Latency (s)			
	4K	8K	16K	32K	4K	8K	16K	32K
Baseline (Vanilla Softmax)	264.63	765.92	–(OOM)[1]	–(OOM)[1]	305.26	1123.57	–(OOM)[1]	–(OOM)[1]
GLIDE $(0, 0, 0) \equiv$ Baseline (Hybrid)	453.68	919.66	1850.92	3705.72	707.25	1396.80	2789.58	5589.72
GLIDE $(0, 0, w)$	227.71	453.08	973.58	2638.35	724.06	1448.75	2998.59	5856.51
GLIDE $(0, w/2, w)$	223.58	446.23	966.50	2637.42	723.66	1476.85	2854.07	5770.93
GLIDE $(0, 15w/16, w)$	225.47	449.67	972.20	2637.54	679.15	1354.91	2711.12	5444.56
GLIDE $(w, w, w) \equiv$ Baseline (Linear)	205.12	409.11	818.59	1640.75	710.38	1415.44	2824.71	5652.72

[1] –(OOM) denotes Out-of-Memory; longer sequence lengths are not supported.



(a) Llama-3-8B (b) Mistral-7B

Figure 6: **Performance vs. KV Cache I/O Trade-off.** Average task accuracy versus KV cache I/O per token for (a) Llama-3-8B and (b) Mistral-7B. Markers indicate: baseline (▲), uniform (●), non-uniform (■), and non-uniform fine-tuned (◆). Fine-tuning recovers 6–8 accuracy points, achieving 96% baseline performance at 45× lower memory bandwidth while non-uniform allocations consistently outperform uniform alternatives.

End-to-End Latency. Table 2 reports 4K–32K end-to-end latencies. For Llama-3-8B, vanilla softmax OOMs beyond 8K. The $(0, 0, 0)$ hybrid baseline resolves this, scaling to 3705.72s at 32K. Progressive linearization further accelerates inference: $(0, 0, w)$ completes 4K in 227.71s (a 2.0× speedup) and 32K in 2638.35s. Crucially, mid-layer sparsity variants, $(0, 0, w)$, $(0, w/2, w)$, and $(0, 15w/16, w)$, exhibit near-identical latencies (∼2638s at 32K). This iso-latency behavior confirms the bottleneck is memory bandwidth, not compute. The primary differentiator is KV cache I/O, where these configurations require 88, 64, and 43 MB/token respectively (a 2–3× bandwidth reduction).

The pure linear baseline (w, w, w) is fastest (1640.75s at 32K) but catastrophically degrades accuracy. Mistral-7B exhibits similar iso-latency trends across configurations, which we attribute to hardware-level hybrid execution overheads (warp divergence, irregular memory access, and kernel launches). *Ultimately,* **GLIDE** *achieves 1.4–2.0× latency speedups and 2–3× bandwidth reductions while preserving 92–96% of baseline accuracy.*

KV I/O Efficient Reasoning Scalability. Scaling reasoning models to long contexts typically causes linear KV cache I/O growth, shifting inference from compute-bound to memory-bandwidth-bound. As shown in Fig. 7, the vanilla softmax baseline accumulates over 17.5 GB of KV I/O at just 8K tokens. This memory pressure severely bottlenecks throughput and limits concurrent serving capacity, as cache consumption scales linearly with sequence length and batch size. **GLIDE** mitigates this through layer-wise hybridization. At 128K tokens, the aggressive $(0, 15w/16, w)$ configuration requires only 5.2 GB of KV I/O, a 3× reduction compared to the $(0, 0, 0)$ hybrid baseline (16.3 GB). By compressing the per-request memory footprint while retaining 92–96% accuracy, **GLIDE** allows deployments to serve 2–3× more concurrent users per device. *These compounding efficiency gains reduce memory bandwidth contention and tail latency, making long-context inference viable in multi-user serving environments.*

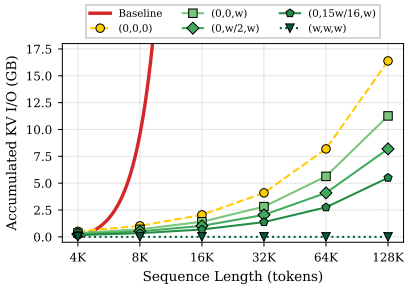


Figure 7: **Accumulated KV I/O vs. Sequence Length. GLIDE** configurations achieve sub-linear KV cache growth compared to baseline's linear scaling.

Long-Context Retrieval. We evaluate **GLIDE**'s backbone-agnostic retrieval on Liger, LoLCATS (Fig. 8), and Lizard (Fig. 10) using the 4K Needle-in-a-Haystack (NIAH) benchmark. On holdout tasks, where retrieval prompts differ syntactically but not semantically from training, baseline Liger and LoLCATS suffer complete recall collapse, highlighting the representational inflexibility of pure linear kernels. In contrast, **GLIDE** variants retain recall up to 3K tokens on holdout tasks and achieve competitive full recall on in-distribution formats. We attribute this robustness to **GLIDE**'s early-block softmax, which preserves critical token-level precision. All **GLIDE** models utilize the $(0, w/2, w)$ configuration ($w = 1024$), augmented with a vanilla softmax anchor every 7 layers and an additional anchor in the highly-sensitive early block (Observation 1). Full training details are provided in Appendix D.

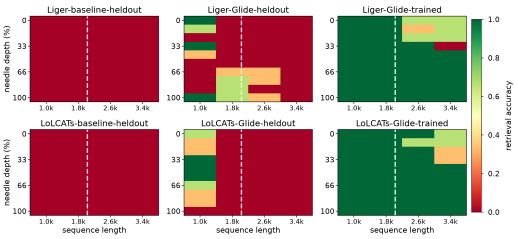


Figure 8: **Needle-in-a-Haystack Retrieval.** The dashed line marks 2K training context.

5 Conclusion

We introduce **GLIDE**, an adaptive layer-wise hybrid attention framework for efficient long-context decoding. Based on the observation that transformers exhibit depth-dependent sensitivity to linearization, **GLIDE** allocates full softmax to early bottleneck layers while replacing deeper, approximation-tolerant layers with memory-efficient linear recurrence. This strategy directly mitigates KV-cache I/O bottlenecks, achieving up to a 62× reduction in memory bandwidth while retaining 92–96% of baseline accuracy across reasoning tasks. By compressing the per-request memory footprint, **GLIDE** enables resource-constrained accelerators to support 2–3× more concurrent users, offering a practical path to scalable long-context deployment.

AI use statement

In this work, we have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses,

design or provide feedback on research methodology or experiments, implement methods, clean and reformat dataset, support qualitative and thematic data analysis, nor interpret results. Using AI to assist with translation is not applicable to this work. Additionally, we used generative AI tools for polishing writing, grammar-checking and spell-checking human-written texts, and for assistance with LaTeX formatting. We have reviewed all AI-assisted work. We have read over all AI-suggested syntax changes to ensure the final content reflects the authors' original intent and ideas. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

The method is specified in full in the main text and appendix. Sec. 3.2 gives the **GLIDE** attention formulation and Sec. 3.3 the block-wise δ-allocation strategy, with Algorithm 1 in Appendix B stating the allocation search in full; the attention and KV cache preliminaries are in Appendix A, the extended treatment of window attention in Appendix C, and the speedup derivation in Appendix E. All experiments use publicly available datasets (Alpaca-cleaned and the LM Evaluation Harness benchmarks: PiQA, ARC-Easy, ARC-Challenge, HellaSwag, WinoGrande, MMLU) and released model weights (Llama-3-8B, Mistral-7B, Llama-3.1-8B); the full protocol, the block partitioning, the δ-configuration grid, and all hyperparameters are reported in Sec. 4 and Appendix F, with the complete per-backbone training configuration in Table 3. The generation procedure for the synthetic Needle-in-a-Haystack recall dataset, including the needle template, the four task shapes and the train/validation split, is described in Appendix D. The full codebase will be publicly released with the camera-ready version.

References

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. *arXiv preprint arXiv:2402.18668*, 2024. URL https://doi.org/10.48550/arXiv.2402.18668.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://doi.org/10.48550/arXiv.2004.05150.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024. URL https://doi.org/10.48550/arXiv.2406.02069.

Jiyu Chen, Shuang Peng, Daxiong Luo, Fan Yang, Renshou Wu, Fangyuan Li, and Xiaoxin Chen. Edgeinfinite: A memory-efficient infinite-context transformer for edge devices. *arXiv preprint arXiv:2503.22196*, 2025. URL https://doi.org/10.48550/arXiv.2503.22196.

Jyotikrishna Dass, Shang Wu, Huihong Shi, Chaojian Li, Zhifan Ye, Zhongfeng Wang, and Yingyan Lin. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 415–428. IEEE, 2023. URL https://doi.org/10.48550/arXiv.2211.05109.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024. URL https://doi.org/10.48550/arXiv.2402.09398.

Juechu Dong, BOYUAN FENG, Driss Guessous, Yanbo Liang, and Horace He. Flexattention: A programming model for generating fused attention variants. In *Eighth Conference on Machine Learning and Systems*, 2025. URL https://doi.org/10.48550/arXiv.2412.05496.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024. URL https://doi.org/10.48550/arXiv.2407.11550.

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. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers,

Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://doi.org/10.48550/arXiv.2407.21783.

Dongchen Han, Yifan Pu, Zhuofan Xia, Yizeng Han, Xuran Pan, Xiu Li, Jiwen Lu, Shiji Song, and Gao Huang. Bridging the divide: Reconsidering softmax and linear attention. *Advances in Neural Information Processing Systems*, 37:79221–79245, 2024. URL https://doi.org/10.48550/arXiv.2412.06590.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://doi.org/10.48550/arXiv.2106.09685.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://doi.org/10.48550/arXiv.2310.06825.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020. URL https://doi.org/10.48550/arXiv.2006.16236.

Disen Lan, Weigao Sun, Jiaxi Hu, Jusen Du, and Yu Cheng. Liger: Linearizing large language models to gated recurrent structures. In *Forty-second International Conference on Machine Learning*, 2025. URL https://doi.org/10.48550/arXiv.2503.01496.

Haoyang LI, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole HU, Wei Dong, Li Qing, and Lei Chen. A survey on large language model acceleration based on KV cache management. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://doi.org/10.48550/arXiv.2412.19442.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024. URL https://doi.org/10.48550/arXiv.2404.14469.

Jiecheng Lu, Xu Han, Yan Sun, Viresh Pati, Yubin Kim, Siddhartha Somani, and Shihao Yang. Zeros: Zero-sum linear attention for efficient transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Ms6IXbfzzX.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101, 2024. URL https://doi.org/10.48550/arXiv.2404.07143.

Chien Van Nguyen, Huy Huu Nguyen, Ruiyi Zhang, Hanieh Deilamsalehy, Puneet Mathur, Viet Dac Lai, Haoliang Wang, Jayakumar Subramanian, Ryan A. Rossi, Trung Bui, Nikos Vlassis, Franck Dernoncourt, and Thien Huu Nguyen. Lizard: An efficient linearization framework for large language models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34935–34948, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1613. URL https://aclanthology.org/2026.acl-long.1613/.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. URL https://doi.org/10.48550/arXiv.1706.03762.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. *URL https://arxiv. org/abs/2309.17453*, 1, 2024. URL https://doi.org/10.48550/arXiv.2309.17453.

Yixing Xu, Chao Li, Dong Li, Xiao Sheng, Fan Jiang, Lu Tian, and Emad Barsoum. Qt-vit: Improving linear attention in vit with quadratic taylor expansion. *Advances in Neural Information Processing Systems*, 37:83048–83067, 2024. URL https://doi.org/10.52202/079017-2642.

Yixing Xu, Shivank Nag, Dong Li, Lu Tian, and Emad Barsoum. Mswa: Refining local attention with multi-scalewindow attention. *arXiv preprint arXiv:2501.01039*, 2025. URL https://doi.org/10.48550/arXiv.2501.01039.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training, 2024. URL https://arxiv.org/abs/2312.06635.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry, 2024b. *URL https://arxiv. org/abs/2402.04347*, 2024. URL https://doi.org/10.48550/arXiv.2402.04347.

Michael Zhang, Simran Arora, Rahul Chalamala, Benjamin Frederick Spector, Alan Wu, Krithik Ramesh, Aaryan Singhal, and Christopher Re. LoLCATs: On low-rank linearizing of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://doi.org/10.48550/arXiv.2410.10254.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023. URL https://doi.org/10.48550/arXiv.2306.14048.

A ATTENTION AND KV CACHE PRELIMINARIES

Overview of Attention. Consider the next token prediction using the current query $\mathbf{q}_i \in \mathbb{R}^d$, where H denotes the number of heads present in a single multi-head attention layer, d_{model} is the overall dimension of the model, and $d = d_{\text{model}}/H$. The attention score α_{ij} captures the similarity between the current query $\mathbf{q}_i$ and the keys $\mathbf{k}_j \in \mathbb{R}^d$, where $j \in [1, i]$ defined as,

$$\alpha_{ij} = \frac{\text{sim}(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{j'=1}^{i} \text{sim}(\mathbf{q}_i, \mathbf{k}_{j'})}, \tag{4}$$

$$O_i = \sum_{j=1}^{i} \alpha_{ij} \mathbf{v}_j, \tag{5}$$

where, $\text{sim}(\mathbf{q}_i, \mathbf{k}_j) = \exp(\mathbf{q}_i, \mathbf{k}_j^T/\sqrt{d})$ corresponds to softmax attention (Vaswani et al., 2017). The output of the attention block O_i defined in (4) is the attention-weighted aggregation of value vectors $\mathbf{v}_j \in \mathbb{R}^d$. Further, the computed output O_i is transformed linearly by dot product with the projection matrix $W_o \in \mathbb{R}^{n \times d}$, where n denotes the context length of the model. According to (4), predicting $i + 1^{th}$ token depends on the set of previously generated key & value pairs $\{ \mathbf{k}_j, \mathbf{v}_j \mid j \in [1, i] \}$ and computing those matrices at time t could require expensive $O(n^2)$ computational cost.

KV Cache. The overhead involved in computing the past key and value pairs can be mitigated by caching each generated key $\mathbf{k_i}$ and $\mathbf{v_i}$ at every iteration. KV caching involves two computational stages namely prefill and decode. *Prefill Stage:* The stage during which the model computes self-attention over all prompts, and populates the key–value (KV) cache by storing the key $\mathbf{k_i}$ and value $\mathbf{v_i}$ pairs for each token. *Decode Stage:* The stage in which the model generates tokens auto-regressively; at each step, self-attention is computed using the previously stored key–value (KV) cache, and the key and value corresponding to the newly generated token are appended to the cache. Despite minimizing redundant computes, the memory required for the KV cache scales linearly with the context length n, and at each token prediction, attention incurs an expensive $O(n)$ memory footprint.

Linear Attention. To understand how global context is preserved in retention-based SWA, we formalize linear attention, which underlies these methods. Considering (4), the similarity function $sim(\cdot)$ in softmax attention can be approximated using separable kernel functions ϕ (Katharopoulos et al., 2020), applied independently to the query $\mathbf{q}_i$ and key $\mathbf{k}_j$. Under this formulation, the approximated attention score α_{ij} can be expressed as,

$$\frac{\text{sim}(\mathbf{q}_i.\mathbf{k}_j)}{\sum_{j=1}^{i} \text{sim}(\mathbf{q}_i.\mathbf{k}_j)} \approx \frac{\phi(\mathbf{q}_i).\phi(\mathbf{k}_j)^T}{\sum_{j=1}^{i} \phi(\mathbf{q}_i).\phi(\mathbf{k}_j)^T}, \tag{6}$$

On computing O_i, the matrices can be rearranged from $\sum_{j=i}^{i} \phi(\mathbf{q}_i).\phi(\mathbf{k}_j)^T\mathbf{v}_j$ to $\phi(\mathbf{q}_i).(\sum_{j=i}^{i} \phi(\mathbf{k}_j)^T\mathbf{v}_j)$ resulting in the constant recurrent memory of $d \times d$ and it can be expressed as,

$$O_i^{\text{Linear}} = \frac{\phi(\mathbf{q}_i).(\sum_{j=1}^{i} \phi(\mathbf{k}_j)^T.\mathbf{v}_j)}{\phi(\mathbf{q}_i).\sum_{j=1}^{i} \phi(\mathbf{k}_j)^T} = \frac{\phi(\mathbf{q}_i).\mathbf{S}_i}{\phi(\mathbf{q}_i).\mathbf{Z}_i}, \tag{7}$$

where the recurrent state $\mathbf{S}_i$ is updated as $\mathbf{S}_i = \mathbf{S}_{i-1} + \phi(\mathbf{k}_j)^\top \mathbf{v}_j \in \mathbb{R}^{d \times d}$, and the corresponding normalization factor is updated as $\mathbf{Z}_i = \mathbf{Z}_{i-1} + \sum_{j=1}^{i} \phi(\mathbf{k}_j)^\top \in \mathbb{R}^d$. Unlike softmax attention, linear attention incurs only an $O(d)$ data movement cost, and its memory usage remains constant with respect to the context length n. Vanilla softmax attention can be approximated using linear attention with different choices of ϕ-kernels. Commonly used ϕ-kernels include ELU + 1 (EdgeInfinite (Chen et al., 2025), Infini-Transformer (Munkhdalai et al., 2024)), MLP-based kernels (LESS (Dong et al., 2024), Hedgehog (Zhang et al., 2024), LoLCats (Zhang et al., 2025), LOLA (Hu et al., 2022)), ReLU

Algorithm 1: Blockwise δ Allocation

Input: Model M, window w
Output: Optimized model M^*

$(B_1, B_2, B_3) \leftarrow \textsc{Split}(M, 3)$;
$\textsc{Assign}(B_1, 0)$;
$\textsc{Assign}(B_3, w)$;
$\mathcal{S} \leftarrow \{0, \frac{w}{2}, \frac{15w}{16}, w\}$;
foreach $\delta \in \mathcal{S}$ **do**
  $\textsc{Assign}(B_2, \delta)$;
$\delta^* \leftarrow \arg\max_{\delta \in \mathcal{S}} \textsc{TaskAccEval}(M_\delta)$;
$\textsc{Assign}(B_2, \delta^*)$;
return M^*;

(InLine (Han et al., 2024)), and Taylor-series expansions (ViTaLity (Dass et al., 2023)(degree-1), QT-ViT (Xu et al., 2024)(degree-2), BASED (Arora et al., 2024)(degree 2)) and Softmax (Liger (Lan et al., 2025)).

B Blockwise δ-Allocation Procedure

Algorithm 1 states the block-wise search of Section 3.3 in full. The early and last blocks are fixed at $\delta_{b=1} = 0$ and $\delta_{b=3} = w$, so the search runs over the middle block alone, across the four-point grid $\mathcal{S}$.

C Extended Study on Window Attention

Sliding Window Attention. To alleviate the linear growth in memory footprint and per-token attention cost induced by full KV caching, prior work proposes Sliding Window Attention (Beltagy et al., 2020). Under sliding window attention, self-attention for query $\mathbf{q}_i$ is restricted to the w most recent key–value pairs, and the attention score α_{ij} is computed only within this window as,

$$\alpha_{ij}^{\text{SWA}} = \frac{\exp(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{j'=i-w+1}^{i} \exp(\mathbf{q}_i, \mathbf{k}_{j'})}, \tag{8}$$

$$O_i^{\text{SWA}} = \sum_{j=i-w+1}^{i} \alpha_{ij}^{\text{SWA}} . \mathbf{v}_j, \tag{9}$$

where the attention incurs a reduced $O(w)$ memory footprint, with the window size w constrained such that $w < n$. In SWA, the initial token within the window gets evicted as a new token comes in; this first-in-first-out eviction policy could remove important tokens from the window, leading to a degradation in the model's recall rate, which measures the model's ability to retrieve, attend to, and condition its generation on relevant information from earlier context tokens.

(a) Eviction-based SWA. Eviction-based approaches compute attention scores under two categories: a fixed KV cache and an adaptive KV cache budget. Among the fixed-budget works, SnapKV (Li et al., 2024) compresses the KV cache by retaining head-specific salient tokens identified from attention patterns. H_2O (Zhang et al., 2023) dynamically manages the KV cache by preserving both recent tokens and heavy hitters with high cumulative attention scores. StreamingLLM (Xiao et al., 2024) enables unbounded sequence generation by explicitly retaining attention sink tokens. Beyond fixed-budget eviction, several works introduce adaptive KV-cache budget allocation strategies. PyramidKV (Cai et al., 2024) proposes dynamic KV-cache memory allocation across transformer layers, selectively retaining important KV pairs under a global budget constraint.

(b) Retention-based SWA. Despite advances in local context optimization, eviction-based methods inherently restrict access to global context, which can adversely affect recall. This limitation can be mitigated by preserving global context as a recurrent state via linear attention mechanisms, enabling long-range information retention under a constant memory budget. Building on this framework,

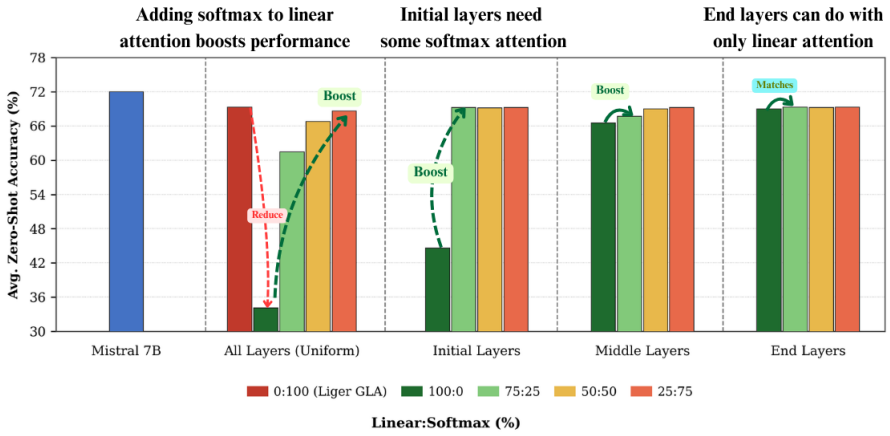


Figure 9: **Analysis of Zero-Shot Accuracy Under Varying Softmax Attention Intensity (Non Fine-Tuned).** Layer-wise sensitivity analysis: initial layers require softmax attention, while end layers tolerate full linearization. We evaluate uniform configurations $\{0,\ w/4,\ w/2,\ 3w/4,\ w\}$ with the base window size $w = 64$ and model dimension $d = 4096$ across all transformer layers using LM-Eval (PiQA, ARC-e, ARC-c, HellaSwag, WinoG), then examine recovery by applying varying softmax attention to isolated layer groups (Early, Middle, Late).

LOLA (Hu et al., 2022) introduces a sparse caching mechanism to reduce computational overhead while maintaining approximation quality. Additionally, prior work formulates attention hybridization as a two-component output combining a linear recurrent with a local softmax term over the active window. The various mathematical formulations of hybrid attention can be given by,

$$O_i^{\text{hybrid}} = \frac{\phi(\mathbf{q}_i)\,\mathbf{S}_{i-w}}{\phi(\mathbf{q}_i)\,\mathbf{Z}_{i-w}} + \frac{\sum_{j=i-w+1}^{i} \exp(\mathbf{q}_i \mathbf{k}_j^\top)\,\mathbf{v}_j}{\sum_{j'=i-w+1}^{i} \exp(\mathbf{q}_i \mathbf{k}_{j'}^\top)} \tag{10}$$

where $\mathbf{S}_{i-w} = \sum_{j=1}^{i-w} \phi(\mathbf{k}_j)^\top \mathbf{v}_j$ and $\mathbf{Z}_{i-w} = \sum_{j=1}^{i-w} \phi(\mathbf{k}_j)^\top$ are the recurrent state and normalization factor accumulating all tokens evicted from the sliding window. Further, another form of hybrid attention can be written as,

$$O_i^{\text{hybrid}} = \frac{\phi(\mathbf{q}_i)\,\mathbf{S}_{i}}{\phi(\mathbf{q}_i)\,\mathbf{Z}_{i}} + \frac{\sum_{j=i-w+1}^{i} \exp(\mathbf{q}_i \mathbf{k}_j^\top)\,\mathbf{v}_j}{\sum_{j'=i-w+1}^{i} \exp(\mathbf{q}_i \mathbf{k}_{j'}^\top)} \tag{11}$$

where, the constant recurrent state $S_i \in \mathbb{R}^{d\times d}$ includes redundant information from the prefill or the instance before the sliding window attention starts evicting the tokens. **GLIDE** adapts both forms of the hybrid attention mechanism, unified through the generic $\oplus$ operator.

D LONG-CONTEXT ADAPTATION

Long-Context Fine-Tuning. To instill long-context retrieval, we fine-tune our hybrid-linearized Llama-3.1-8B (6 full-softmax layers kept at layers 0, 3, 7, 14, 21 and 28, the remaining 26 linearized with a 1024-token sliding-window anchor) in a final recall stage on a synthetic Needle-in-a-Haystack dataset. Each example embeds RULER-style needles ("One of the special magic numbers for {key} is: {value}.") at random positions in filler text and covers the four NIAH task shapes: single, multi-key (with distractors), multi-value, and multi-query, in an even mix; every sequence is pre-tokenized to exactly 2048 tokens with the loss masked to the answer tokens only. We generate 10,000 examples (9,800 train / 200 validation), train at a 2048-token context with LoRA (rank 8) on the linearized layers, and evaluate retrieval at longer 4K and 8K contexts to measure extrapolation beyond the training length.

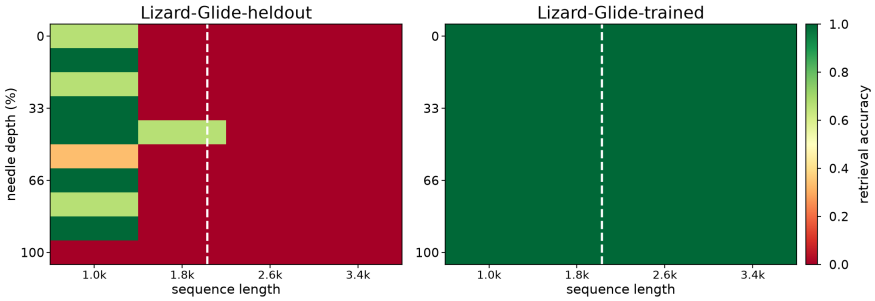


Figure 10: **Needle-in-a-Haystack Retrieval for Lizard-GLIDE.** Retrieval accuracy against needle depth and sequence length for the Lizard backbone under **GLIDE**, evaluated with the same protocol as Fig. 8. Lizard-**GLIDE** holds full recall across the trained format out to 4K, and retains partial recall on held-out needles near the training length.

Table 3: Hyperparameters for the long-context experiments.

	Liger	**LoLCats**	**Lizard**
Model			
Base model	Llama-3.1-8B (32 layers, $d = 4096$, head dim 128)		
Feature map	LoLCats–Hedgehog, softmax activation		
Attention precision	fp32 accumulation, fused-chunk linear attention		
Sliding window w	1024		
Full-softmax anchors	layers 0, 3, 7, 14, 21, 28		
Linearized layers	26		
Optimizer and Schedule			
Optimizer	AdamW		
Epochs	2		
Sequence length	2048		
Global batch size	8		
Micro batch size	1 (gradient accumulation 8)		
Validation split	200 examples		
LoRA			
Rank and alpha	$r = 8$, $\alpha = 8$		
Dropout	0.0		
Projections	W_q, W_k, W_v		
Trainable parameters	4.72M (0.059%)		
Stage 1: Attention Transfer			
Dataset	–	Alpaca-cleaned	Alpaca-cleaned
Peak learning rate	–	1.0×10^{-2}	1.0×10^{-3}
LR schedule	–	ReduceLROnPlateau	Cosine annealing
Trainable parameters	–	33.55M	33.69M
Stage 2: Autoregressive			
Dataset	Alpaca-cleaned (≈20M tokens)		
Peak learning rate	1.0×10^{-3}	1.0×10^{-4}	5.0×10^{-4}
LR schedule	ReduceLROnPlateau	ReduceLROnPlateau	Cosine annealing
Stage 3: Recall			
Dataset	NIAH-2048 (9,800 train / 200 val)		
Peak learning rate	5.0×10^{-4}		
LR schedule	ReduceLROnPlateau		

E DESCRIPTION OF GLIDE'S LAYER-WISE ALLOCATIONS

Based on our empirical study, the auto-regressive transformer architecture exhibits a heterogeneous layer-wise sensitivity and reduction in tolerance to linearization as the layers get deeper. Based on the observed behavior of LLMs towards linearization, we introduced various δ values to the different regions of LLMs, i.e., initial, middle and later layers. Additionally, we formulated the speedup

equation, which can estimate the improved speedup due to the varying ratio of amount of linearization within the sliding window attention in the hybrid attention.

The speedup in Section 3.3 is strictly decreasing in d/w, reaching $3\times$ in the long-context regime ($w \gg d$) under full cache sparsity ($\alpha \to 1$) and $1.5\times$ at $\alpha = 0$, where softmax attention is the primary bottleneck. In the short-context regime ($d \gg w$), the linear projections dominate latency and $S \to 1$ irrespective of α.

F Training Configurations

Table 3 lists the hyperparameters for all three backbones. Liger has no attention-transfer stage, as its GLA gates are pooled from $\mathbf{k}$ with no learnable feature map. Between Stage 2 and Stage 3 the adapter is merged into the base weights.

We conducted our experiments on a single NVIDIA Grace Hopper (GH200 Superchip: Grace CPU + Hopper GPU) 120GB, and we fine-tuned the model using LoRA with a rank of 8 and a scaling coefficient of 8. Training is performed for two epochs on 100K cleaned Alpaca instruction samples ($\approx$ 0.02B tokens) to support gate-recurrent linearization. All sequences are set to a maximum length of 1024 tokens in the LM eval benchmark framework. Optimization is carried out with a micro-batch size of 1, and an effective batch size of 8 is realized via gradient accumulation, consistent with the LoLCATs training setup (Zhang et al., 2025) with base window size $w = 1024$.