ELASTIC THRESHOLD ATTENTION: LEARNED CONTEXTUAL SPARSITY FOR LONG-CONTEXT DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Massive KV caches can cause severe memory-bandwidth bottlenecks during long-context decoding. Sparse attention methods mitigate this problem, but often drop necessary context, leading to quality degradation. We introduce **Elastic Threshold Attention (ETA)**, an end-to-end trainable architecture that rivals dense model quality under hardware-aligned block-sparse decoding. ETA predicts dynamic, contextual thresholds directly from query representations, adjusting context retention depending on the task at hand. To learn this policy from scratch while enabling near lossless KV cache pruning at inference time, we filter attention logits through a shifted SiLU gate during training. We show theoretically and empirically that this creates a smooth, near-uniform attention floor that neutralizes sub-threshold value contributions while simultaneously causing localized attention sinks on initial tokens to disappear. To materialize these advantages, we implement a fused inference-time kernel in Triton that screens KV blocks in $O(1)$ time using cached geometric-probabilistic bounds. Across language modeling, reasoning, and RULER benchmarks, our 1.45B ETA model matches or exceeds dense quality, outperforming alternative fast decoding methods (Quest, H_2O, NSA) while achieving higher sparsity levels. Our kernel also achieves up to $2.15\times$ end-to-end speedups over FlashAttention-2 at context lengths of up to 512K tokens. Finally, we introduce an offline calibration algorithm for domain-specific deployments that freezes per-head constant thresholds, cutting attention compute by an additional 27% at no quality cost.

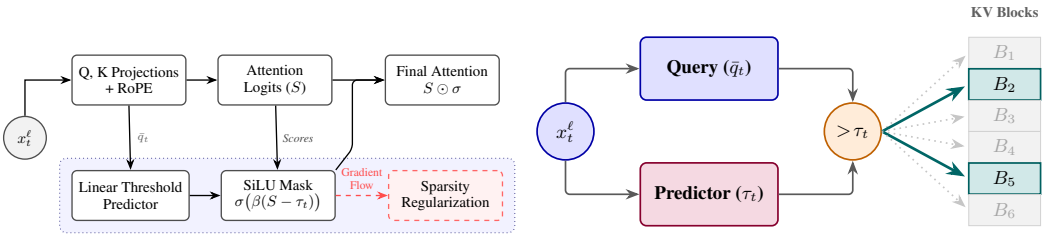


Figure 1: Overview of Elastic Threshold Attention (ETA). *Left (Training):* Queries predict a dynamic threshold τ_t that gates attention logits via a differentiable SiLU filter, compressing unselected logits toward zero rather than $-\infty$. *Right (Inference):* A custom Triton kernel evaluates geometric-probabilistic block bounds against τ_t in $\mathcal{O}(1)$ time, loading only active KV blocks from HBM.

1 INTRODUCTION

The attention mechanism is the core computational and memory bottleneck of long-context Transformers (Vaswani et al., 2017). While prefilling is compute-bound, autoregressive decoding is strictly memory-bound because generating each token requires streaming the entire Key-Value (KV) cache from High-Bandwidth Memory (HBM) to on-chip SRAM. As demand for efficient, long-context inference is greater than ever, numerous approaches have been proposed to ameliorate this issue. Most methods implement a version of selective loading, in which unimportant tokens in the KV cache are skipped and not loaded to SRAM, thus improving latency. A common problem that appears, however, is that such selectivity often trims necessary information or distorts the embedding distribution that the model was trained on, thus leading to quality 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.

In this work, we ask whether inference-time selectivity can be made naturally compatible with training-time quality in a simple, unified architecture. We introduce **Elastic Threshold Attention (ETA)**, an end-to-end trainable design that achieves hardware-accelerated decoding while matching dense model quality (Figure 1). Rather than enforcing a fixed token budget, ETA predicts a dynamic, query-conditioned score threshold τ_t via a lightweight linear projection. When a query encounters a routine token, τ_t rises to prune the cache; when a query requires retrieval, τ_t drops automatically to expand the active context window.

To learn this contextual policy from scratch without losing dense representational fidelity, ETA redefines how sub-threshold tokens are treated. Whereas prior sparse mechanisms delete unselected tokens by pushing logits to $-\infty$, we find that hard deletion is counterproductive for end-to-end training. We instead apply a context-based tempering to the distribution by passing the attention logits through a centered SiLU filter. As a result, ETA can adaptively move between two regimes: dense attention, and a sparse operating mode in which a smooth, uniform attention floor replaces the tail of sub-threshold keys.

This uniform background floor offers two important advantages: First, as we formalize in Section 2 and Subsection G.1, flattening sub-threshold weights causes background value averages to cancel at a $1/\sqrt{m}$ rate up to a positive scalar factor absorbed by $\mathrm{RMSNorm}$, rendering the representation invariant to both dropping pruned KV blocks and co-admitting sub-threshold keys inside loaded blocks. In contrast, an identical model trained with hard masking suffers a 16.6% perplexity penalty while remaining $1.84\times$ denser. Second, ETA exhibits no attention sinks as they are absorbed into the uniform probability floor. This makes ETA mechanistically more interpretable and renders it compatible with modern KV cache compression algorithms.

Our Contributions

- **Temper, not Prune & Elimination of Attention Sinks:** We show theoretically and empirically (Appendix G) that SiLU filtering in attention logit space during training creates a uniform attention floor whose sub-threshold value contributions cancel under $\mathrm{RMSNorm}$, making the model resilient to inference-time block pruning and over-inclusion while eliminating localized attention sinks on initial anchor tokens.
- **Contextual Sparsity & Domain-Specific Calibration:** We show that runtime query-conditioned thresholds preserve dense model quality, while also supporting optional offline calibration into static per-head constants for domain-specific serving.
- **Hardware-Co-Designed Decoding via Dual Probabilistic-Geometric Screening:** We develop a fused Triton decode kernel that evaluates geometric and probabilistic block bounds in $\mathcal{O}(1)$ time, eliminating KV cache memory traffic for pruned blocks.
- **Quality Retention & Decoding Acceleration:** Pre-trained at 1.45B parameters, ETA rivals dense attention and surpasses alternative efficient decoding algorithms (like Quest (Tang et al., 2024), H_2O (Zhang et al., 2023), NSA (Yuan et al., 2025)) across language modeling, reasoning, and multi-needle/2-hop retrieval benchmarks, showing up to $2.15\times$ decode speedups over FlashAttention-2.

1.1 Related Work

The literature of efficient attention methods is vast with techniques spanning both theoretical and practical considerations.

Heuristic Eviction and Windowing. Early subquadratic attention methods relied on structural heuristics like sliding windows (Child et al., 2019), random routing (Zaheer et al., 2020), or low-rank kernel approximations (Choromanski et al., 2020; Shen et al., 2021). For long-context generation, modern KV cache management methods such as H_2O (Zhang et al., 2023) and StreamingLLM (Xiao et al., 2024) evict past tokens dynamically by tracking cumulative attention scores or preserving initial sink tokens. While effective at reducing cache footprints, these heuristics decouple pruning from model training, frequently evicting non-recent information and degrading long-context reasoning.

Dynamic Routing and Block-Sparse Selectors. To bridge theoretical FLOP reductions and practical GPU acceleration (Jiang et al., 2024; Tang et al., 2024), dynamic selection methods identify salient context on the fly via k-nearest neighbor search (Roy et al., 2021; Haris, 2025), Locality

Sensitive Hashing (Chen et al., 2025), or learned projections (Desai et al., 2024). Hardware-aligned variants further structure selection into memory tiles: Quest (Tang et al., 2024) screens KV pages using min/max bounds, SpargeAttention (Zhang et al., 2025) filters blocks via compressed inner products, and SeerAttention (Gao et al., 2025) distills a 2D block gate from a dense teacher. Furthermore, Hu et al. (2022) explore differentiable score thresholding for Named Entity Recognition (NER), though their formulation relies on a tuned top-k hyperparameter, hard-masks unselected tokens to $-\infty$, and lacks a custom kernel to avoid quadratic materialization.

Trainable Sparse Attention. Recent architectures seek to incorporate sparsity natively into pre-training. HiRE (Yerram et al., 2024) explores learnable compression for low-rank attention, and DMA (Shi et al., 2025) mixes fixed masking patterns using learned gates. Native Sparse Attention (NSA) (Yuan et al., 2025) achieves notable quality by combining sliding-window, compressed-coarse, and selected-fine branches. However, NSA's multi-branch design requires substantial architectural reconfiguration and tuning. In contrast, ETA introduces a minimal, drop-in modification ($< 0.1\%$ parameter overhead) that preserves standard Transformer execution while accelerating decoding.

2 Elastic Threshold Attention

We consider a standard decoder-only Transformer with L layers and H attention heads (full preliminaries in Appendix A), where query, key, and value vectors $q_t, k_s, v_s \in \mathbb{R}^{d_h}$ at positions $s \le t \le T$ compute dense attention output

$$o_t = \sum_{s=1}^{t} A_{ts} v_s, \quad \text{with } A_{ts} = \frac{\exp(\langle q_t, k_s \rangle / \sqrt{d_h})}{\sum_{j=1}^{t} \exp(\langle q_t, k_j \rangle / \sqrt{d_h})}$$

While prefill requires $\Theta(T^2 d_h)$ FLOPs, autoregressive decoding is strictly memory-bandwidth bound because generating each token t requires streaming all cached past keys and values from GPU High-Bandwidth Memory (HBM) to SRAM at every step. Sparse attention addresses this bottleneck by restricting attention to an active index subset $\mathcal{I}_t \subseteq \{1, \dots, t\}$ with $|\mathcal{I}_t| \ll t$. Converting theoretical index sparsity into wall-clock speedups requires token selection to align with coalesced memory blocks on hardware, and the model to remain accurate when unselected context is omitted.

2.1 Threshold Prediction

In an effort to satisfy both desiderata, we propose Elastic Threshold Attention (ETA). ETA learns sparsity patterns by quantifying key relevance through query-key inner products $\langle q_t, k_s \rangle / \sqrt{d_h}$. Tokens falling below an adaptive threshold τ_t are suppressed. We predict τ_t using a per-layer linear projection on post-RoPE query vectors across all H attention heads:

$$\bar{q}_t^{\ell} = \left[\, q_{t,1}^{\ell} \, \middle\| \, \dots \, \middle\| \, q_{t,H}^{\ell} \, \right] \in \mathbb{R}^{H d_h}, \qquad \tau_t^{\ell} = \bar{q}_t^{\ell} W_{\tau}^{\ell} + b_{\tau}^{\ell} \in \mathbb{R}^{H} \tag{1}$$

where $W_{\tau}^{\ell} \in \mathbb{R}^{H d_h \times H}$ and $b_{\tau}^{\ell} \in \mathbb{R}^{H}$ are learnable parameters. Conditioning on concatenated post-RoPE queries provides two essential benefits: (1) rotary positional information is preserved, and (2) all heads within a layer coordinate jointly, allowing dynamic allocation of attention budgets across heads.

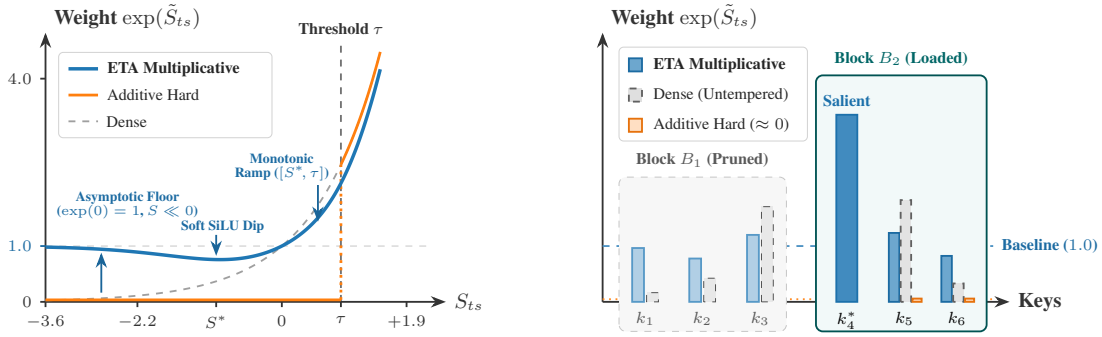
To ensure training stability and avoid premature pruning, we employ a *start-dense* initialization: projection weights are zero-initialized ($W_{\tau}^{\ell} = 0$) and biases are set to $b_{\tau}^{\ell} = -8.0$. Initially, 100% of keys pass the attention mask, enabling a smooth transition from dense to sparse attention.

2.2 Differentiable Masking and Tail Tempering

To allow gradient propagation to W_{τ}^{ℓ}, we formulate a continuous relaxation using an annealed sigmoid gate. For query t and key s at head h, let $S_{ts}^{h} = \langle q_t, k_s \rangle / \sqrt{d_h}$ denote the pre-softmax score. The soft inclusion probability $m_{ts}^{h} \in (0, 1)$ is:

$$m_{ts}^{h} = \sigma\Big(\beta \cdot (S_{ts}^{h} - \tau_{t,h}^{\ell}) \Big) \tag{2}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ and temperature $\beta \ge 1$ is annealed linearly from 1.0 to 5.0 over pre-training. Annealing provides high gradient flow early in training to escape initialization.



(a) Shifted-SiLU Logit Tempering **(b)** Block Pruning and Co-Admitted Keys

Figure 2: SiLU-gated tail tempering vs. support truncation. (a) A shifted SiLU in logit space tempers the attention distribution without dropping tokens from its support. **(b)** Across pruned blocks (B_1) and co-admitted keys (k_5, k_6) in loaded blocks (B_2), tempering compresses wide raw weight spreads (gray) into a tightly bounded band around 1.0 while preserving near-threshold order ($k_2 < k_3$ and $k_6 < k_5$).

We then apply multiplicative score gating. Let $M_{ts}^{\text{causal}} = \begin{cases} 0 & \text{if } s \le t \\ -\infty & \text{otherwise} \end{cases}$ and define:

$$A_{ts}^h = \text{softmax}\left((S_{ts}^h \odot M_{ts}) + M_{ts}^{\text{causal}}\right), \quad M_{ts} = \begin{cases} 1 & \text{if } s = t \\ m_{ts}^h & \text{otherwise} \end{cases} \tag{3}$$

Multiplicative score gating is equivalent to passing the attention logits through a shifted SiLU gate. Let $\tilde{S}(S) = S \cdot \sigma(\beta(S - \tau))$ denote the gated scores. Taking the derivative at τ,

$$\left.\frac{d\tilde{S}}{dS}\right|_{S=\tau} = M_{ts}[1 + \beta S(1 - M_{ts})]|_{S=\tau} = \frac{1}{2}\left(1 + \frac{\beta\tau}{2}\right),$$

reveals two operating regimes across our trained models:

1. **Regime 1:** $\tau \lesssim -\frac{2}{\beta}$. This is the regime ETA occupies at initialization ($b_\tau^\ell = -8$) and retains in Layer 0 after training ($\tau \in [-9.3, -4.8]$). Here the gate acts essentially as dense attention since $\tilde{S}(S) \approx S$ throughout the support of S.
2. **Regime 2:** $\tau > 0 > -\frac{2}{\beta}$. This is the regime 99.9% of heads in Layers 1–21 converge to after training (median $\tau = +2.81$). Here the mechanism acts as a *two-sided contraction toward zero* (Figure 2a): negative scores ($S < 0$) are pulled upward toward 0, while positive sub-threshold scores ($0 < S < \tau$) are damped downward below S. Solving $\frac{d\tilde{S}}{dS} = 0$ bounds the unique global minimizer S^* of $\tilde{S}$ analytically[1]:

$$-\frac{1.28}{\beta} < S^*(\beta, \tau) = -\frac{1 + W_0(e^{-1-\beta\tau})}{\beta} < -\frac{1}{\beta} < 0, \qquad \tilde{S}(S^*) = -\frac{W_0(e^{-1-\beta\tau})}{\beta} \approx 0.$$

   Consequently, $\tilde{S}$ remains strictly monotonically increasing on $[S^*, \infty)$ while the entire sub-threshold tail $(-\infty, \tau)$ is *tempered* into a low-variance, high-entropy band around $\exp(0) = 1$: normalized tail entropy rises from 0.790 to 0.916 and the max-to-mean tail weight ratio $\kappa = w_{\max}/\bar{w}$ drops from 15.6 to 2.03 (Table 7).

Invariance to block pruning and over-inclusion. At block-sparse inference (Figure 2b), the causal context $\mathcal{S}_t = \{1, \ldots, t\}$ partitions into three disjoint sets: salient keys $\mathcal{K}_t = \{s : S_{ts}^h \ge \tilde{\tau}_t\}$ (k_4^*), co-admitted sub-threshold keys $\mathcal{C}_t$ inside loaded blocks (k_5, k_6 in B_2), and dropped sub-threshold keys $\mathcal{D}_t$ inside pruned blocks (k_1–k_3 in B_1). Both training and inference evaluate the same gated weights $w_s := \exp(\tilde{S}_{ts}^h)$ on all retained tokens. For any token subset $\mathcal{X} \subseteq \mathcal{S}_t$, let $Z_\mathcal{X} := \sum_{s \in \mathcal{X}} w_s$ denote its partition sum and $\tilde{v}_\mathcal{X} := \frac{1}{Z_\mathcal{X}} \sum_{s \in \mathcal{X}} w_s v_s$ its locally normalized attention output.

[1] W_0 denotes the principal branch of the Lambert-W function, with $0 < W_0(e^{-1-\beta\tau}) < W_0(e^{-1}) \approx 0.278$ for $\tau > 0$.

Then the token-pruned output $o_t^{\text{token}} := \tilde{v}_{\mathcal{K}_t}$ (retaining $\mathcal{K}_t$), the block-sparse inference output $o_t^{\text{block}} := \tilde{v}_{\mathcal{K}_t \cup \mathcal{C}_t}$ (loading blocks $\mathcal{K}_t \sqcup \mathcal{C}_t$ while skipping $\mathcal{D}_t$), and the training output $o_t^{\text{train}} := \tilde{v}_{\mathcal{K}_t \cup \mathcal{C}_t \cup \mathcal{D}_t}$ (attending over all $\mathcal{S}_t$) satisfy the exact decomposition:

$$o_t^{\text{train}} = \frac{Z_{\mathcal{K}} o_t^{\text{token}} + Z_{\mathcal{C}} \tilde{v}_{\mathcal{C}} + Z_{\mathcal{D}} \tilde{v}_{\mathcal{D}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}, \qquad o_t^{\text{block}} = \frac{Z_{\mathcal{K}} o_t^{\text{token}} + Z_{\mathcal{C}} \tilde{v}_{\mathcal{C}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}. \tag{4}$$

Because SiLU tempering compresses the max-to-mean tail weight ratio ($\kappa = 2.03$ vs. 15.6), we show under simplifying statistical assumptions in Subsection G.1 that both tail averages $\tilde{v}_{\mathcal{C}}$ and $\tilde{v}_{\mathcal{D}}$ cancel towards zero at rate $\mathcal{O}(\sqrt{\kappa/|\mathcal{X}|})$ for $\mathcal{X} \in \{\mathcal{C}_t, \mathcal{D}_t\}$ (Proposition 1). Dropping these near-zero vectors leaves all three outputs pointing along the exact same active signal vector o_t^{token}:

$$o_t^{\text{train}} \approx \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}} o_t^{\text{token}}, \qquad o_t^{\text{block}} \approx \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}} o_t^{\text{token}}. \tag{5}$$

Since RMSNorm is invariant to positive scalar multiplication, neither dropping pruned blocks $\mathcal{D}_t$ nor co-admitting sub-threshold keys $\mathcal{C}_t$ alters the post-norm representation significantly; on the 22-layer 1.45B model, tail value norms follow the theoretical $1/\sqrt{|\mathcal{K}_t| + |\mathcal{C}_t|}$ decay and block-sparse decoding maintains a 0.9917 residual cosine similarity with training-time gating (Subsection G.2, Table 11, Table 12).

Co-admitting $\mathcal{C}_t$ actually moves o_t^{block} strictly closer to o_t^{train} than exact token pruning o_t^{token} (Proposition 2). By contrast, additive hard masking sets $Z_{\mathcal{C}} = Z_{\mathcal{D}} = 0$ during training ($o_t^{\text{train,add}} = o_t^{\text{token}}$) and puts zero gradient on tail values, so co-admitted keys $\mathcal{C}_t$ at inference push o_t^{block} away from $o_t^{\text{train,add}}$ and degrade PPL by $+16.6\%$ (Appendix C, Table 7).

Sparsity regularization. To incentivize sparsity during pre-training, we penalize the mean inclusion probability across layers L, heads H, and positions T:

$$\mathcal{L}_{\text{sparsity}} = \frac{\lambda_{\text{sparsity}}}{LHT} \sum_{\ell=1}^{L} \sum_{h=1}^{H} \sum_{t=1}^{T} \frac{1}{t} \sum_{s \le t} m_{ts}^h \tag{6}$$

Because m_{ts}^h is smooth, $\mathcal{L}_{\text{sparsity}}$ supplies direct gradients $\partial \mathcal{L}_{\text{sparsity}} / \partial \tau$, balancing predictive loss against computational budget.

2.3 Efficient Fused Training Kernel

To prevent materializing the dense $T \times T$ attention matrix in HBM, we implement a custom fused Triton kernel (Tillet et al., 2019). As a result, ETA pre-training incurs the exact same $\mathcal{O}(T^2 d)$ FLOPs, $\mathcal{O}(Td)$ HBM footprint, and $\mathcal{O}(T^2 d^2 M^{-1})$ I/O complexity as dense FlashAttention-2 (Dao et al., 2022). Forward/backward pseudocode (Algorithm 2, Algorithm 3), exact gradient derivations (Subsection E.3), and complexity analyses (Subsection E.4) are detailed in Appendix E.

3 Hardware-Aligned Inference Flow

In line with our formalization, during autoregressive decoding, ETA avoids loading non-salient KV blocks ($\mathcal{D}_t$) into SRAM. Realizing wall-clock speedups requires identifying non-significant blocks in $\mathcal{O}(1)$ time without computing individual dot products.

3.1 Block-Based Scanning via Dual Metadata Caching

To align with GPU memory layouts, we partition the KV cache into contiguous blocks of size b. A block is termed *significant* if it contains at least one token with attention score $S_{ts}^h \ge \tau_t$.

To filter non-significant blocks in $\mathcal{O}(1)$ time without loading full KV tiles from HBM, we cache a **Dual Probabilistic-Geometric Block Index** storing three summary statistics per block B: the key centroid $\boldsymbol{\mu}_B = \frac{1}{b} \sum_{k \in B} k \in \mathbb{R}^{d_h}$, the coordinate-wise variance $\boldsymbol{\sigma}_B^2 = \frac{1}{b} \sum_{k \in B} (k - \boldsymbol{\mu}_B)^{\odot 2} \in \mathbb{R}^{d_h}$, and the maximum Euclidean norm $M_B = \max_{k \in B} \|k\|_2 \in \mathbb{R}$. Given a query $q \in \mathbb{R}^{d_h}$, we estimate the maximum in-block dot product by intersecting a directional probabilistic moment bound with the exact Cauchy–Schwarz spherical norm ceiling:

$$\text{Score}_{\text{dual}}(q, B) = \frac{1}{\sqrt{d_h}} \min \left(\langle q, \boldsymbol{\mu}_B \rangle + z \sqrt{\sum_{i=1}^{d_h} q_i^2 \sigma_{B,i}^2}, \ \|q\|_2 M_B \right). \tag{7}$$

Any block where $\text{Score}_{\text{dual}}(q, B) < \tau_t$ is bypassed during HBM loading[2], reducing decode complexity from $\mathcal{O}(n \cdot d)$ to $\mathcal{O}\big((\lceil n/b \rceil + m \cdot b)d\big)$ for $m \ll \lceil n/b \rceil$ retrieved blocks. At coarse hardware tile sizes ($b = 64$), we can also optionally replace $\boldsymbol{\sigma}_B$ with a sub-block envelope standard deviation and pin the two most recent blocks to prevent variance dilution without increasing the metadata footprint (see Subsection J.1 and Appendix K).

3.2 Fused Triton Kernel Implementation with Top-1 Rescue

We implement block-sparse decoding as a custom fused Triton kernel (Tillet et al., 2019). As illustrated in Figure 6 (with full algorithmic details and pseudocode in Appendix F, Algorithm 4), the kernel streams block metadata into SRAM to evaluate $\text{Score}_{\text{dual}}(q, B)$ on the fly, skipping HBM transfers for pruned blocks. To maximize Streaming Multiprocessor (SM) occupancy, the sequence is partitioned into independent block chunks processed in parallel and merged via associative online-softmax reduction. If an aggressive threshold prunes all candidate blocks in the first chunk ($k = 1$), a localized *Top-1 Rescue Mechanism* unconditionally admits the single highest-scoring block, guaranteeing numerical stability without cross-SM synchronization (Appendix F). Because metadata is indexed per KV head (H_{KV}), the footprint scales as $\mathcal{O}(1/b)$, adding merely 1.6% memory overhead at $b = 64$ in a held-out table compatible with paged caching. Context extrapolation up to 8192 tokens ($4 \times L_{\text{train}}$) is preserved via NTK-aware RoPE scaling (bloc97, 2023).

4 Experiments

4.1 Training Dynamics and Regularization Scheduling

We train a 1.45B parameter ETA language model from scratch on FineWeb (Penedo et al., 2024) using a LLaMA-style backbone (AI@Meta, 2024). See Appendix B for more details on training. To isolate the effect of trainable thresholding, we train an identical dense baseline under matching hyperparameters and optimizer settings.

As illustrated in Figure 3, ETA converges to pretraining loss parity with the dense baseline while achieving 85% effective soft sparsity ($1 - \overline{m_{ts}^h}$) across all heads and layers. Although start-dense initialization ($b_\tau^\ell = -8$) begins in a low-gradient regime ($\sigma'(8) = 3.35 \times 10^{-4}$), Adam's scale-invariant updates combined with the β-annealing strategy drive rapid threshold migration into the score distribution within $\sim 10^2$ optimizer steps (Subsection B.1).

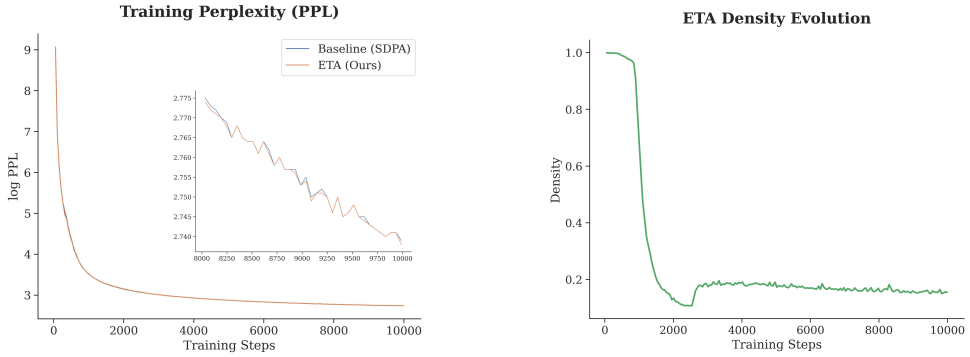


Figure 3: Training dynamics of ETA against standard dense attention. ETA achieves loss parity with dense attention while maintaining $\approx 85\%$ average soft sparsity across all heads and layers.

4.2 Inference-Time Quality and Knowledge Benchmarks

We evaluate the 1.45B model at native context length ($L = 2048$) on five standard benchmarks: WikiText-2 (Merity et al., 2016), FineWeb, and C4 (Raffel et al., 2020) for language modeling perplexity, and ARC-Easy (Clark et al., 2018) and HellaSwag (Zellers et al., 2019) for zero-shot common-sense reasoning. We compare both fine-grained ETA ($b = 4$) and our hardware-aligned Tensor Core configuration ($b = 64$) against dense attention (FlashAttention-2 / SDPA), Sliding Window Attention (SWA-256), StreamingLLM (Xiao et al., 2024) ($W, S = 256, 4$), BigBird (Zaheer et al., 2020) ($W, S, \text{Str} = 256, 4, 16$), and H_2O (Zhang et al., 2023) ($W, S, \text{HH} = 256, 4, 10\%$).

[2]In practice we often loosen the threshold by adding a slack constant α to obtain $\tilde{\tau}_t = \tau_t - \alpha$

Table 1: Language modeling perplexity and zero-shot reasoning accuracy at $L = 2048$ on the 1.45B model. Lower sub-text indicates the average fraction of past tokens attended per head. Because ARC-Easy and HellaSwag prompts (25–110 tokens) are shorter than the $W = 256$ local window of the heuristic baselines, those methods degenerate to 100% dense attention on reasoning tasks, whereas fine-grained ETA actively prunes redundant tokens.

Method Variant	WikiText-2 PPL (↓)	FineWeb PPL (↓)	C4 PPL (↓)	ARC-Easy Acc (%) (↑)	HellaSwag Acc (%) (↑)
Dense Baseline	19.35 ± 0.80 100%	16.19 ± 1.25 100%	17.49 ± 0.33 100%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%
H_2O	20.44 ± 0.80 33.7%	16.39 ± 1.29 33.4%	17.72 ± 0.69 33.1%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%
BigBird	22.54 ± 0.90 28.5%	16.81 ± 1.33 28.5%	18.04 ± 0.74 28.5%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%
StreamingLLM	23.06 ± 0.91 23.8%	16.90 ± 1.34 23.8%	18.12 ± 0.74 23.8%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%
SWA-256	25.63 ± 0.99 23.4%	18.32 ± 1.48 23.4%	19.83 ± 0.82 23.4%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%
ETA ($b = 4, z = 2.0$)	$\mathbf{19.39 \pm 0.78}$ 42.7%	$\mathbf{16.31 \pm 1.26}$ 38.2%	$\mathbf{17.56 \pm 0.33}$ 36.3%	$\mathbf{44.5\% \pm 3.51\%}$ 91.3%	$\mathbf{40.0\% \pm 3.46\%}$ 89.7%
ETA ($b = 64, a = -0.4$)	20.68 ± 0.81 55.0%	17.11 ± 1.39 50.2%	18.60 ± 0.29 48.3%	$44.0\% \pm 3.51\%$ 100%	$36.0\% \pm 3.39\%$ 100%

As shown in Table 1, static-pattern baselines (SWA-256, StreamingLLM, and BigBird) incur substantial perplexity degradation across all three language modeling corpora because they blindly discard distant context once tokens exit the fixed local window. Fine-grained ETA ($b = 4$) achieves dense-level perplexity across all three corpora and attains the best performance on every language modeling and zero-shot reasoning benchmark. When scaling block size by $16\times$ to our hardware-aligned Tensor Core deployment ($b = 64$), ETA continues to remain competitive across all corpora while preserving full reasoning accuracy.

4.3 Block Granularity and GQA Union Loading

GQA Union Sharing Favors Larger Block Sizes On GPU hardware, Grouped-Query Attention ($G = H_Q/H_{\text{KV}}$) loads the union of active KV blocks across all query heads sharing a KV head. We find that while fine-grained blocks ($b = 4$) minimize per-head density (Table 16), coarser blocks ($b = 64$) lead to higher cross-head overlapping and are thus significantly more efficient at a fixed *physical HBM bandwidth budget* (Table 2). Thus, for the exact same HBM memory transfer, $b = 64$ allows each query head to attend to $+9.5\%$ more context and improves perplexity by 8.8 points. This is very beneficial during inference because it enables coalesced Tensor Core execution (Appendix J).

Table 2: Block size at matched $\sim$51% GQA-union bandwidth and 126M C4 PPL ($L = 2048$).

b	z	D_{head}	D_{union}	PPL (↓)
4	2.0	34.77%	51.57%	102.32
8	2.0	37.23%	53.95%	98.39
16	1.5	32.70%	47.51%	99.07
32	1.0	39.00%	50.61%	95.92
64	1.0	**44.26%**	51.94%	**93.48**

Comparison with Quest (Tang et al., 2024) At block size $b = 64$, ETA's Dual Bound (Equation 7) captures a substantially higher fraction of true above-threshold keys than Quest's coordinate-wise bounding box (Table 3; see Appendix K for the geometric analysis). This advantage stems from ETA's robustness to coordinate outliers. Quest bounds each block by summing per-coordinate extrema ($\sum_{i=1}^{d_h} \max(q_i k_{\max,i}, q_i k_{\min,i})$), which loosens in high dimensions whenever isolated coordinates spike. In contrast, ETA's dual bounds are much sharper, reducing block false negatives and closing most of the perplexity gap between $b = 64$ and $b = 4$ (Table 3, Table 17).

4.4 Long-Context RULER Multi-Needle and Multi-Hop Retrieval

To evaluate whether sparse selection preserves recall under distractors and multi-hop reasoning, we benchmark four RULER-like tasks (Hsieh et al., 2024)—Single-Needle ($K = 1$), Multi-Key distractors ($K = 4$), Multi-Value recall ($K = 2$), and 2-Hop Variable Tracking—across $L \in \{2048, 4096\}$ (Table 4), alongside single-needle extrapolation up to $L = 8192$ (Table 19, Subsection J.2).

While H_2O and Quest degrade severely under distractors and 2-hop chains, ETA dynamically lowers τ_t on retrieval heads ($\Delta\tau_{\text{ret}} = -0.48$ to -2.19) to expand active density by $+2.4\%$–6.5% (Table 4). Across $N = 432$ trials at $L = 2048$, ETA ($b = 4$) achieves $\mathbf{88.0\% \pm 2.9\%}$ RULER accuracy,

Table 3: Pareto Frontier on C4 Perplexity ($L = 2048$), Token Recall, and Needle Retrieval ($L = 4096$) on the 1.45B model across block sizes $b \in \{4, 64\}$ vs. Quest (Tang et al., 2024) ($b = 64$) and H_2O (Zhang et al., 2023). Extended offset sweep in Table 17.

Method	Config	D_{head}	D_{union}	Recall (↑)	C4 PPL (↓)	Needle (↑)
Dense Baseline	FA2 / SDPA	100%	100%	100%	17.49 ± 0.33	66.7%
ETA ($b = 4$)	$a = +0.4$	36.3%	66.6%	98.3%	$\mathbf{17.56 \pm 0.33}$	**66.7%**
	$a = -0.8$	24.4%	50.6%	97.1%	17.82 ± 0.33	**66.7%**
	$a = -2.5$	**13.1%**	**30.1%**	94.8%	19.81 ± 0.34	**66.7%**
ETA ($b = 64$)	$a = +0.2$	55.3%	79.0%	**98.4%**	18.30 ± 0.29	**66.7%**
	$a = -0.4$	48.3%	69.7%	97.3%	18.60 ± 0.29	**66.7%**
	$a = -1.8$	37.6%	49.6%	91.6%	19.21 ± 0.30	**66.7%**
Quest ($b = 64$)	top-38%	41.1%	66.7%	54.0%	18.17 ± 0.34	66.7%
	top-25%	26.5%	47.3%	36.6%	19.26 ± 0.35	66.7%
	top-10%	12.8%	24.2%	19.4%	22.37 ± 0.39	66.7%
H_2O Eviction	$W = 256$	48.0%	68.0%	—	17.59 ± 0.33	22.2%
	$W = 64$	17.5%	28.8%	—	17.91 ± 0.33	0.0%

surpassing dense attention and matching it at $L = 4096$. At hardware-aligned tile size $b = 64$, ETA also outperforms dense attention at $L = 2048$, beating Quest and H_2O by $\mathbf{3.3} \times$ –$\mathbf{8.8}\times$.

Table 4: RULER multi-needle and 2-hop retrieval. Full 16-arm suite with 95% CIs in Subsection J.2.

Method	GQA Union (2K)	Single (2K)	MultiKey (2K)	MultiVal (2K)	2-Hop VT (2K)	RULER Avg $L = 2048$	RULER Avg $L = 4096$
Dense Baseline	100.0%	87.0	76.9	**88.9**	87.0	85.0 ± 3.2	$\mathbf{55.6 \pm 4.1}$
H2O ($W = 256, \sim 38\%$)	72.3%	25.0	12.0	*0.9*	*0.9*	9.7 ± 2.8	*1.6 ± 1.2*
Quest ($b = 64$, **top-**38%)	51.2%	55.6	23.1	12.0	13.0	25.9 ± 4.1	10.0 ± 2.8
ETA ($b = 64, a = 0.0$)	**37.4 → 39.8%**	**87.0**	**76.9**	**78.7**	**67.6**	$\mathbf{77.6 \pm 3.8}$	$\mathbf{46.5 \pm 4.2}$
ETA ($b = 64, a = 0.4$)	54.6 → 57.9%	**87.0**	**88.0**	**88.9**	**77.8**	$\mathbf{85.4 \pm 3.1}$	$\mathbf{53.0 \pm 4.2}$
ETA ($b = 4, a = 0.4$)	66.3 → 69.8%	**88.0**	**87.0**	**88.9**	**88.0**	$\mathbf{88.0 \pm 2.9}$	$\mathbf{55.6 \pm 4.1}$

4.5 Comparison with Native Sparse Attention (NSA)

We benchmark ETA against Native Sparse Attention (NSA) (Yuan et al., 2025) on an identical 126M backbone (163.8M tokens across 10,000 steps on single-node H100 GPUs; see Appendix I). ETA optimizes faster and converges to a lower pretraining loss (4.370–4.399 vs. 4.508, $\Delta\text{PPL} = -9.4$) at matched H100 training step throughput ($\mathbf{0.5315}$ s/step vs. $\mathbf{0.5311}$ s/step; Figure 8). In downstream generation ($L = 2048$), ETA achieves lower perplexity on FineWeb (99.80 vs. 103.55) and WikiText-2 (158.64 vs. 161.65) at 53% sparsity (Table 14). Furthermore, ETA's fused single-pass decode kernel eliminates multi-kernel launch overhead, achieving up to 2.72× lower step latency and 2.72× higher throughput with 1.63× speedup at 32K context (Table 15).

4.6 Elimination of Attention Sinks and Autonomous Layer Specialization

In standard transformers, excess probability mass is deposited onto initial anchor tokens, creating persistent *attention sinks* (Xiao et al., 2024) that post-hoc eviction methods (e.g., StreamingLLM, H_2O) must explicitly pin to avoid collapse. Under ETA, **localized attention sinks disappear entirely** (Figure 4, Left): initial-token attention mass drops from $> 50\%$ in dense models to $< 2\%$, as unallocated probability mass routes into the diffuse background floor (Appendix H). Simultaneously, ETA discovers distinct layer-wise sparsity allocations autonomously (Figure 4, Right): Layer 0 acts as a broad aggregator ($\approx 96\%$ density), intermediate layers compress context aggressively (reaching $< 8\%$ density at Layer 7), and deeper layers selectively restore capacity for final token synthesis.

4.7 Runtime Contextuality and Domain-Specific Offline Calibration

Why Contextuality Matters and What It Encodes. Pretraining an input-independent baseline with static per-head thresholds τ_h^ℓ (without $W_\tau^\ell q_t + b_\tau^\ell$) converges to 2.13× higher active density than ETA. Regressing realized thresholds $\tau_{t,h}^\ell$ against eight query and score-distribution statistics on held-out C4 explains at most 32.7% of variance (Table 8), with $\approx 70\%$ occurring *within* individual documents (Subsection D.1). These results confirm that contextuality is a necessary part of our architecture.

Distilling Static Thresholds for Stationary Deployments. For stationary deployments, the predictor can be distilled post-hoc into frozen constants $c^{\ell h}$ via offline calibration (Algorithm 1 in

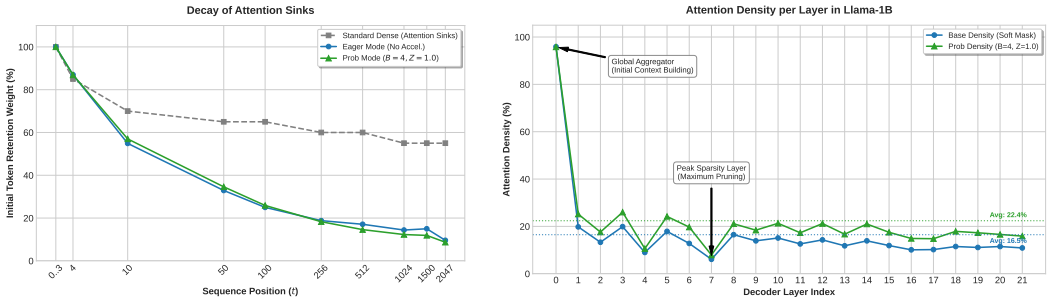


Figure 4: Mechanistic Dynamics of Attention in ETA. (Left) Localized attention sinks disappear under ETA's uniform background floor. **(Right)** Autonomous layer-wise active density across Transformer depth.

Subsection D.2), which accumulates a $1/t$-weighted score histogram over 16 sequences and solves via binary search for each head's training density. On held-out C4 (Table 9, Subsection D.3), calibrated static thresholds match dynamic perplexity in-distribution and curb RoPE density inflation at $2 \times L_{\text{train}}$, cutting compute at no perplexity cost.

4.8 Decode Speed and End-to-End Efficiency

We evaluate ETA's hardware speedups by benchmarking single-step decode latency ($q_{\text{len}} = 1$) against FlashAttention-2 (Dao, 2023) on an NVIDIA H100 GPU (batch sizes $B \in \{32, 64, 128\}$, context lengths up to 512K; Table 5, with block/chunk size ablations in Table 10).

Although short sequences ($L = 2048$) face a $\sim 50\%$ GQA union floor due to fixed boundary blocks (`lwb` $= 2$ plus the tail block), this overhead becomes negligible ($< 0.2\%$) at $L = 64$K–512K, enabling effective sparsity levels of 16%–35.5% union density (Subsection 4.7). At those configurations, ETA achieves a $1.31\times$ –$1.48\times$ kernel speedup while preserving full dense retrieval accuracy on needle-in-a-haystack tasks. At higher sparsity (16.0% union), kernel speedup reaches $2.36\times$. Across a full 22-layer model, these kernel gains deliver up to a $2.15\times$ end-to-end generation speedup with negligible indexing overhead during prefill (Appendix L).

Table 5: Single decode-step attention latency, ETA versus FlashAttention-2. NVIDIA H100 SXM 80 GB, `float16`, $H_Q = 32$, $H_{KV} = 4$ ($G = 8$), $d_h = 64$, $b = 64$. Speedup ($\times$) is $t_{\text{FA2}}/t_{\text{ETA}}$.

			51.2% **union** (38% **head**)			35.5% **union** (25% **head**)			16.0% **union** (10% **head**)		
B	L	KV	FA2	ETA	$\times$	FA2	ETA	$\times$	FA2	ETA	$\times$
32	64K	2 GB	0.710	0.983	0.72	0.710	0.767	0.93	0.711	0.526	**1.35**
32	128K	4 GB	1.398	1.830	0.76	1.395	1.426	0.98	1.396	0.936	**1.49**
32	256K	8 GB	2.772	3.510	0.79	2.775	2.732	1.02	2.774	1.747	**1.59**
32	512K	16 GB	5.529	6.787	0.81	5.525	5.275	1.05	5.533	3.364	**1.64**
64	64K	4 GB	1.757	1.833	0.96	1.885	1.435	**1.31**	1.897	0.937	**2.02**
64	128K	8 GB	3.872	3.504	1.11	3.653	2.746	**1.33**	3.519	1.750	**2.01**
64	256K	16 GB	7.275	6.813	1.07	7.790	5.280	**1.48**	7.722	3.373	**2.29**
64	512K	32 GB	15.085	13.432	**1.12**	15.009	10.430	**1.44**	14.790	6.520	**2.27**
128	64K	8 GB	3.807	3.516	1.08	3.518	2.737	**1.29**	3.939	1.746	**2.26**
128	128K	16 GB	7.624	6.841	1.11	7.430	5.368	**1.38**	7.923	3.351	**2.36**
128	256K	32 GB	14.836	13.481	1.10	15.302	10.461	**1.46**	15.039	6.622	**2.27**

5 Limitations and Conclusion

We introduced Elastic Threshold Attention (ETA), an end-to-end trainable sparse attention architecture that contracts unselected logits toward 0 via a SiLU gate rather than deleting them to $-\infty$. This creates a uniform attention floor that eliminates localized attention sinks and makes the model naturally invariant to inference-time distribution shifts. Paired with our fused Triton decode kernel, ETA matches or exceeds dense quality across language modeling, reasoning, and RULER retrieval benchmarks while delivering up to $2.36\times$ kernel decode speedups over FlashAttention-2.

Natural extensions of this work span both systems and modeling. On the systems side, integrating Hopper asynchronous warp-specialization (`WGMMA`/`TMA` (Dao, 2023)) and `FP8` metadata quantization can further reduce screening overhead at moderate sparsity levels ($\sim 50\%$ union density), while incorporating continual long-context pretraining (32K–128K) and exploring alternative zero-contracting gates offer promising directions for large-scale inference serving.

AI USE STATEMENT

In this work, we used generative AI tools for editing and tightening prose and as a coding aid when implementing and debugging the Triton kernels and the evaluation harness. We also used AI tools for assistance in deriving the attention gradients used in our backward kernel and in running the ANOVA decomposition test in Subsection D.1. Additionally, we used generative AI tools for creating and refining Tikz figures, as well as for suggesting experimental parameters.

We have not used generative AI tools for research ideation, retrieval and discovery, or generating synthetic datasets. We did not use AI tools in writing the core logic of the ETA model. We have reviewed all AI-assisted work. We manually inspected all lines of code written by generative AI and ran numerous experiments for correctness verification. All mathematical and conceptual claims, including potentially erroneous ones, are our own. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

bloc97. Ntk-aware scaled rope allows llama models to have extended (8k+) context size..., 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.

Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, N. Nolte, Y. Tian, M. Douze, L. Bottou, Z. Jia, et al. Magicpig: Lsh sampling for efficient llm generation. In *International Conference on Learning Representations*, volume 2025, pages 44169–44190, 2025.

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

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

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

A. Desai, S. Yang, A. Cuadron, M. Zaharia, J. E. Gonzalez, and I. Stoica. Hashattention: Semantic sparsity for faster inference. *arXiv preprint arXiv:2412.14468*, 2024.

Y. Gao, Z. Zeng, D. Du, S. Cao, P. Zhou, J. Qi, J. Lai, H. K.-H. So, T. Cao, F. Yang, and M. Yang. Seerattention: Learning intrinsic sparse attention in your LLMs. In *Advances in Neural Information Processing Systems*, 2025.

T. Haris. knn attention demystified: A theoretical exploration for scalable transformers. In *International Conference on Learning Representations*, volume 2025, pages 47576–47603, 2025.

C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

B. Hu, Z. Huang, M. Hu, Z. Zhang, and Y. Dou. Adaptive threshold selective self-attention for chinese ner. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1823–1833, 2022.

H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models, 2016.

G. Penedo, H. Kydlíček, A. Lozhkov, M. Mitchell, C. Raffel, L. Von Werra, T. Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Z. Shen, M. Zhang, H. Zhao, S. Yi, and H. Li. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3531–3539, 2021.

J. Shi, Y. Wu, Y. Peng, B. Wu, L. Wang, G. Liu, and Y. Luo. Trainable dynamic mask sparse attention. *arXiv preprint arXiv:2508.02124*, 2025.

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

P. Tillet, H.-T. Kung, and D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895, 2024.

V. Yerram, C. You, S. Bhojanapalli, S. Kumar, P. Jain, P. Netrapalli, et al. Hire: High recall approximate top-k estimation for efficient llm inference. *arXiv preprint arXiv:2402.09360*, 2024.

J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025.

M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

J. Zhang, C. Xiang, H. Huang, J. Wei, H. Xi, J. Zhu, and J. Chen. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. In *Proceedings of the 42nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2025.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. 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.

A TRANSFORMER PRELIMINARIES

We focus on the Decoder-only Transformer architecture, viewed as a mapping from sequences $(x_1, ..., x_n) \in \mathbb{R}^{n\times d}$ of token embeddings to sequences of latent embeddings $(h_1^L, ..., h_n^L) \in \mathbb{R}^{n\times d}$, with L being the total number of layers and d being the embedding dimension. Let $h_t^0 := x_t \in \mathbb{R}^d$.

The processing flow of a single token t through a specific layer ℓ maps $h_t^{\ell-1}$ to h_t^ℓ. Let H be the number of attention heads and $d_h = \frac{d}{H}$ be the head dimension. We compute the query q_t, key k_t, and value v_t embeddings via linear projections $W_Q, W_K, W_V \in \mathbb{R}^{d_h\times d}$ on the normalized input $\hat{h} = \text{RMSNorm}(h_t^{\ell-1})$. Rotary positional embeddings (RoPE) are applied as follows:

$$q_t = \mathcal{R}_t(W_Q \cdot \hat{h}), \quad k_t = \mathcal{R}_t(W_K \cdot \hat{h}), \quad v_t = W_V \cdot \hat{h}$$

where $\mathcal{R}_t \in \mathbb{R}^{d_h\times d_h}$ is a fixed unitary rotation matrix depending on position t.

The model computes the attention output by having query q_t attend to keys $\{k_s\}_{s\le t}$ and values $\{v_s\}_{s\le t}$. The attention weights and output are computed via scaled dot-product:

$$A_{ts} = \text{softmax}\left(\frac{\langle q_t, k_s\rangle}{\sqrt{d_h}} + M_{ts}\right), \qquad o_t = \sum_{s\le t} A_{ts} v_s$$

where M_{ts} is a causal mask (0 if $s \le t$, $-\infty$ otherwise).

The layer output h_t^ℓ is formed via residual connections and a Multi-Layer Perceptron (MLP), with output projection $W_O \in \mathbb{R}^{d\times d_h}$:

$$z_t = h_t^{\ell-1} + W_O \cdot o_t$$
$$h_t^\ell = z_t + \text{MLP}(\text{RMSNorm}(z_t))$$

During inference, Transformer-based LLMs operate in two distinct stages:

1. **Pre-fill**: A prompt of size L is processed in a single forward pass. This is *compute-heavy*, requiring $\Theta(L^2 d)$ FLOPS.
2. **Decoding**: Subsequent tokens are autoregressively predicted. The keys and values of previous tokens are stored in a **KV Cache** to avoid recomputation, making decoding a strictly *memory-bound* operation.

B TRAINING DETAILS AND HYPERPARAMETERS

We train our 1.45B-parameter ETA model on a slice of the FineWeb dataset (`CC-MAIN-2018-26`) (Penedo et al., 2024) at sequence length $L = 2048$ for 10,000 steps across 8 NVIDIA H100 GPUs (effective global batch size 2048 sequences, ≈42B tokens total, exceeding Chinchilla-optimal compute by $1.4\times$). The backbone comprises 22 Transformer decoder layers with $d_{\text{model}} = 2048$, intermediate SwiGLU dimension 8192, $H_Q = 32$ query heads ($d_k = 64$), and $H_{KV} = 4$ GQA key-value heads. Optimization uses AdamW ($\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 10^{-8}$, weight decay 0.1, gradient clipping 1.0) with cosine learning rate decay from 4×10^{-4} to 4×10^{-5} after 300 warmup steps.

Per-layer linear threshold predictors ($W_\tau^\ell \in \mathbb{R}^{2048\times 32}$, zero-initialized weights with start-dense bias $b_\tau^\ell = -8.0$) act on concatenated post-RoPE query heads. The inverse temperature β is linearly annealed from 1.0 to 5.0 over steps 0–7,000 and held constant thereafter. To prevent premature pruning before stable representations form, sparsity regularization is zero for the first 750 steps, ramps linearly over 1,690 steps to a peak of $\lambda_{\text{sparse}} = 5 \times 10^{-4}$, and is then reduced to 2.5×10^{-5} (5% of peak) for the remainder of training.

B.1 OPTIMIZER DYNAMICS AND ESCAPE FROM START-DENSE INITIALIZATION

With start-dense initialization ($b_\tau^\ell = -8.0, W_\tau^\ell = 0$) and initial attention logits centered near zero, terminal sharpness ($\beta = 5$) would evaluate to $\sigma(40)$, stalling gradient descent ($\sigma'(40) \approx 4.3 \times 10^{-18}$). Two mechanisms can explain the observed threshold migration without gradient starvation: (1) *temperature annealing* starts at $\beta = 1.0$, keeping the initial sigmoid derivative fourteen

orders of magnitude higher ($\sigma'(8) = 3.35 \times 10^{-4}$) until thresholds enter the score bulk; (2) *Adam's scale normalization* ($\hat{m}/(\sqrt{\hat{v}} + \epsilon) \approx \pm 1$) converts the monotone directional pressure of $\mathcal{L}_{\text{sparsity}}$ into steady step-size drift independent of small gradient magnitude.

C Additive vs Multiplicative Masking

In this section we examine additive masking as a substitute for multiplicative masking in ETA. We first compare the pretraining trajectories of ETA under additive (*Strict Mode*, $\tilde{S}^h_{ts} = S^h_{ts} + 100(m^h_{ts} - 1)$) and multiplicative (*Normal Mode*, $\tilde{S}^h_{ts} = S^h_{ts} \cdot m^h_{ts}$) formulations under identical regularization schedules ($\lambda = 0.05$, ramping over steps 750–2050) on 126M-parameter models.

As illustrated in Figure 5(a), both formulations exhibit nearly identical cross-entropy loss trajectories on FineWeb, with multiplicative masking gaining a slight edge. However, Figure 5(b) reveals distinct sparsity steady states. Under multiplicative masking, sub-threshold logits scale towards 0, leaving a uniform baseline softmax floor that allows the threshold predictor to push sparsity aggressively to 83.14% (16.86% active density) without language modeling loss penalties. Conversely, Strict Mode imposes hard exponential suppression ($e^{S^h_{ts}-100} \approx 0$), eliminating masked key-value states entirely from the attention context.

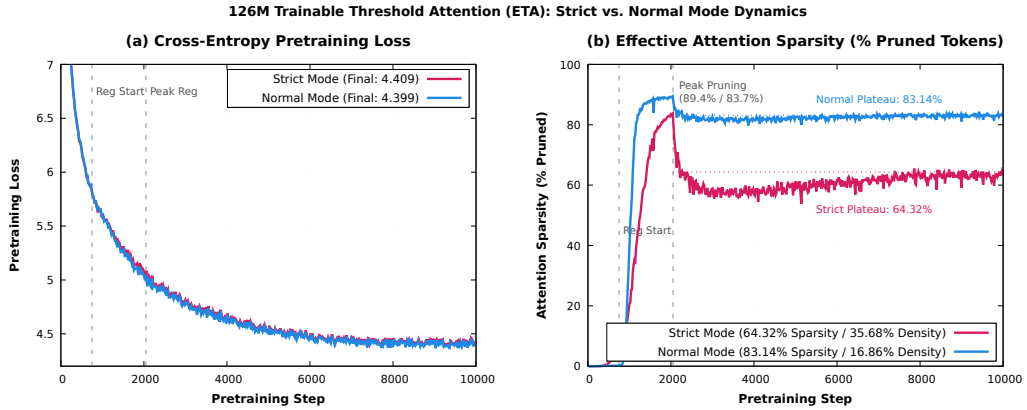


Figure 5: Pretraining dynamics and sparsity evolution across ETA masking formulations (126M parameters, 10k steps).

Inference Performance and Masking Mode Comparison. As reported in Table 6, additive masking proves inferior to multiplicative masking across all downstream evaluation benchmarks when deployed with block-sparse inference kernels. Specifically, additive masking incurs higher perplexity across FineWeb (116.38 vs. 99.80), C4 (117.93 vs. 100.73), and WikiText-2 (177.16 vs. 158.64) while retaining significantly lower compute sparsity ($\sim$ 35–38% vs. 65.91% pruned).

D Analysis of Runtime Contextuality and Offline Threshold Calibration

This appendix provides the complete empirical and mathematical analysis supporting Subsection 4.7: (1) testing whether query conditioning is necessary during pretraining and probing what the learned thresholds encode (Subsection D.1), (2) deriving the $1/t$-weighted score histogram formulation for exact density-matched calibration (Algorithm 1), and (3) evaluating distilled static thresholds in-distribution and under length extrapolation (Subsection D.3).

D.1 Empirical Analysis of Runtime Contextuality

Necessity of Query Conditioning During Pretraining. To test whether input-dependent thresholds ($\tau^\ell_{t,h} = w^{\ell\top}_{\tau,h} q_t + b^\ell_{\tau,h}$) are required to learn sparse attention representations from scratch, we trained an ablated baseline where each head's threshold is parameterized as a single learnable, input-independent scalar τ^ℓ_h. Under identical sparsity regularization ($\lambda = 0.05$), the input-independent baseline fails to sparsify effectively, converging to 2.13$\times$ higher active density than contextual ETA. Because a

Table 6: Downstream perplexity and sparsity comparison between Normal and Strict ETA (126M parameters, 10k pretraining steps). Evaluated using 1% prefill and 99% autoregressive decode under the block-sparse inference kernel ($b = 4$, threshold adjustment $a = 0.4$, $z = 2.0$).

Metric / Benchmark	Normal ETA	Strict ETA
Pretraining Loss (Step 10k)	**4.3986**	4.4085
FineWeb (200 batches, 1.62M tokens)		
Perplexity (PPL)	**99.80 ± 22.59**	116.38 ± 24.38
Compute Sparsity (FLOPs Pruned)	**65.91%** (34.09% density)	37.26% (62.74% density)
GQA Union Sparsity (I/O Pruned)	**49.23%** (50.77% density)	23.59% (76.41% density)
C4 (200 batches, 1.62M tokens)		
Perplexity (PPL)	**100.73 ± 17.56**	117.93 ± 21.20
Compute Sparsity (FLOPs Pruned)	**65.91%** (34.09% density)	38.18% (61.82% density)
GQA Union Sparsity (I/O Pruned)	**49.23%** (50.77% density)	24.54% (75.46% density)
Wikitext-2 (Full test set, 2.94M tokens)		
Perplexity (PPL)	**158.64 ± 22.15**	177.16 ± 23.98
Compute Sparsity (FLOPs Pruned)	**65.91%** (34.09% density)	34.92% (65.08% density)
GQA Union Sparsity (I/O Pruned)	**49.23%** (50.77% density)	21.23% (78.77% density)

Table 7: Mechanistic and downstream comparison of masking formulations (126M parameters, FineWeb). Tail statistics are measured over the sub-threshold key set ($n = 24$ sequences, $L = 1024$). Downstream metrics reflect block-sparse decode ($b = 4, z = 2.0, a = 0.4$). Multiplicative gating flattens sub-threshold keys into a uniform background, achieving superior perplexity and density over additive deletion.

Sub-threshold tail metric	Pre-gate (raw)	Normal mode (multiplicative)	Strict mode (additive)
Sub-threshold probability mass	42.4%	**43.8**% (preserved)	$\mathbf{1.5 \times 10^{-25}}$ (annihilated)
Normalized tail entropy ($H/\log n$)	0.790	**0.916** (near-uniform)	**0.081** (collapsed)
Tail max-to-mean ratio	15.6	**2.03** (flattened)	**249.7** (extreme peakiness)
Block-sparse kernel decode, FineWeb			
Perplexity (↓)	—	**99.80**	116.38 (+16.6%)
Compute density (↓)	—	**34.09**%	62.74% (1.84×)

static threshold applies uniformly across all sequence positions, any threshold high enough to prune routine tokens inevitably starves complex synthesis tokens of required context; consequently, gradient descent pushes the shared scalar downward to protect high-loss predictions.

Probing What $\tau_{t,h}^{\ell}$ Encodes. Given that query conditioning is essential, we next ask whether the learned linear predictor W_τ^ℓ merely computes a proxy for simple surface statistics (such as query norm or sequence position), or whether it captures higher-order semantic context. We evaluate our 126M ETA checkpoint on held-out C4 sequences and regress the realized per-token thresholds $\tau_{t,h}^{\ell}$ (separately for each layer, head, and position) against eight candidate summary statistics: the query Euclidean norm $\|q_t\|$, the log-sum-exp of the pre-softmax attention row (lse), the row score maximum ($s_{\max}$), standard deviation (s_{std}), mean (s_{mean}), and Shannon entropy (H), the logarithmic sequence position ($\log t$), and the input hidden-state norm ($\|x_t\|$).

Table 8: Proportion of variance (R^2) in learned thresholds $\tau_{t,h}^{\ell}$ explained by surface query and score-distribution statistics on held-out C4 across all layers and heads (126M model). Even when combined in a joint multiple regression, all eight statistics account for less than one-third of threshold variance.

Statistic	$\|q_t\|$	lse	$s_{\max}$	s_{std}	H	$\log t$	s_{mean}	$\|x_t\|$	**Joint (All 8)**
R^2	0.187	0.176	0.168	0.156	0.091	0.071	0.059	0.021	**0.327**

As shown in Table 8, the strongest individual feature ($\|q_t\|$) accounts for only 18.7% of the variance in $\tau_{t,h}^{\ell}$, and a joint linear regression combining all eight statistics explains only 32.7%. More than two-thirds (67.3%) of the variance in the learned threshold is irreducible to surface summary statistics, confirming that W_τ^ℓ exploits fine-grained directional information in q_t.

Within- vs. Between-Document ANOVA Decomposition. To test whether ETA's threshold variations merely reflect global document-level domain shifts (e.g., allocating higher density to technical passages and lower density to conversational text) rather than local token-by-token difficulty, we decompose the variance of realized active densities $d_{i,t} = \frac{1}{t}\sum_{s=1}^{t} \mathbb{I}(S_{i,t,s} > \tau_{i,t})$ across N held-out C4 documents of length T. Because active density naturally decays with sequence position t (as

a fixed number of relevant tokens represents a smaller fraction of a growing causal window), we first remove this mechanical positional trend by subtracting the cross-document positional mean $\bar{d}_{\cdot,t} = \frac{1}{N}\sum_{i=1}^{N} d_{i,t}$, yielding position-detrended residual densities $\tilde{d}_{i,t} = d_{i,t} - \bar{d}_{\cdot,t}$. Applying a one-way analysis of variance (ANOVA) partitions the total sum of squares across all $N \times T$ tokens into orthogonal between-document and within-document components:

$$\underbrace{\sum_{i=1}^{N}\sum_{t=1}^{T}(\tilde{d}_{i,t} - \bar{\tilde{d}})^2}_{\text{SS}_{\text{total}}} = \underbrace{T\sum_{i=1}^{N}(\bar{\tilde{d}}_{i,\cdot} - \bar{\tilde{d}})^2}_{\text{SS}_{\text{between}}\ (\approx 30\%)} + \underbrace{\sum_{i=1}^{N}\sum_{t=1}^{T}(\tilde{d}_{i,t} - \bar{\tilde{d}}_{i,\cdot})^2}_{\text{SS}_{\text{within}}\ (\approx 70\%)},$$

where $\bar{\tilde{d}}_{i,\cdot} = \frac{1}{T}\sum_{t=1}^{T}\tilde{d}_{i,t}$ is the mean detrended density of document i. If ETA functioned merely as a coarse document-level register switch, document means $\bar{\tilde{d}}_{i,\cdot}$ would dominate and the between-document share would approach 100%. Instead, $\approx 70\%$ of the variance occurs *within* individual documents ($\text{SS}_{\text{within}}/\text{SS}_{\text{total}} \approx 0.70$), demonstrating that ETA continuously expands and contracts its attention window token by token—pruning aggressively on predictable syntax tokens while widening context on semantically demanding retrieval steps.

D.2 Offline Threshold Calibration via $1/t$-Weighted Histograms

At decoding step t, an attention head's active density is the fraction of its t available causal keys retained by the soft gate, $\frac{1}{t}\sum_{s=1}^{t} m_{ts}^{h}$. Averaged over a small, disjoint validation/calibration split $\mathcal{D}$ ($|\mathcal{D}| = 16$ sequences of length T), the dynamic predictor $\tau_{t,h}^{\ell}$ realizes a target head density

$$\bar{d}^{\ell h} = \frac{1}{|\mathcal{D}|T}\sum_{x\in\mathcal{D}}\sum_{t=1}^{T}\frac{1}{t}\sum_{s=1}^{t} m_{ts}^{h}(x).$$

For stationary deployments, our goal is to replace the dynamic predictor with a single static constant $c^{\ell h}$ per head that yields this exact same empirical density on $\mathcal{D}$: $g(c^{\ell h}) = \bar{d}^{\ell h}$, where

$$g(c) = \frac{1}{|\mathcal{D}|T}\sum_{x\in\mathcal{D}}\sum_{t=1}^{T}\frac{1}{t}\sum_{s=1}^{t}\sigma(\beta\cdot(S_{ts}^{\ell h}(x) - c))$$

denotes the head's expected density on $\mathcal{D}$ under threshold c. Because $g(c)$ decreases strictly and monotonically from 1 to 0 as c increases, we can find the unique root $c^{\ell h}$ via continuous binary search.

Score Histograms and $1/t$ Row-Weighting. Evaluating $g(c)$ directly over millions of raw attention scores S_{ts}^{h} across $\mathcal{D}$ at every binary search iteration would require repeatedly scanning the calibration split. Instead, during a single pass over $\mathcal{D}$, we bin raw scores $S_{ts}^{\ell h}$ into a 1D histogram $H^{\ell h}$ (B bins with centers s_b). When building this histogram, we increment bin counts by $+\frac{1}{t}$ rather than $+1$ to account for the causal weighting. Once $H^{\ell h}$ is built, evaluating any candidate threshold during binary search reduces to a fast B-element dot product:

$$g(c) = \sum_{b=1}^{B} H^{\ell h}[b]\,\sigma\big(\beta(s_b - c)\big).$$

Assuming B is large enough, binary search approximates the optimal value of c within machine precision (residual $< 2\times 10^{-16}$). See Algorithm 1 for more details.

D.3 Distillation Evaluation and Length-Extrapolation Stability

We evaluate calibrated static thresholds $\{c^{\ell h}\}$ against dynamic per-token thresholds $\tau_{t,h}^{\ell}$ on held-out C4 sequences executed directly through our fused block-sparse Triton kernel ($b = 4$) on the 126M model ($L_{\text{train}} = 1024$).

Both in-distribution and length-extrapolated experiments show that our calibrated static thresholds achieve slightly better perplexity and higher sparsity than the dynamic thresholds when evaluated on the target domain used for calibration.

Algorithm 1 Density-Matched Offline Threshold Calibration

Require: Trained ETA model M with dynamic predictor $\tau(\cdot)$; calibration split $\mathcal{D}$; bin count B; inverse temperature β

$H^{\ell h} \leftarrow \mathbf{0} \in \mathbb{R}^B, \quad \bar{d}^{\ell h} \leftarrow 0$ for all (ℓ, h)
for each calibration sequence $x \in \mathcal{D}$ **do**
  Forward pass $M(x)$ with dynamic thresholds, recording raw scores $S^{\ell h}$ and gates $m^{\ell h}$
  for each layer ℓ and head h **do**
    $\bar{d}^{\ell h} \mathrel{+}= \frac{1}{|\mathcal{D}|} \cdot \mathrm{rowmean}\left(m^{\ell h}\right)$ ▷ Accumulate target per-head density
    for each causal pair (t, s) with $s \le t$ **do**
      $H^{\ell h}\left[\,\mathrm{bin}(S^{\ell h}_{ts})\right] \mathrel{+}= \frac{1}{t}$ ▷ Accumulate $1/t$ causal row weight
    end for
  end for
end for
for each layer ℓ and head h **do**
  $H^{\ell h} \leftarrow H^{\ell h} / \sum_{b=1}^{B} H^{\ell h}[b]$ ▷ Normalize weighted score histogram
  Define $g(c) = \sum_{b=1}^{B} H^{\ell h}[b]\, \sigma\big(\beta(s_b - c)\big)$ ▷ Monotonically decreasing in c
  $c^{\ell h} \leftarrow \textsc{Bisect}\big(g(c) = \bar{d}^{\ell h}\big)$ ▷ Exact root-finding via bisection
end for
return Static threshold table $\{c^{\ell h}\}$

Table 9: Dynamic per-token thresholds ($\tau^{\ell}_{t,h}$) versus calibrated static per-(layer, head) thresholds ($c^{\ell h}$), evaluated on held-out C4 via the block-sparse decode kernel ($b = 4$, 126M model with $L_{\text{train}} = 1024$). Constants are calibrated on a disjoint 16-sequence split. In-distribution evaluations are paired at the batch level ($\rho = 0.9996$, $t = 8.64$).

Evaluation Regime	Metric	Dynamic $\tau^{\ell}_{t,h}$	Calibrated Static $c^{\ell h}$
In-Distribution ($L = 1024$)	Perplexity (↓)	91.96	**91.60**
	Active Head Density (↓)	24.14%	**22.40%**
Length Extrapolation ($L = 2048$, $z = 3.0$)	Perplexity (↓)	100.52	**100.48**
	Active Head Density (↓)	44.77%	**32.83%**

E Hardware-Aligned Training Kernel Implementation

To train our model efficiently without materializing $T \times T$ matrices, we implement a custom fused training attention kernel in Triton. Our implementation extends the block-based online softmax algorithm of FlashAttention-2 by tightly integrating the differentiable thresholding mechanism and the gradient tracking for the sparsity regularizer.

E.1 Forward Pass Formulation

In the forward pass, the kernel divides the query sequence into blocks of size B_M and the key/value sequences into blocks of size B_N. The outer loop iterates over query blocks, loading queries Q_{block} and their corresponding predicted thresholds τ_{block} from HBM to SRAM. The inner loop iterates causally over key/value blocks (K_{block}, V_{block}).

For a given query q_i and key k_j within these blocks, the scaled attention score is computed as $S_{ij} = \text{scale} \cdot \langle q_i, k_j \rangle$. Concurrently, the kernel computes the soft inclusion gate m_{ij} (denoted σ_{ij} below) via the sigmoid relaxation:

$$\sigma_{ij} = \sigma(\beta(S_{ij} - \tau_i))$$

In particular, the sum of these probabilities is accumulated directly in SRAM to compute the sparsity regularizer $\mathcal{L}_{\text{sparsity}}$ without allocating an explicit full sequence tensor:

$$R_i = \sum_{j \le i} \sigma_{ij}$$

The kernel then applies the selected dynamic mask operation (additive or multiplicative) to yield the masked scores $\tilde{S}_{ij}$. To prevent numerical overflow, we track the running maximum m_i and the

running exponential sum l_i via a standard online softmax update[3]:

$$m_i^{\text{new}} = \max(m_i, \max_j \tilde{S}_{ij})$$
$$l_i^{\text{new}} = l_i e^{m_i - m_i^{\text{new}}} + \sum_j e^{\tilde{S}_{ij} - m_i^{\text{new}}}$$

The attention output is accumulated as a weighted sum of V_{block} and normalized by l_i^{new} before being written back to HBM. We additionally store the final log-sum-exp vectors $L_i = m_i + \ln(l_i)$ and the row-wise sums R_i to facilitate the backward pass. Algorithm 2 gives the full forward pass.

Algorithm 2 Hardware-Aligned Forward Kernel

Require: Queries Q, Keys K, Values $V \in \mathbb{R}^{T \times d}$
Require: Thresholds $\tau \in \mathbb{R}^T$, Scale s, Temperature β
Ensure: Output $O \in \mathbb{R}^{T \times d}$, Regularizer Sums $\Sigma_{\text{sums}} \in \mathbb{R}^T$, Log-Sum-Exp $L \in \mathbb{R}^T$

for each query block i from 1 to T/B_M **do**
  Load Q_i, τ_i from HBM to SRAM
  Initialize $m_i \leftarrow -\infty$, $l_i \leftarrow 0$, $acc_i \leftarrow 0$, $\Sigma_{\text{sums},i} \leftarrow 0$
  for each key/value block j from 1 to i **do** ▷ Iterate up to causal limit
    Load K_j, V_j from HBM to SRAM
    $S_{ij} \leftarrow s \cdot Q_i K_j^T$ ▷ Compute pre-softmax scores
    $\sigma_{ij} \leftarrow \text{sigmoid}(\beta \cdot (S_{ij} - \tau_i))$ ▷ Compute inclusion gates
    $\Sigma_{\text{sums},i} \leftarrow \Sigma_{\text{sums},i} + \text{row_sum}(\sigma_{ij})$ ▷ Accumulate regularizer
    if additive **then**
      $\tilde{S}_{ij} \leftarrow S_{ij} + C \cdot \sigma_{ij} - C$ ▷ Additive masking
    else
      $\tilde{S}_{ij} \leftarrow S_{ij} \odot \sigma_{ij}$ ▷ Multiplicative masking
    end if
    Apply causal mask to $\tilde{S}_{ij}$; for diagonal blocks ($j = i$), restore token diagonal $\text{diag}(\tilde{S}_{ii}) \leftarrow \text{diag}(S_{ii})$ ▷ Bypass threshold on diagonal
    $m_{\text{new}} \leftarrow \max(m_i, \max(\tilde{S}_{ij}))$ ▷ Online softmax update
    $\alpha \leftarrow \exp(m_i - m_{\text{new}})$
    $P_{ij} \leftarrow \exp(\tilde{S}_{ij} - m_{\text{new}})$
    $l_{\text{new}} \leftarrow l_i \cdot \alpha + \text{row_sum}(P_{ij})$
    $acc_i \leftarrow acc_i \cdot \alpha + P_{ij} V_j$
    $m_i \leftarrow m_{\text{new}}, \quad l_i \leftarrow l_{\text{new}}$
  end for
  $O_i \leftarrow acc_i / l_i$ ▷ Normalize output
  $L_i \leftarrow m_i + \ln(l_i)$ ▷ Store log-sum-exp for backward pass
  Store $O_i, \Sigma_{\text{sums},i}, L_i$ from SRAM to HBM
end for

E.2 Backward Pass and Custom Gradients

The backward pass is significantly more complex than standard attention due to the chain rule dependencies flowing through the threshold τ_i and the regularization signal $\nabla_\sigma \mathcal{L}_{\text{sparsity}}$.

The backward kernel loops over the K and V blocks in the outer loop, and iterates over the Q blocks in the inner loop. For each block pair, the kernel recomputes the forward logits S_{ij}, the inclusion probabilities σ_{ij}, and the softmax probabilities $P_{ij} = \exp(\tilde{S}_{ij} - L_i)$.

Let $dO_i \in \mathbb{R}^d$ be the upstream gradient of the attention output, and $d\Sigma_i$ be the upstream gradient from the sparsity regularization loss. The standard derivatives are computed as follows (for completeness

[3] A familiar formulation from the original FlashAttention (Dao et al., 2022).

we include the derivations in Subsection E.3):

$$dV_j = \sum_i P_{ij} dO_i$$
$$dP_{ij} = dO_i^\top \cdot V_j$$
$$dS_{ij}^{\text{masked}} = P_{ij} \left(dP_{ij} - \sum_k P_{ik} dP_{ik} \right)$$

Because our soft inclusion mask relies on σ_{ij}, gradients must be routed back to the threshold predictor. The local gradient with respect to the threshold τ_i incorporates both the task loss (routed through the mask) and the sparsity regularizer (routed through $d\Sigma_i$). Using the derivative of the sigmoid function, $\sigma'_{ij} = \sigma_{ij}(1 - \sigma_{ij})$, the gradient step for the threshold is the following (see Subsection E.3):

$$d\tau_i = \sum_j \left(dS_{ij}^{\text{masked}} \frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} + d\Sigma_i \right) \left(-\beta \cdot \sigma'_{ij} \right)$$

Similarly, the gradient with respect to the raw pre-softmax attention score S_{ij} receives an additional term from the thresholding mechanism, yielding the final dS_{ij}^{raw} used to compute dK and dQ.

Because the backward pass's inner loop iterates over queries while holding keys constant, gradients for Q and τ are accumulated atomically across multiple inner iterations (`tl.atomic_add`), ensuring thread-safe gradient reductions across the sequence length without necessitating massive workspace buffers. Algorithm 3 gives the full backward pass.

Algorithm 3 Hardware-Aligned Backward Kernel

Require: Forward tensors Q, K, V, τ, O, L, upstream gradients $dO, d\Sigma$
Require: Scale s, Temperature β
Ensure: Gradients $dQ, dK, dV \in \mathbb{R}^{T \times d}$, $d\tau \in \mathbb{R}^T$

Initialize $dQ, d\tau \leftarrow 0$ in HBM
for each key block j from 1 to T/B_N **do**
  Load K_j, V_j from HBM to SRAM
  Initialize $dK_j \leftarrow 0$, $dV_j \leftarrow 0$ in SRAM
  for each query block i from j to T/B_M **do** ▷ Respect causal bounds
    Load $Q_i, \tau_i, O_i, L_i, dO_i, d\Sigma_i$ from HBM to SRAM
    Recompute $S_{ij} \leftarrow s \cdot Q_i K_j^\top$
    Recompute $\sigma_{ij} \leftarrow \text{sigmoid}(\beta \cdot (S_{ij} - \tau_i))$
    Recompute masked scores $\tilde{S}_{ij}$ ▷ Using strict or soft mask
    Apply causal/diagonal mask to $\tilde{S}_{ij}$
    $P_{ij} \leftarrow \exp(\tilde{S}_{ij} - L_i)$ ▷ Recompute attention probabilities
    $dV_j \leftarrow dV_j + P_{ij}^\top dO_i$ ▷ Accumulate value gradients
    $dP_{ij} \leftarrow dO_i V_j^\top$
    $D_i \leftarrow \text{row_sum}(dO_i \odot O_i)$ ▷ Softmax scaling factor
    $dS_{ij}^{\text{masked}} \leftarrow P_{ij} \odot (dP_{ij} - D_i)$ ▷ Gradient through softmax
    Compute threshold gradient $d\tau_{\text{local}}$ using dS_{ij}^{masked}, $d\Sigma_i$, and σ'_{ij}
    Compute raw pre-softmax gradient dS_{ij}^{raw}
    $dK_j \leftarrow dK_j + s(dS_{ij}^{\text{raw}})^\top Q_i$ ▷ Accumulate key gradients
    `atomic_add`$(dQ_i, dS_{ij}^{\text{raw}} K_j)$ ▷ Accumulate to HBM safely
    `atomic_add`$(d\tau_i, \text{row_sum}(d\tau_{\text{local}}))$
  end for
  Store dK_j, dV_j from SRAM to HBM
end for

E.3 Derivation of the Attention Gradients

To formulate the backward pass, we differentiate the scalar loss $\mathcal{L}$ with respect to the intermediate variables in reverse. We denote the upstream gradient of the loss with respect to any variable X as

$dX = \frac{\partial \mathcal{L}}{\partial X}$. Recall the forward attention operations for a query i and key j:

$$S_{ij} = s\langle Q_i, K_j \rangle, \quad P_{ij} = \frac{\exp(\tilde{S}_{ij})}{\sum_k \exp(\tilde{S}_{ik})}, \quad O_i = \sum_k P_{ik} V_k \tag{8}$$

where s is the scaling factor and $\tilde{S}_{ij}$ is the dynamically masked pre-softmax logit.

We first compute the gradients with respect to the value vectors V_j and the scalar attention probabilities P_{ij}. Applying the chain rule to the forward output sum yields:

$$dV_j = \sum_i P_{ij} dO_i, \quad \text{and} \quad dP_{ij} = dO_i \cdot V_j \tag{9}$$

Next, we route the gradient through the softmax normalization to find the derivative with respect to the masked logits, $\tilde{S}_{ij}$. Because $\tilde{S}_{ij}$ affects all probabilities P_{ik} for query i, we sum over k using the standard softmax Jacobian, $\frac{\partial P_{ik}}{\partial \tilde{S}_{ij}} = P_{ij}(\delta_{jk} - P_{ik})$, where δ_{jk} is the Kronecker delta:

$$dS_{ij}^{\text{masked}} = \sum_k dP_{ik} \frac{\partial P_{ik}}{\partial \tilde{S}_{ij}} = P_{ij} \left(dP_{ij} - \sum_k P_{ik} dP_{ik} \right) \tag{10}$$

The threshold τ_i and the raw pre-softmax logit S_{ij} interact through the inclusion probability $\sigma_{ij} = \sigma(\beta(S_{ij} - \tau_i))$. This probability branches into two computational pathways: the task loss (acting as a dynamic mask to produce $\tilde{S}_{ij}$) and the sparsity regularization loss (accumulating into the row-wise sum $R_i = \sum_j \sigma_{ij}$).

Both gradient derivations rely on the shared upstream gradient flowing through σ_{ij}. Using $dS_{ij}^{\text{masked}} = \frac{\partial \mathcal{L}}{\partial \tilde{S}_{ij}}$ and $d\Sigma_i = \frac{\partial \mathcal{L}}{\partial R_i}$, we apply the chain rule to find:

$$\frac{\partial \mathcal{L}}{\partial \sigma_{ij}} = dS_{ij}^{\text{masked}} \frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} + d\Sigma_i$$

Threshold Gradient ($d\tau_i$): The threshold τ_i only affects the loss via σ_{ij}. Because it influences all keys j up to the causal limit, we sum the chain rule across the sequence. Evaluating the inner derivative $\frac{\partial \sigma_{ij}}{\partial \tau_i} = -\beta \sigma'_{ij}$ (where $\sigma'_{ij} = \sigma_{ij}(1 - \sigma_{ij})$), we obtain:

$$d\tau_i = \sum_j \frac{\partial \mathcal{L}}{\partial \sigma_{ij}} \frac{\partial \sigma_{ij}}{\partial \tau_i} = \sum_j \left(dS_{ij}^{\text{masked}} \frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} + d\Sigma_i \right) \left(-\beta \cdot \sigma'_{ij} \right)$$

Raw Score Gradient (dS_{ij}^{raw}): Unlike the threshold, the raw score S_{ij} affects the loss through both the indirect sigmoid path (σ_{ij}) and the direct dynamic mask path ($\tilde{S}_{ij}$). Applying the multivariate chain rule:

$$dS_{ij}^{\text{raw}} = \frac{\partial \mathcal{L}}{\partial \tilde{S}_{ij}} \frac{\partial \tilde{S}_{ij}}{\partial S_{ij}} + \frac{\partial \mathcal{L}}{\partial \sigma_{ij}} \frac{\partial \sigma_{ij}}{\partial S_{ij}}$$

Notice the mathematical symmetry in the sigmoid's argument: the inner derivative with respect to the score is $\frac{\partial \sigma_{ij}}{\partial S_{ij}} = +\beta \sigma'_{ij}$. Substituting this and the shared intermediate term yields the final local gradient:

$$dS_{ij}^{\text{raw}} = dS_{ij}^{\text{masked}} \frac{\partial \tilde{S}_{ij}}{\partial S_{ij}} + \left(dS_{ij}^{\text{masked}} \frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} + d\Sigma_i \right) \left(\beta \cdot \sigma'_{ij} \right)$$

Masking Derivatives: To implement these equations natively, the exact values of the local mask derivatives are determined by the chosen dynamic masking strategy:

- **Additive Masking** ($\tilde{S}_{ij} = S_{ij} + C \cdot \sigma_{ij} - C$): The direct derivative is $\frac{\partial \tilde{S}_{ij}}{\partial S_{ij}} = 1$, and the sigmoid interaction is $\frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} = C$.
- **Multiplicative Masking** ($\tilde{S}_{ij} = S_{ij} \odot \sigma_{ij}$): The direct derivative is $\frac{\partial \tilde{S}_{ij}}{\partial S_{ij}} = \sigma_{ij}$, and the sigmoid interaction is $\frac{\partial \tilde{S}_{ij}}{\partial \sigma_{ij}} = S_{ij}$.

Query and Key Gradients As we saw, dS_{ij}^{masked} is combined with the sparsity regularization gradient to form the total upstream gradient for the raw score, denoted as dS_{ij}^{raw}. We distribute this raw gradient back to the query and key vectors. Differentiating the scaled dot-product $S_{ij} = sQ_iK_j^T$ yields symmetric accumulations:

$$dQ_i = \sum_j s \cdot dS_{ij}^{\text{raw}} K_j, \quad \text{and} \quad dK_j = \sum_i s \cdot dS_{ij}^{\text{raw}} Q_i \tag{11}$$

From a hardware implementation perspective, the scalar s is absorbed into dS_{ij}^{raw} immediately prior to these matrix multiplications. Because our backward kernel holds keys in the outer loop and iterates over queries in the inner loop, dK_j is accumulated safely in on-chip SRAM, whereas dQ_i requires atomic additions to prevent race conditions in global High Bandwidth Memory (HBM).

E.4 Complexity Analysis

To formally evaluate the hardware efficiency of our Threshold Attention mechanism, we analyze its space, time (FLOPs), and memory access (I/O) complexities. We follow the analytical framework of FlashAttention (Dao et al., 2022).

Let T be the sequence length and d be the head dimension. Let M denote the capacity of the fast on-chip SRAM. To maximize SRAM utilization without overflowing, our block sizes are constrained such that $B_M, B_N = \Theta(M/d)$. For this analysis, we assume a causal masking paradigm, meaning the inner loops traverse approximately $T^2/2$ total element interactions.

Forward Pass Complexity In standard attention, materializing the dense $T \times T$ attention matrix requires $\mathcal{O}(T^2)$ space and $\mathcal{O}(T^2)$ High Bandwidth Memory (HBM) accesses. Our hardware-aligned forward pass avoids this.

- **Space:** We only store the inputs $Q, K, V \in \mathbb{R}^{T\times d}$, the threshold parameters $\tau \in \mathbb{R}^T$, the final output $O \in \mathbb{R}^{T\times d}$, and the vectors required for the backward pass ($L, \Sigma \in \mathbb{R}^T$). The dynamic mask σ_{ij} is computed locally in SRAM and never explicitly materialized. Thus, the total space complexity is $\mathcal{O}(Td)$, identical to standard FlashAttention.
- **Time:** The inner loop computes the dot product, the sigmoid inclusion probability, and the softmax accumulator updates. While the sigmoid and masking operations introduce a constant factor overhead compared to raw FlashAttention, they are element-wise $\mathcal{O}(1)$ operations. Across all block pairs, the dominant operation remains the matrix multiplications ($Q_iK_j^T$ and $P_{ij}V_j$), yielding a time complexity of $\mathcal{O}(T^2d)$.
- **I/O (HBM Accesses):** The outer loop iterates over T/B_M query blocks. For each query block, it loads Q_i and τ_i from HBM ($\Theta(B_Md)$ elements). The inner loop iterates over T/B_N key/value blocks, loading K_j and V_j ($\Theta(B_Nd)$ elements). The total number of HBM accesses is given by:

$$\text{HBM Accesses} = \sum_{i=1}^{T/B_M} \left[\Theta(B_Md) + \sum_{j=1}^{i} \Theta(B_Nd) \right] = \mathcal{O}\left(\frac{T^2d}{B_M}\right) \tag{12}$$

  Substituting $B_M = \Theta(M/d)$, the overall I/O complexity is $\mathcal{O}(T^2d^2M^{-1})$.

Backward Pass Complexity The backward pass reconstructs the forward pass intermediate states to avoid saving the massive $T \times T$ matrices, trading additional FLOPs for substantially reduced memory reads.

- **Space:** The backward pass requires storing the gradients $dQ, dK, dV \in \mathbb{R}^{T\times d}$ and $d\tau \in \mathbb{R}^T$, requiring $\mathcal{O}(Td)$ space.
- **Time:** Recomputing S_{ij} and σ_{ij} requires the same $\mathcal{O}(T^2d)$ FLOPs as the forward pass. Computing the standard attention gradients ($dS_{ij}, dP_{ij}, dV_j, dK_j, dQ_i$) and the threshold gradient $d\tau_i$ involves purely linear matrix multiplications and element-wise derivatives. Thus, the total time complexity remains $\mathcal{O}(T^2d)$.

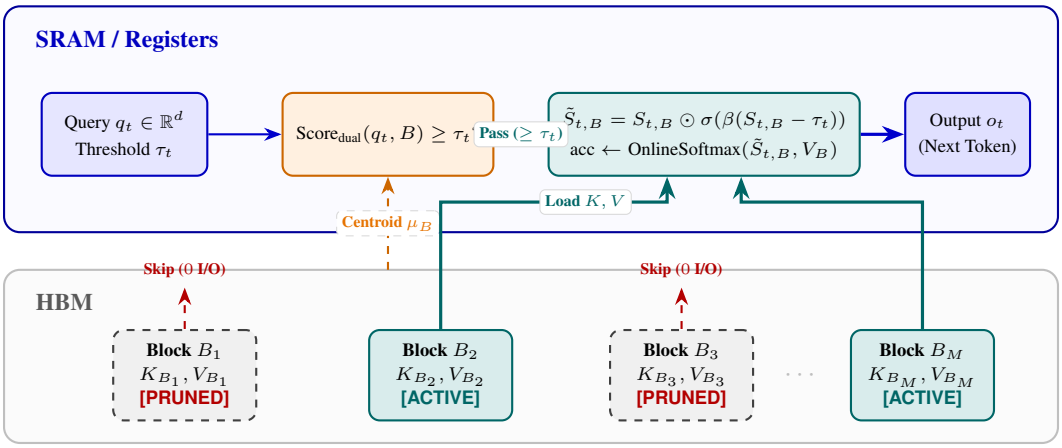


Figure 6: Inference-time block-sparse selective loading mechanism. Unimportant blocks are pruned in SRAM via lightweight metadata screening, while active blocks are loaded from HBM for fused online softmax accumulation.

- **I/O (HBM Accesses):** The outer loop iterates over T/B_N key blocks, holding K_j and V_j in SRAM while accumulating dK_j and dV_j. The inner loop streams the query blocks Q_i, dO_i and vectors $L, \tau, D, d\Sigma$ from HBM ($\Theta(B_M d)$ elements). The gradients dQ_i and $d\tau_i$ are accumulated to HBM using atomic additions. The total HBM access cost is:

$$\text{HBM Accesses} = \sum_{j=1}^{T/B_N} \left[\Theta(B_N d) + \sum_{i=j}^{T/B_M} \Theta(B_M d) \right] = \mathcal{O}\left(\frac{T^2 d}{B_N} \right) \tag{13}$$

  Substituting $B_N = \Theta(M/d)$, the backward I/O complexity is $\mathcal{O}(T^2 d^2 M^{-1})$.

F Inference-Time Block-Sparse Fused Kernel

Here we detail the algorithmic design and systems optimizations of our custom Triton (Tillet et al., 2019) fused kernel for block-sparse attention decoding. The design can be visualized in Figure 6.

F.1 GQA-Grouped Execution and Compact Metadata Layout

In Grouped-Query Attention (GQA), $G = H_Q/H_{KV}$ query heads share a single key-value head ($G = 8$ in our 1.45B model). If a decoding kernel launches a separate thread block for each query head, any KV block needed by multiple query heads in the same group is fetched from HBM up to G times, wasting memory bandwidth. To eliminate redundant reads, our kernel instead assigns each thread block to an entire GQA group of G query heads sharing a single KV head:

1. **Batched Metadata Screening:** For each chunk of candidate blocks, the thread block loads the per-block metadata ($\boldsymbol{\mu}_B, \boldsymbol{\sigma}_B, M_B$) once into SRAM and evaluates Score$_{\text{dual}}(q_t^h, B) \geq \tilde{\tau}_{t,h}$ across all G query heads simultaneously via a single Tensor Core matrix multiplication (`tl.dot`).
2. **Union KV Loading with Per-Head Masking:** A full KV block B is fetched from HBM into SRAM *at most once*—whenever at least one of the G query heads selects it (forming the GQA union). Once in SRAM, each query head updates its own online-softmax accumulator using only the blocks that passed its individual threshold $\tilde{\tau}_{t,h}$.

As a result, actual HBM traffic scales with the *GQA union density* D_{union} across the H_{KV} groups rather than the sum of individual query-head densities.

To minimize metadata I/O, each per-block record stores only the 64-dimensional centroid $\boldsymbol{\mu}_B$ (`float16`), 64-dimensional coordinate standard deviation $\boldsymbol{\sigma}_B$ (`float16`), and scalar maximum norm M_B (`float32`), totaling 258 bytes per block per KV head (1.6% overhead at $b = 64$). Caching $\boldsymbol{\sigma}_B$ rather than variance $\boldsymbol{\sigma}_B^2$ prevents half-precision underflow below 6×10^{-5} in low-variance blocks.

F.2 Mixed Precision and Numerical Stability

While Tensor Core matrix multiplications ($qK_B^\top$ and PV_B) execute in `float16`, squaring coordinates in half precision during proxy bound evaluation can induce overflow or cancellation. Consequently, all coordinate-wise square accumulations, dot-product bounding terms ($\langle q, \boldsymbol{\mu}_B \rangle$), scaling factors, and online softmax accumulators (m_c, l_c) are strictly promoted to `float32`.

F.3 Chunked Decoding and Two-Stage Reduction

To maximize Streaming Multiprocessor (SM) occupancy during long-context decoding, the block sequence is partitioned into independent *chunks* of c contiguous blocks. The primary scan kernel evaluates block thresholds and updates local online softmax accumulators ($m_{\text{mid}}^{(k)}$, $l_{\text{mid}}^{(k)}$, $O_{\text{mid}}^{(k)} \in \mathbb{R}^d$) independently for each chunk k. When $N_{\text{chunks}} > 1$, a secondary reduction kernel aggregates chunk statistics via associative online softmax reduction:

$$o = \frac{\sum_{k=1}^{N_{\text{chunks}}} O_{\text{mid}}^{(k)} \exp\left(m_{\text{mid}}^{(k)} - m_{\text{global}}\right)}{\sum_{k=1}^{N_{\text{chunks}}} l_{\text{mid}}^{(k)} \exp\left(m_{\text{mid}}^{(k)} - m_{\text{global}}\right)}, \quad \text{where } m_{\text{global}} = \max_k m_{\text{mid}}^{(k)}.$$

When $N_{\text{chunks}} = 1$, a fast path normalizes and writes $o = \text{acc}_1 / l_1$ directly to HBM without staging intermediate buffers. Larger chunk sizes c reduce intermediate buffer traffic and reduction trip counts at the cost of grid parallelism; as shown in Table 10, $b = 64, c = 128$ achieves optimal throughput at 51.2% union density, whereas coarser blocks ($b = 128, c = 64$) reach $2.50\times$ speedup in sparse regimes (16.0% union).

Table 10: Effect of block size (b) and chunk size (c) at batch size $B = 64$, $L = 512$K (32.8 GB KV cache), NVIDIA H100 SXM 80 GB, `float16`. c denotes blocks per chunk. Coarser blocks reduce metadata overhead in sparse regimes (16%), while $b = 64$ achieves the best balance at 51.2% union density.

	51.2% union		35.5% union		16.0% union	
Arm	ms	×	ms	×	ms	×
FlashAttention-2	15.085	—	15.009	—	14.790	—
ETA $b = 32, c = 64$	16.617	0.91	13.498	1.11	9.207	1.61
ETA $b = 64, c = 64$	13.797	1.09	10.566	1.42	6.429	2.30
ETA $b = 64, c = 128$	**13.432**	**1.12**	**10.430**	**1.44**	6.520	2.27
ETA $b = 128, c = 64$	16.733	0.90	11.472	1.31	**5.921**	**2.50**

F.4 Top-1 Rescue, Tail Handling, and Kernel Validation

If an aggressive threshold prunes all candidate blocks in the first chunk ($k = 1$), leaving an empty softmax denominator ($l_k = 0$), the kernel executes a localized *Top-1 Rescue Phase*: it tracks the maximum proxy bound index during scanning and unconditionally loads $K_{\text{max_idx}}, V_{\text{max_idx}}$ into SRAM. Furthermore, whenever sequence length T is not an exact multiple of block size b, the final chunk ($k = N_{\text{chunks}}$) executes a predicated tail phase (`IS_TAIL=True`) to process remaining tokens with out-of-bounds positions masked to $-\infty$.

We validate the grouped decode kernel across 31 test configurations against an independent PyTorch reference implementation modeling greedy block selection and top-1 rescue. Integer selection counts (per-head density, attended token fraction, and GQA union blocks) match with zero discrepancy across all cases, while attention outputs match within 10^{-6} in `float32` and 10^{-3} in `float16`. The complete decode procedure is summarized in Algorithm 4.

Algorithm 4 Hardware-Aligned Block-Sparse Decoding (Inference)

Require: Query $q \in \mathbb{R}^d$, KV Cache $K, V \in \mathbb{R}^{T \times d}$
Require: Block Size b, Blocks per Chunk c, Total Full Blocks $N_{\text{full}} \leftarrow \lfloor T/b \rfloor$
Require: Metadata: Centroids $\mu_j \in \mathbb{R}^d$, Variances $\sigma_j^2 \in \mathbb{R}^d$, Max Norms $M_j \in \mathbb{R}$ for $j \in [1, N_{\text{full}}]$
Require: Dynamic Threshold τ, Softmax Scale s, Operating Offset a, Z-score z
Ensure: Output $o \in \mathbb{R}^d$
1: Calculate $N_{\text{chunks}} \leftarrow \lceil N_{\text{full}}/c \rceil$
2: **if** $N_{\text{chunks}} > 1$ **then**

Allocate intermediate buffers $M_{\text{mid}}, L_{\text{mid}} \in \mathbb{R}^{N_{\text{chunks}}}$, $O_{\text{mid}} \in \mathbb{R}^{N_{\text{chunks}} \times d}$ in HBM
end if
$\tau_{\text{eff}} \leftarrow \tau - a$ ▷ Effective threshold with operating offset
for each chunk k from 1 to N_{chunks} **do** ▷ Processed in parallel across GPU SMs
  $m_k \leftarrow -\infty, \quad l_k \leftarrow 0, \quad acc_k \leftarrow \mathbf{0} \in \mathbb{R}^d, \quad$ tokens_attended $\leftarrow 0$
  max_bound $\leftarrow -\infty, \quad$ max_idx $\leftarrow -1$
  for each full block $j \in [(k-1)c + 1, \min(kc, N_{\text{full}})]$ **do**
    Load μ_j, σ_j^2, M_j from HBM into SRAM
    $\text{bound}_{\text{gauss}} \leftarrow s \cdot \left(\langle q, \mu_j \rangle + z \cdot \sqrt{\sum_i q_i^2 (\sigma_j^2)_i} \right)$
    $\text{bound}_{\text{spher}} \leftarrow s \cdot \|q\| \cdot M_j$
    $\text{bound} \leftarrow \min(\text{bound}_{\text{gauss}}, \text{bound}_{\text{spher}})$
    if bound $\geq \tau_{\text{eff}}$ **then**
      Load K_j, V_j from HBM into SRAM ▷ Significant block
      Update m_k, l_k, acc_k via gated online softmax ($\tilde{S}_j = S_j \odot \sigma(\beta(S_j - \tau_{\text{eff}}))$)
      tokens_attended $\leftarrow$ tokens_attended $+ b$
    end if
    if $k == 1$ **and** bound $>$ max_bound **then** ▷ Track max only for Chunk 1 rescue
      max_bound $\leftarrow$ bound, $\quad$ max_idx $\leftarrow j$
    end if
  end for
  if $k == 1$ **and** tokens_attended $== 0$ **then** ▷ Top-1 Rescue Phase (Chunk 1 only)
    Load $K_{\text{max_idx}}, V_{\text{max_idx}}$ from HBM into SRAM
    Update m_k, l_k, acc_k via gated online softmax using $q, K_{\text{max_idx}}, V_{\text{max_idx}}$
  end if
  if $k == N_{\text{chunks}}$ **and** $T > N_{\text{full}} \cdot b$ **then** ▷ Tail-Block Phase (partial remaining tokens)
    Load remaining tail tokens $K_{\text{tail}}, V_{\text{tail}}$ from HBM into SRAM
    Update m_k, l_k, acc_k via gated online softmax with partial masking
  end if
  if $N_{\text{chunks}} == 1$ **then**
    return acc_k / l_k ▷ Single-chunk fast path: return normalized output directly
  else
    Store unnormalized accumulators: $m_k \rightarrow M_{\text{mid}}[k], \quad l_k \rightarrow L_{\text{mid}}[k], \quad acc_k \rightarrow O_{\text{mid}}[k]$
  end if
end for
Global Reduction Kernel: ▷ Executed when $N_{\text{chunks}} > 1$
$m_{\text{global}} \leftarrow -\infty, \quad l_{\text{global}} \leftarrow 0, \quad o \leftarrow \mathbf{0} \in \mathbb{R}^d$
for each chunk k from 1 to N_{chunks} **do**
  $m_{\text{new}} \leftarrow \max(m_{\text{global}}, M_{\text{mid}}[k])$
  $\alpha_{\text{global}} \leftarrow \exp(m_{\text{global}} - m_{\text{new}}), \quad \alpha_{\text{chunk}} \leftarrow \exp(M_{\text{mid}}[k] - m_{\text{new}})$
  $l_{\text{global}} \leftarrow l_{\text{global}} \cdot \alpha_{\text{global}} + L_{\text{mid}}[k] \cdot \alpha_{\text{chunk}}$
  $o \leftarrow o \cdot \alpha_{\text{global}} + O_{\text{mid}}[k] \cdot \alpha_{\text{chunk}}$
  $m_{\text{global}} \leftarrow m_{\text{new}}$
end for
return o / l_{global} ▷ Final normalization across all chunks

G Train-to-Inference Representational Consistency

G.1 Theoretical Analysis of Block Pruning and Over-Inclusion

To build intuition for why SiLU tail tempering supports both block pruning and block over-inclusion, we analyze a simplified regime at decoding step t. Let $S_s = \frac{q_t k_s^\top}{\sqrt{d_k}}$ denote the raw pre-softmax logit for key $s \leq t$, $\tilde{\tau}_t = \tau_t - a$ the effective head threshold, and $\tilde{S}_s = S_s \sigma(\beta(S_s - \tilde{\tau}_t))$ the SiLU-tempered logit with weight $w_s := \exp(\tilde{S}_s)$.

At block-sparse inference, the causal context $\mathcal{S}_t = \{1, \ldots, t\}$ partitions into three disjoint sets:

1. $\mathcal{K}_t = \{s : S_s \geq \tilde{\tau}_t\}$: the salient active keys,

2. $\mathcal{C}_t \subset \mathcal{P}_t$: the co-admitted sub-threshold keys inside loaded blocks, and
3. $\mathcal{D}_t = \mathcal{P}_t \setminus \mathcal{C}_t$: the dropped sub-threshold keys inside pruned blocks,

where $\mathcal{P}_t = \mathcal{C}_t \sqcup \mathcal{D}_t$ is the collective sub-threshold tail.

Definition 1. *For any subset $\mathcal{X} \subseteq \mathcal{S}_t$, let $Z_{\mathcal{X}} := \sum_{s\in\mathcal{X}} w_s$ be its total weight and $\tilde{v}_{\mathcal{X}} := \frac{1}{Z_{\mathcal{X}}} \sum_{s\in\mathcal{X}} w_s v_s$ be its locally normalized value average. The token-sparse output o_t^{token} (retaining $\mathcal{K}_t$), block-sparse inference output o_t^{block} (retaining $\mathcal{K}_t \sqcup \mathcal{C}_t$), and training output o_t^{train} (retaining $\mathcal{S}_t = \mathcal{K}_t \sqcup \mathcal{C}_t \sqcup \mathcal{D}_t$) are defined as:*

$$
\begin{aligned}
o_t^{\text{token}} &:= \tilde{v}_{\mathcal{K}_t}, \\
o_t^{\text{block}} &:= \tilde{v}_{\mathcal{K}_t \cup \mathcal{C}_t} = \frac{Z_{\mathcal{K}} o_t^{\text{token}} + Z_{\mathcal{C}} \tilde{v}_{\mathcal{C}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}, \\
o_t^{\text{train}} &:= \tilde{v}_{\mathcal{S}_t} = \frac{Z_{\mathcal{K}} o_t^{\text{token}} + Z_{\mathcal{C}} \tilde{v}_{\mathcal{C}} + Z_{\mathcal{D}} \tilde{v}_{\mathcal{D}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}.
\end{aligned} \tag{14}
$$

We make two simplifying assumptions on the sub-threshold tail $\mathcal{P}_t$:

(A1) Bounded Tail Weight Spread: On any sub-threshold set $\mathcal{X} \subseteq \mathcal{P}_t$, the max-to-mean weight ratio is bounded:

$$\kappa = \frac{\max_{s\in\mathcal{X}} w_s}{\frac{1}{|\mathcal{X}|}\sum_{s\in\mathcal{X}} w_s} \geq 1.$$

We verify this empirically: $\kappa = 2.03$ for ETA vs. 15.6 for dense (Table 7).

(A2) Uncorrelated Tail Values: Sub-threshold value vectors $\{v_s\}_{s\in\mathcal{P}_t}$ have mean $\boldsymbol{\mu}_v = \mathbb{E}[v_s] = \mathbf{0}$, variance $\operatorname{tr}(\Sigma_v) = \mathbb{E}[\|v_s\|_2^2]$, and zero pairwise correlation $\mathbb{E}[\langle v_s, v_r\rangle \mid q_t] = 0$ for $s \neq r$ (verified empirically on C4 in Table 11).

We show that non-salient tokens can be safely ignored during inference:

Proposition 1. *Under **(A1)–(A2)**, for any sub-threshold set $\mathcal{X} \in \{\mathcal{C}_t, \mathcal{D}_t, \mathcal{P}_t\}$ of size $|\mathcal{X}|$, the expected squared norm of $\tilde{v}_{\mathcal{X}}$ satisfies:*

$$\frac{\operatorname{tr}(\Sigma_v)}{|\mathcal{X}|} \;\leq\; \mathbb{E}\big[\|\tilde{v}_{\mathcal{X}}\|_2^2\big] \;\leq\; \frac{\kappa \operatorname{tr}(\Sigma_v)}{|\mathcal{X}|}, \tag{15}$$

Furthermore, the minimum $\operatorname{tr}(\Sigma_v)/|\mathcal{X}|$ is uniquely attained by uniform weights ($w_s \equiv c, \kappa = 1$). In that case, we can substitute $\tilde{v}_{\mathcal{C}} \approx \mathbf{0}$ and $\tilde{v}_{\mathcal{D}} \approx \mathbf{0}$, so all three outputs are positive scalar multiples of o_t^{token} and coincide under RMSNorm:

$$\operatorname{RMSNorm}\big(o_t^{\text{train}}\big) \approx \operatorname{RMSNorm}\big(o_t^{\text{block}}\big) \approx \operatorname{RMSNorm}\big(o_t^{\text{token}}\big) \tag{16}$$

Proof. For any non-empty sub-threshold subset $\mathcal{X} \subseteq \mathcal{P}_t$, define the normalized tail weights $p_s := w_s/Z_{\mathcal{X}}$ for $s \in \mathcal{X}$, so that $p_s > 0$ and $\sum_{s\in\mathcal{X}} p_s = 1$.

We first decompose the variance conditioned on q_t:

$$
\begin{aligned}
\mathbb{E}\big[\|\tilde{v}_{\mathcal{X}}\|_2^2\big] &= \mathbb{E}\left[\left\langle \sum_{s\in\mathcal{X}} p_s v_s,\, \sum_{r\in\mathcal{X}} p_r v_r \right\rangle\right] \\
&= \sum_{s\in\mathcal{X}} p_s^2\, \mathbb{E}\big[\|v_s\|_2^2\big] + \sum_{\substack{s,r\in\mathcal{X} \\ s\neq r}} p_s p_r\, \mathbb{E}[\langle v_s, v_r\rangle \mid q_t] \\
&\overset{\text{(A2)}}{=} \operatorname{tr}(\Sigma_v) \sum_{s\in\mathcal{X}} p_s^2 = \operatorname{tr}(\Sigma_v)\, \frac{\sum_{s\in\mathcal{X}} w_s^2}{Z_{\mathcal{X}}^2}.
\end{aligned} \tag{17}
$$

The lower bound comes via the Cauchy–Schwarz inequality, which gives:

$$
\begin{aligned}
Z_{\mathcal{X}}^2 = \left(\sum_{s\in\mathcal{X}} w_s \cdot 1\right)^2 &\leq \left(\sum_{s\in\mathcal{X}} w_s^2\right)\left(\sum_{s\in\mathcal{X}} 1^2\right) = |\mathcal{X}| \sum_{s\in\mathcal{X}} w_s^2 \\
\implies \quad \frac{\sum_{s\in\mathcal{X}} w_s^2}{Z_{\mathcal{X}}^2} &\geq \frac{1}{|\mathcal{X}|},
\end{aligned}
$$

Here equality holds if and only if $(w_s)_{s\in\mathcal{X}}$ is collinear with $\mathbf{1}$, i.e., $w_s \equiv c$ for all $s \in \mathcal{X}$ ($\kappa = 1$). Substituting this into Equation 17 gives the lower bound $\mathbb{E}[\|\tilde{v}_{\mathcal{X}}\|_2^2] \geq \mathrm{tr}(\Sigma_v)/|\mathcal{X}|$.

For the upper bound we rely on **(A1)**, to obtain that

$$\sum_{s\in\mathcal{X}} w_s^2 \leq w_{\max} \sum_{s\in\mathcal{X}} w_s = \left(\frac{\kappa}{|\mathcal{X}|} Z_{\mathcal{X}}\right) Z_{\mathcal{X}} = \frac{\kappa}{|\mathcal{X}|} Z_{\mathcal{X}}^2$$

$$\implies \quad \frac{\sum_{s\in\mathcal{X}} w_s^2}{Z_{\mathcal{X}}^2} \leq \frac{\kappa}{|\mathcal{X}|}.$$

Combining the above establishes Equation 15.

Finally, under SiLU tail tempering, both $\tilde{v}_{\mathcal{C}}$ and $\tilde{v}_{\mathcal{D}}$ concentrate around $\mathbf{0}$ at rate $\mathcal{O}(\sqrt{\kappa/|\mathcal{X}|})$. Substituting $\tilde{v}_{\mathcal{C}} \approx \mathbf{0}$ and $\tilde{v}_{\mathcal{D}} \approx \mathbf{0}$ into Equation 4 leaves o_t^{train} and o_t^{block} collinear with o_t^{token}:

$$o_t^{\text{train}} \approx \alpha_{\text{train}}\, o_t^{\text{token}}, \qquad \alpha_{\text{train}} := \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}} > 0,$$

$$o_t^{\text{block}} \approx \alpha_{\text{block}}\, o_t^{\text{token}}, \qquad \alpha_{\text{block}} := \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}} > 0.$$

Because $\mathrm{RMSNorm}(x) = \frac{x}{\|x\|_{\text{rms}}} \odot \gamma$ with $\|x\|_{\text{rms}} = \sqrt{\frac{1}{d}\sum_{i=1}^d x_i^2}$ is invariant to scaling by constants we have

$$\mathrm{RMSNorm}\big(\alpha\, o_t^{\text{token}}\big) = \frac{\alpha\, o_t^{\text{token}}}{\alpha\, \|o_t^{\text{token}}\|_{\text{rms}}} \odot \gamma = \mathrm{RMSNorm}\big(o_t^{\text{token}}\big),$$

which yields Equation 16. □

Next, we show that co-admitting sub-threshold keys $\mathcal{C}_t$ inside loaded GPU blocks moves the pre-norm attention output closer to the training output under ETA tempering, whereas it introduces output error under additive hard masking:

Proposition 2. *Suppose tail value averages cancel ($\tilde{v}_{\mathcal{C}} = \tilde{v}_{\mathcal{D}} = \mathbf{0}$) and $\|o_t^{\text{token}}\|_2 > 0$. When loaded GPU blocks co-admit sub-threshold keys $\mathcal{C}_t$ ($Z_{\mathcal{C}} > 0$) alongside pruned blocks $\mathcal{D}_t$ ($Z_{\mathcal{D}} > 0$):*

1. ***ETA Tempering** ($o_t^{\text{train}} = \tilde{v}_{\mathcal{S}_t}$): Block-sparse inference o_t^{block} lies strictly closer to the training output o_t^{train} than exact token pruning o_t^{token}:*

$$\big\|o_t^{\text{block}} - o_t^{\text{train}}\big\|_2 \;<\; \big\|o_t^{\text{token}} - o_t^{\text{train}}\big\|_2. \tag{18}$$

2. ***Additive Hard Masking** ($o_t^{\text{train,add}} = o_t^{\text{token}}$): Because sub-threshold keys receive zero weight during training ($Z_{\mathcal{C}}^{\text{train}} = Z_{\mathcal{D}}^{\text{train}} = 0$), co-admitting $\mathcal{C}_t$ at inference strictly* increases *output error:*

$$\big\|o_t^{\text{block}} - o_t^{\text{train,add}}\big\|_2 \;>\; \big\|o_t^{\text{token}} - o_t^{\text{train,add}}\big\|_2 = 0. \tag{19}$$

Proof. Under tail value cancellation ($\tilde{v}_{\mathcal{C}} = \tilde{v}_{\mathcal{D}} = \mathbf{0}$), Equation 4 simplifies the token-sparse, block-sparse, and ETA training outputs to positive scalar multiples of o_t^{token}:

$$o_t^{\text{token}} = 1 \cdot o_t^{\text{token}}, \qquad o_t^{\text{block}} = \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}\, o_t^{\text{token}}, \qquad o_t^{\text{train}} = \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}\, o_t^{\text{token}}. \tag{20}$$

Since $Z_{\mathcal{K}}, Z_{\mathcal{C}}, Z_{\mathcal{D}} > 0$, the partition denominators satisfy $Z_{\mathcal{K}} < Z_{\mathcal{K}} + Z_{\mathcal{C}} < Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}$. We thus have:

$$0 \;<\; \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}} \;<\; \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}} \;<\; 1.$$

Thus o_t^{block} interpolates strictly between o_t^{train} and o_t^{token} along o_t^{token}, so:

$$\begin{aligned}
\big\|o_t^{\text{block}} - o_t^{\text{train}}\big\|_2 &= \left|\frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}} - \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}\right| \|o_t^{\text{token}}\|_2 \\
&= \left(\frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}} - \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}\right) \|o_t^{\text{token}}\|_2 \\
&< \left(1 - \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}} + Z_{\mathcal{D}}}\right) \|o_t^{\text{token}}\|_2 = \big\|o_t^{\text{token}} - o_t^{\text{train}}\big\|_2,
\end{aligned}$$

which establishes Equation 18.

Under additive hard masking, exact token pruning matches the training output ($\|o_t^{\text{token}} - o_t^{\text{train,add}}\|_2 = 0$), whereas co-admitting $\mathcal{C}_t$ inside loaded blocks ($Z_{\mathcal{C}} > 0$) shifts o_t^{block} away from $o_t^{\text{train,add}}$:

$$\begin{aligned}\left\|o_t^{\text{block}} - o_t^{\text{train,add}}\right\|_2 &= \left\|\frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}\, o_t^{\text{token}} - o_t^{\text{token}}\right\|_2 \\ &= \left(1 - \frac{Z_{\mathcal{K}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}\right) \|o_t^{\text{token}}\|_2 \\ &= \frac{Z_{\mathcal{C}}}{Z_{\mathcal{K}} + Z_{\mathcal{C}}}\, \|o_t^{\text{token}}\|_2 \;>\; 0 = \left\|o_t^{\text{token}} - o_t^{\text{train,add}}\right\|_2,\end{aligned}$$

which establishes Equation 19. □

G.2 Empirical Verification of Residual Stream Alignment and Proposition 1

We empirically test the predictions of Proposition 1 across model scales on held-out C4 sequences. Specifically, we examine two theoretical claims:

1. **Tail Decay Rate:** The centered pruned-tail value average $\|\bar{v}_{\mathcal{P}_t} - \boldsymbol{\mu}_v\|_2 / \sqrt{\text{Tr}(\Sigma_v)}$ decays at a rate of $1/\sqrt{m}$ as the pruned set size $m = |\mathcal{P}_t|$ increases.
2. **Radial Scaling and Direction Preservation:** Omitting pruned blocks $\mathcal{D}_t$ (while retaining SiLU-tempered weights on loaded blocks $\mathcal{K}_t \sqcup \mathcal{C}_t$) scales the pre-norm attention output radially by $(\alpha_{\text{block}}/\alpha_{\text{train}})$ while preserving the residual direction ($\mathcal{S}_{\cos} \approx 1$).

To validate these behaviors, we evaluate both the 12-layer 126M and 22-layer 1.45B ETA models ($b = 64$, `lwb` $= 2$, sub-block envelope).

Direct Empirical Verification of Proposition 1 on the 1.45B Model. For each head and layer of the 1.45B model during C4 decoding, we extract the pruned value vectors $\{v_j\}_{j \in \mathcal{P}_t}$, center them by the sequence value mean $\boldsymbol{\mu}_v$, and measure the normalized Euclidean norm $\|\bar{v}_{\mathcal{P}_t} - \boldsymbol{\mu}_v\|_2 / \sqrt{\text{Tr}(\Sigma_v)}$ as a function of the pruned set size $m = |\mathcal{P}_t| \in \{16, 32, 64, 128, 256, 384\}$.

As reported in Table 11, the empirical norm tracks the theoretical $1/\sqrt{m}$ bound closely at small m and decays even faster than $1/\sqrt{m}$ at larger m. Simultaneously, the active partition fraction satisfies $\alpha_{\text{train}} = \mathbf{0.9973 \pm 0.0153}$ (median $= 1.0000$, 25th percentile $= 0.9998$), confirming that $(1 - \alpha_{\text{train}})\|\bar{v}_{\mathcal{P}_t}\|_2 \ll 10^{-3}$.

Table 11: Empirical verification of Proposition 1 ($1/\sqrt{m}$ background value cancellation) on the 22-layer 1.45B ETA model. Normalized centered norm $\|\bar{v}_{\mathcal{P}_t} - \boldsymbol{\mu}_v\|_2 / \sqrt{\text{Tr}(\Sigma_v)}$ measured across pruned set sizes $m = |\mathcal{P}_t|$ on C4, compared against the theoretical $1/\sqrt{m}$ rate.

Pruned Set Size $m = \lvert\mathcal{P}_t\rvert$	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 256$	$m = 384$
Theoretical Rate $(1/\sqrt{m})$	0.2500	0.1768	0.1250	0.0884	0.0625	0.0510
Empirical 1.45B Norm	**0.2504**	**0.1750**	**0.1202**	**0.0799**	**0.0448**	**0.0274**

Layer-by-Layer Residual Stream Alignment (126M 12-Layer and 1.45B 22-Layer Models). Let $\mathbf{x}_{\text{train}}^{(l)}$ and $\mathbf{x}_{\text{infer}}^{(l)}$ denote the residual stream hidden states at layer l produced by full-sequence SiLU-gated attention ($\mathcal{S}_t$) and block-sparse SiLU-gated decoding ($\mathcal{K}_t \sqcup \mathcal{C}_t$), respectively. At each decoding step t, we compute the directional cosine similarity and the relative Euclidean norm shift:

$$\mathcal{S}_{\cos}(l) = \frac{\langle \mathbf{x}_{\text{train}}^{(l)}, \mathbf{x}_{\text{infer}}^{(l)} \rangle}{\|\mathbf{x}_{\text{train}}^{(l)}\|_2 \, \|\mathbf{x}_{\text{infer}}^{(l)}\|_2}, \qquad \mathcal{E}_{\ell_2}(l) = \frac{\|\mathbf{x}_{\text{train}}^{(l)} - \mathbf{x}_{\text{infer}}^{(l)}\|_2}{\|\mathbf{x}_{\text{train}}^{(l)}\|_2}. \tag{21}$$

As shown in Figure 7 and Table 12, the empirical measurements confirm our theoretical decomposition at both the 126M (12-layer) and 1.45B (22-layer) scales:

- **22-Layer 1.45B Alignment:** Across all 22 layers ($L0$–$L21$), the 1.45B residual stream maintains an overall mean directional cosine similarity of $\mathcal{S}_{\cos} = \mathbf{0.9917}$ (1.0000 at $L0$–$L2$, 0.9910 at $L9$,

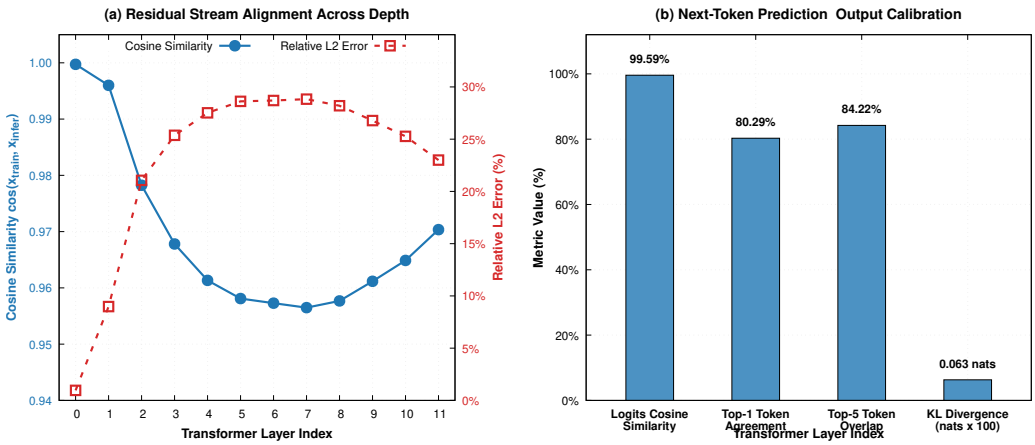


Figure 7: Internal representation and output distribution alignment between full-sequence SiLU-gated attention ($\mathcal{S}_t$) and block-sparse decoding ($\mathcal{K}_t \sqcup \mathcal{C}_t$). (a) Layer-by-layer residual stream trajectory alignment across network depth. (b) Final language modeling head calibration metrics.

and 0.9888 at $L21$), while the final language modeling head achieves a logit cosine similarity of $\mathbf{0.9944}$ and a KL divergence of only $\mathbf{0.0379}$ nats.

- **12-Layer 126M Alignment:** Under aggressive sparsity ($\lambda = 0.05$), the 126M model maintains a mean directional cosine similarity of $\mathcal{S}_{\cos} = \mathbf{0.9691}$ (0.9997 at Layer 0) with a radial norm shift of $\mathcal{E}_{\ell_2} = \mathbf{22.77}\%$ absorbed by $\mathrm{RMSNorm}$, achieving a logit cosine similarity of $\mathbf{0.9959}$ and a KL divergence of $\mathbf{0.0631}$ nats.

Table 12: Layer-wise residual stream alignment ($\mathcal{S}_{\cos}$) and relative ℓ_2 shift ($\mathcal{E}_{\ell_2}$) across both the 12-layer 126M and 22-layer 1.45B ETA models on held-out C4 sequences during autoregressive block-sparse decoding.

126M Model (Layers 0–11)			**1.45B Model (Layers 0–21, $b = 64$ Sub-Block Envelope + `lwb` $= 2$)**					
Layer	$\mathcal{S}_{\cos}$	$\mathcal{E}_{\ell_2}$	**Layer**	$\mathcal{S}_{\cos}$	$\mathcal{E}_{\ell_2}$	**Layer**	$\mathcal{S}_{\cos}$	$\mathcal{E}_{\ell_2}$
Layer 0	0.9997	0.96%	Layer 0	1.0000	0.01%	Layer 11	0.9881	13.56%
Layer 1	0.9960	8.98%	Layer 1	1.0000	0.34%	Layer 12	0.9867	14.19%
Layer 2	0.9782	21.06%	Layer 2	1.0000	0.41%	Layer 13	0.9863	14.38%
Layer 3	0.9678	25.37%	Layer 3	0.9999	1.26%	Layer 14	0.9860	14.53%
Layer 4	0.9613	27.52%	Layer 4	0.9997	2.19%	Layer 15	0.9863	14.19%
Layer 5	0.9581	28.62%	Layer 5	0.9997	2.35%	Layer 16	0.9856	14.47%
Layer 6	0.9573	28.70%	Layer 6	0.9975	6.62%	Layer 17	0.9854	14.46%
Layer 7	0.9565	28.83%	Layer 7	0.9932	10.22%	Layer 18	0.9858	14.22%
Layer 8	0.9577	28.20%	Layer 8	0.9925	10.69%	Layer 19	0.9865	13.96%
Layer 9	0.9612	26.78%	Layer 9	0.9910	11.80%	Layer 20	0.9882	13.20%
Layer 10	0.9649	25.27%	Layer 10	0.9891	12.82%	Layer 21	0.9888	12.83%
Layer 11	0.9704	23.00%		—		**1.45B Mean**	**0.9917**	**9.94%**
126M Mean	**0.9691**	**22.77%**	**1.45B Logit Cos: 0.9944**			**KL Div: 0.0379** nats		**Top-1 Match: 100.0%**

H Mechanistic Analysis: Elimination of Attention Sinks and Layer Specialization

To analyze ETA's behavior across network depth and sequence position, we record active token densities and initial-token softmax mass ($t \in [0, 3]$) across all 22 layers and 32 query heads (704 heads total) of our 1.45B model on held-out FineWeb sequences ($L = 2048$).

Mathematical Mechanism of Attention Sink Elimination. In standard dense or additive hard-masked attention ($S_{ts} \to -\infty$ for $s \notin \mathcal{I}_t$), attention probabilities must sum to unity over the active set ($\sum_{s \in \mathcal{I}_t} P_{ts} = 1$). When a query q_t has no relevant context matches, softmax still forces 100% of the probability mass onto active keys. To prevent noisy context mixtures during such "no-op" steps, standard Transformers route unused mass onto initial tokens ($t \in [0, 3]$), forming persistent *attention sinks* (Xiao et al., 2024) that absorb over 50% of total attention mass (Figure 4, Left; Subsection 4.6). Consequently, post-hoc eviction methods (e.g., StreamingLLM, H_2O) collapse unless initial sink tokens are heuristically pinned.

Under ETA's multiplicative gating ($\tilde{S}_{ts} = S_{ts} \cdot \sigma(\beta(S_{ts} - \tau_t))$), sub-threshold logits contract toward 0 rather than $-\infty$. Because $\exp(0) = 1$, the training-time partition function Z_t decomposes as:

$$Z_t = \sum_{s \in \mathcal{I}_t} \exp(\tilde{S}_{ts}) + \sum_{s \notin \mathcal{I}_t} \exp(0) = \sum_{s \in \mathcal{I}_t} \exp(\tilde{S}_{ts}) + (t - |\mathcal{I}_t|).$$

The $(t - |\mathcal{I}_t|)$ sub-threshold keys act as a built-in, uniform probability reservoir ($\tilde{P}_{ts} \approx 1/t$) whenever τ_t rises on uninformative queries, removing any optimization pressure to create localized sinks on initial tokens. As shown in Figure 4 (Left), average probability mass on tokens $t \in [0, 3]$ drops from $> 50\%$ in dense attention to $< 2\%$ under ETA across all sequence lengths, without heuristic sink pinning.

Autonomous Layer Specialization Across Depth. The 22×32 layer-head density profile (Figure 4, Right; Subsection 4.6) shows that ETA organizes its attention budget into three stages without layer-specific tuning:

1. **Broad Surface Aggregation (Layer 0):** Retains $\approx 96\%$ active density across heads to mix local and global surface features.
2. **Mid-Network Compression (Layers 3–14):** Compresses context sharply (reaching $< 8\%$ mean density at Layer 7), where heads specialize into targeted retrieval and syntactic routing.
3. **Deep Predictive Synthesis (Layers 15–21):** Restores density to 35%–48% to aggregate distributed representations for next-token prediction.

I NSA VS. ETA: EXPERIMENTAL DETAILS

Both 126M models were pretrained from scratch on FineWeb for 10,000 steps at sequence length $L = 1024$ on a single NVIDIA H100 80GB GPU (effective batch size 16 sequences / 16,384 tokens per step, 163.8M tokens total). We use AdamW ($\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 10^{-8}$, weight decay 0.1) with cosine learning rate decay (peak lr $= 3 \times 10^{-4}$, 500 warmup steps). ETA includes our sparsity regularizer ($\lambda = 0.05$), while NSA (Yuan et al., 2025) trains with standard cross-entropy. To isolate the attention mechanism, both models share an identical 12-layer LLaMA-style backbone (Table 13).

Table 13: Backbone Architectural Hyperparameters. All backbone dimensions and positional encodings are identical between the two models. ETA adds $12 \times (768 \times 12 + 12) = 0.11$M threshold-predictor parameters ($< 0.1\%$ of total parameters).

Hyperparameter	NSA 126M	ETA 126M
Layers (L) / Hidden Size (d_{model}) / FFN (d_{ffn})	12 / 768 / 2048	12 / 768 / 2048
Query Heads (H_Q) / KV Heads (H_{KV}) / Head Dim (d_k)	12 / 4 / 64	12 / 4 / 64
Total Parameters (excl. output embedding)	125.85M (100.66M)	125.85M (100.66M)
Vocabulary Size / RoPE Base (θ)	32,768 / 5×10^5 (LLaMA-3 8×)	32,768 / 5×10^5 (LLaMA-3 8×)

I.1 TRAINING CONVERGENCE AND WALL-CLOCK DYNAMICS

As shown in Figure 8, ETA converges faster and to a lower cross-entropy loss than NSA (**4.370–4.399**, PPL $= 81.37$ vs. **4.508**, PPL $= 90.78$). On identical H100 hardware, ETA's fused Triton forward/backward kernel matches NSA's training step time within 0.08% (**0.5315** s/step vs. **0.5311** s/step), while avoiding NSA's three separate attention branches (compression, selection, and sliding window) and auxiliary gating MLPs.

I.2 AUTOREGRESSIVE DOWNSTREAM PERPLEXITY

We evaluate both checkpoints at $L = 2048$ ($2 \times L_{\text{train}}$) under 1% prompt prefill and 99% autoregressive KV-cached decoding across FineWeb, C4, and WikiText-2 ($a = 0.4, z = 2.0$; Table 14). ETA achieves lower perplexity on FineWeb (99.80 vs. 103.55) and WikiText-2 (158.64 vs. 161.65) while pruning $\approx 53\%$ of attention compute, and matches NSA on C4 (100.73 vs. 100.18).

I.3 AUTOREGRESSIVE DECODING LATENCY AND THROUGHPUT

We benchmark decoding latency and throughput across batch sizes $B \in \{16, 32\}$ and context lengths up to $L = 32$K (10% prefill / 90% decode; Table 15). By replacing NSA's multi-branch kernel launches with a single fused block-sparse pass, ETA reduces per-step decode latency across all

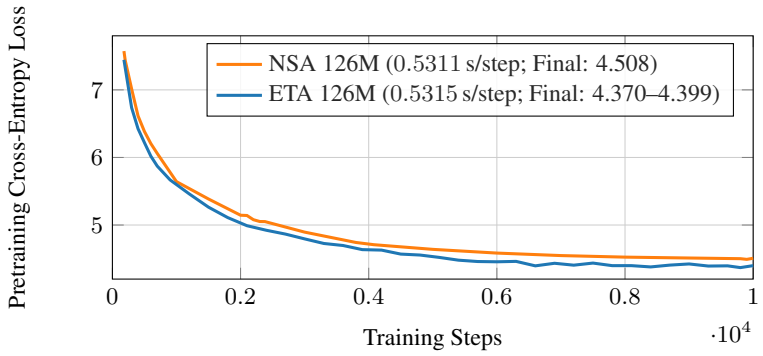


Figure 8: Pretraining Loss Convergence and Matched-H100 Throughput. Training loss trajectories across 10,000 steps (164M tokens, $L = 1024$) on identical H100 GPUs. ETA achieves $\Delta\text{Loss} = -0.11$ to -0.14 ($\Delta\text{PPL} = -9.4$) below NSA at matched step latency (0.5315 s/step vs. 0.5311 s/step).

Table 14: Perplexity (PPL) and Sparsity Comparison. 126M models, evaluated under 1% prompt prefill + 99% autoregressive decode (batch size 4, $L = 2048$).

Dataset	NSA PPL	ETA PPL	Δ PPL	ETA Sparsity
FineWeb (10BT)	103.55 ± 26.51 [± 1.87]	$\mathbf{99.80 \pm 21.71}$ [± 1.54]	$\mathbf{-3.75}$	**53.60%**
C4	$\mathbf{100.18 \pm 16.08}$ [± 1.14]	100.73 ± 17.96 [± 1.27]	$+0.55$	**53.89%**
Wikitext-2	161.65 ± 22.50 [± 1.24]	$\mathbf{158.64 \pm 22.30}$ [± 1.17]	$\mathbf{-3.01}$	**51.17%**

configurations—reaching a 2.72× speedup at $B = 32, L = 8$K (25.35 ms vs. 69.07 ms; 1,262 vs. 463 tok/s) and a 1.63× speedup at $B = 16, L = 32$K (37.36 ms vs. 60.93 ms).

Table 15: Decoding Efficiency & Throughput Sweep. Evaluated under 10% prompt prefill + 90% autoregressive decode across sequence lengths up to 32,768 tokens.

Batch (B)	Context (L)	Model	Prefill (TTFT)	Step Latency (ms)	Throughput (tok/s)
16	16,384 (16K)	NSA	0.992 s	61.319 ± 2.248	260.93
16	16,384 (16K)	**ETA**	**0.319 s**	$\mathbf{24.254 \pm 6.086}$	**659.68**
16	32,768 (32K)	NSA	1.724 s	60.932 ± 2.927	262.59
16	32,768 (32K)	**ETA**	**0.649 s**	$\mathbf{37.356 \pm 12.475}$	**428.31**
32	8,192 (8K)	NSA	0.174 s	69.069 ± 1.573	463.31
32	8,192 (8K)	**ETA**	0.294 s	$\mathbf{25.353 \pm 5.808}$	**1,262.16**
32	16,384 (16K)	NSA	0.407 s	69.102 ± 1.359	463.09
32	16,384 (16K)	**ETA**	0.502 s	$\mathbf{37.737 \pm 12.720}$	**847.98**
32	32,768 (32K)	NSA	0.465 s	66.193 ± 4.728	483.44
32	32,768 (32K)	**ETA**	1.149 s	$\mathbf{61.291 \pm 23.937}$	**522.10**

J Extended Block Granularity, Pareto, and RULER Benchmarks

This appendix provides supplementary figures and full sweep tables for Section 4.3 and Section 4.4.

J.1 GQA Union Loading and Variance Dilution at Coarse Block Sizes

In Grouped-Query Attention (GQA), each KV head is shared by $G = H_Q/H_{KV}$ query heads, so the GPU loads the union of active blocks $\mathcal{U}_g = \bigcup_{q \in \mathcal{G}_g} \mathcal{S}_q$ from HBM. On the 126M model ($G = 3$), Figure 9 shows that the active group union expands by only 1.49× over per-head compute density (50.8% union vs. 34.1% head density), as query heads sharing a KV head attend to overlapping context spans.

While larger blocks ($b = 64$) reduce GQA union overhead and align with Tensor Core tiles (Table 2), computing a single flat variance $\sigma^2_{B,i} = \frac{1}{b}\sum_{k \in B}(k_i - \mu_{B,i})^2$ across all 64 tokens dilutes localized 4-token spikes by up to 16×. As shown in Table 16, scaling b from 4 to 64 under flat variance increases block false negatives from 3.4% to 27.5% and missed softmax mass from 10.2% to 36.3%.

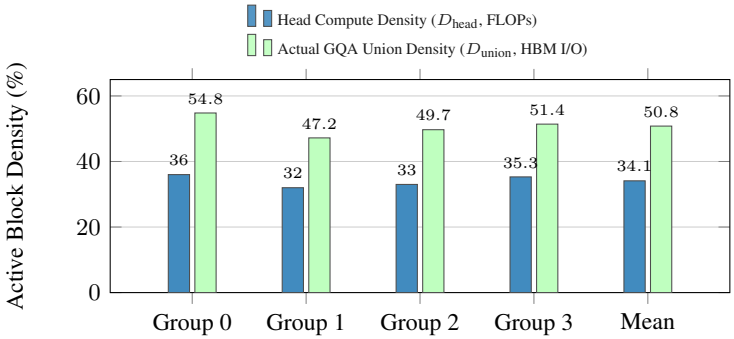


Figure 9: GQA Union Loading vs. Compute Sparsity across KV Groups on C4 decoding (126M model).

Table 16: Effect of variance dilution under flat block variance ($z = 2.0, a = 0.4$) before applying the sub-block envelope (1.45B model on FineWeb, $L = 2048$). Missed mass is the fraction of softmax probability mass in rejected blocks.

b	False neg.	False pos.	Token recall	Missed mass	PPL
4	**3.4**%	30.8%	**98.3**%	**10.2**%	**17.38**
16	11.7%	16.8%	96.9%	21.5%	18.47
64	27.5%	**5.6**%	95.3%	36.3%	19.15

Sub-Block Variance Envelope. To eliminate the $b = 64$ variance dilution shown in Table 16 without increasing the $(1, d_h)$ metadata footprint per block, we partition B into 16 contiguous 4-token sub-blocks $B_1, \ldots, B_{16}$ ($b_{\text{sub}} = 4$), store the coordinate-wise *sub-block standard deviation*

$$\tilde{\sigma}_{B,i} = \max_{1 \le m \le b/b_{\text{sub}}} \sqrt{\frac{1}{b_{\text{sub}}} \sum_{k \in B_m} (k_i - \mu_{B,i})^2}, \tag{22}$$

and pin the two most recent blocks (`lwb` $= 2$, 128 tokens). Replacing flat $\boldsymbol{\sigma}_B$ with $\tilde{\boldsymbol{\sigma}}_B$ at $b = 64$ recovers FineWeb perplexity from 19.15 (Table 16) down to **17.11** (Table 1), and on C4 ($a = -0.4$) boosts token recall from 76.8% (flat variance, 21.02 PPL) to **97.3**% (**18.60** PPL; Table 3).

J.2 Extended C4 Perplexity and RULER Retrieval Tables

Table 17 provides the full offset sweep supplementing Table 3 in Subsection 4.3. For long-context retrieval (Subsection 4.4), Table 18 reports the complete 16-arm RULER suite across all four tasks at $L = 2048$ and $L = 4096$ ($N = 432$ trials/arm), and Table 19 reports single-needle passkey extrapolation from $L = 1024$ to $L = 8192$ ($4 \times L_{\text{train}}$).

Table 17: Extended C4 Perplexity ($L = 2048$) **and Needle-in-a-Haystack** ($L = 4096$) **Pareto sweep** on the 1.45B model across block size b and threshold offset a, supplementing Table 3.

Method Variant	Offset / Budget	Head (D_{head})	Union (D_{union})	Recall / FNR	C4 PPL (↓)	Needle 4K (↑)
Dense Baseline	`FlashAttention-2`	100.0%	100.0%	100.0% / 0.0%	17.49 ± 0.33	66.7% (6/9)
ETA ($b = 4, z = 2.0$)	$a = +0.4$	36.3%	66.6%	98.3% / 7.1%	$\mathbf{17.56 \pm 0.33}$	**66.7**% (6/9)
	$a = 0.0$	32.0%	61.2%	97.8% / 8.9%	17.61 ± 0.33	**66.7**% (6/9)
	$a = -0.8$	24.4%	50.6%	97.1% / 12.1%	17.82 ± 0.33	**66.7**% (6/9)
	$a = -1.5$	18.9%	41.7%	96.0% / 16.1%	18.22 ± 0.33	**66.7**% (6/9)
	$a = -2.0$	15.7%	35.6%	95.4% / 18.8%	18.81 ± 0.34	**66.7**% (6/9)
	$a = -2.5$	**13.1**%	**30.1**%	94.8% / 21.2%	19.81 ± 0.34	**66.7**% (6/9)
ETA ($b = 64$, Sub-Blk + `lwb` $= 2$)	$a = +0.2$	55.3%	79.0%	**98.4**% / 12.3%	18.30 ± 0.29	**66.7**% (6/9)
	$a = -0.4$	48.3%	69.7%	97.3% / 16.6%	18.60 ± 0.29	**66.7**% (6/9)
	$a = -1.0$	42.7%	60.4%	95.6% / 21.3%	18.87 ± 0.29	**66.7**% (6/9)
	$a = -1.8$	37.6%	49.6%	91.6% / 28.3%	19.21 ± 0.30	**66.7**% (6/9)
H_2O Eviction	$W = 256$ (≈38% target)	48.0%	68.0%	—	17.59 ± 0.33	22.2% (2/9)
	$W = 128$ (≈25% target)	33.2%	54.3%	—	17.71 ± 0.33	0.0% (0/9)
	$W = 64$ (≈10% target)	17.5%	28.8%	—	17.91 ± 0.33	0.0% (0/9)

K Dual Probabilistic-Geometric Screening Index: Mathematical Analysis

This section derives the finite-population concentration and geometric bounds governing the Dual Screening Index in Section 3 (Equation 7).

Table 18: RULER Long-Context Suite on the 1.45B model ($L = 2048$ Native & $L = 4096$ with $2\times$ NTK RoPE Scaling; $N = 432$ trials/arm). Reports per-task and average accuracy ($\pm 95\%$ CI), plus routine haystack vs. query-step density (Routine → Query).

Context	Method	Head (Routine → Query)	Union (Routine → Query)	Single	MultiKey	MultiVal	VT (2-Hop)	RULER Avg
$L = 2048$	Dense Baseline	100.0%	100.0%	87.0%	76.9%	88.9%	87.0%	84.95% ± 3.19%
	ETA ($b = 4, z = 1.0, a = 0.4$)	38.0% → 41.0%	66.3% → 69.8%	**88.0%**	87.0%	**88.9%**	**88.0%**	**87.96% ± 2.86%**
	ETA ($b = 64, z = 1.5, a = 0.4$)	40.6% → 43.5%	54.6% → 57.9%	87.0%	**88.0%**	**88.9%**	77.8%	85.42% ± 3.14%
	ETA ($b = 64, z = 1.5, a = 0.0$)	25.1% → 27.8%	**37.4%** → **39.8%**	87.0%	76.9%	78.7%	67.6%	77.55% ± 3.78%
	Quest ($b = 64$, top-38%)	39.1%	51.2%	55.6%	23.1%	12.0%	13.0%	25.93% ± 4.14%
	Quest ($b = 64$, top-25%)	**23.1%**	32.7%	34.3%	12.0%	0.9%	0.9%	12.04% ± 3.07%
	H_2O ($W = 256$, ∼38%)	38.3%	72.3%	25.0%	12.0%	0.9%	0.9%	9.72% ± 2.80%
	H_2O ($W = 256$, ∼25%)	25.3%	50.8%	13.9%	11.1%	0.9%	0.0%	6.48% ± 2.33%
$L = 4096$	Dense Baseline	100.0%	100.0%	66.7%	65.7%	56.5%	33.3%	55.56% ± 4.14%
	ETA ($b = 4, z = 1.0, a = 0.4$)	38.0% → 44.5%	66.3% → 73.9%	**66.7%**	**65.7%**	**56.5%**	**33.3%**	**55.56% ± 4.14%**
	ETA ($b = 64, z = 1.5, a = 0.4$)	38.2% → 44.6%	51.5% → 58.7%	**66.7%**	**65.7%**	55.6%	24.1%	53.01% ± 4.16%
	ETA ($b = 64, z = 1.5, a = 0.0$)	24.8% → 31.1%	**36.9%** → **41.2%**	56.5%	55.6%	50.9%	23.1%	46.53% ± 4.16%
	Quest ($b = 64$, top-38%)	38.3%	50.9%	24.1%	13.0%	1.9%	0.9%	9.95% ± 2.82%
	Quest ($b = 64$, top-25%)	**24.1%**	34.5%	13.0%	1.9%	0.0%	0.0%	3.70% ± 1.78%
	H_2O ($W = 256$, ∼38%)	38.1%	74.6%	3.7%	1.9%	0.9%	0.0%	1.62% ± 1.20%
	H_2O ($W = 256$, ∼25%)	25.1%	56.7%	1.9%	0.9%	0.0%	0.0%	0.69% ± 0.78%

Table 19: Single-Needle passkey retrieval accuracy across context lengths up to $L = 8192$ ($4 \times L_{\text{train}}$). Accuracy averaged over 9 needle depths (0%–100%). Density reports position-averaged head density at $L = 1024 \rightarrow 8192$.

Variant	Config	Density (%)	1024 Acc	2048 Acc	4096 Acc	8192 Acc
SDPA	`Dense Baseline`	100.0	**100.0**	**88.9**	**66.7**	*0.0*
H2O	`W=256, HH=10%`	29.0→11.5	22.2	11.1	*0.0*	*0.0*
BigBird	`W=256, Str=16`	30.0→9.2	22.2	11.1	*0.0*	*0.0*
StreamLLM	`W=256, S=4`	25.4→3.2	22.2	11.1	*0.0*	*0.0*
SWA	`W=256`	25.0→3.1	22.2	11.1	*0.0*	*0.0*
ETA	`b=4, a=0.4`	30.1→69.3	**100.0**	**88.9**	**66.7**	**22.2**

K.1 Cantelli Bound and Certified Finite-Population Maximums

Consider a key block $B = \{k_1, \ldots, k_b\} \subset \mathbb{R}^{d_h}$ with empirical mean $\boldsymbol{\mu}_B = \frac{1}{b}\sum_{j=1}^{b} k_j$ and covariance $\Sigma_B = \frac{1}{b}\sum_{j=1}^{b}(k_j - \boldsymbol{\mu}_B)(k_j - \boldsymbol{\mu}_B)^\top$. Drawing $k \sim \mathrm{Unif}(B)$ uniformly at random yields a scalar inner product $X = \langle q, k\rangle$ with mean $\mathbb{E}[X] = \langle q, \boldsymbol{\mu}_B\rangle$ and variance $\mathrm{Var}(X) = q^\top \Sigma_B\, q$. By Cantelli's one-sided inequality, for any $z > 0$:

$$\Pr_{k\sim\mathrm{Unif}(B)}\left[\langle q, k\rangle \geq \langle q, \boldsymbol{\mu}_B\rangle + z\sqrt{q^\top \Sigma_B\, q}\right] \leq \frac{1}{1+z^2}. \tag{23}$$

Since B has finite cardinality b, multiplying by b and taking the floor bounds the exact number of keys exceeding the threshold:

$$\left|\left\{k \in B : \langle q, k\rangle \geq \langle q, \boldsymbol{\mu}_B\rangle + z\sqrt{q^\top \Sigma_B\, q}\right\}\right| \leq \left\lfloor \frac{b}{1+z^2} \right\rfloor. \tag{24}$$

Whenever $z > \sqrt{b-1}$, the right-hand side is strictly less than 1, so the count must be 0. Thus for $b = 4$ (or a 4-token sub-block $b_{\text{sub}} = 4$) and $z \geq 2$, Cantelli's inequality guarantees a deterministic upper bound on $\max_{k\in B}\langle q, k\rangle$.

K.2 Diagonal Variance Surrogate and Spherical Norm Ceiling

To avoid $\mathcal{O}(d_h^2)$ storage for Σ_B, we cache only the coordinate-wise variance $\boldsymbol{\sigma}_B^2 = \mathrm{diag}(\Sigma_B) \in \mathbb{R}^{d_h}$, yielding the directional variance estimate $\sigma_S(q, B) = \sqrt{\sum_{i=1}^{d_h} q_i^2 \sigma_{B,i}^2}$. Intersecting this moment bound with the exact Cauchy–Schwarz spherical ceiling $\langle q, k\rangle \leq \|q\|_2 \max_{k\in B}\|k\|_2 = \|q\|_2 M_B$ yields the $\mathcal{O}(1)$ Dual Screening Score of Equation 7. Unlike coordinate-wise bounding boxes (Tang et al., 2024), which evaluate inner products against a synthetic corner vertex whose Euclidean norm inflates across $b = 64$ tokens when isolated coordinates spike, clipping by $\|q\|_2 M_B$ prevents coordinate outliers from triggering false admissions while projecting all G query heads in a GQA group onto a shared centroid $\boldsymbol{\mu}_B$.

L End-to-End Systems Accounting

To complement the standalone kernel benchmarks in Subsection 4.8 (Table 5), Table 20 reports full-model autoregressive decode latency ($q_{\text{len}} = 1, B = 64$, `float16`) across all 22 layers of our

1.45B model on an H100 SXM 80GB GPU. At each decode step, non-attention layers (QKV/O projections, SwiGLU MLPs, RMSNorms, and the vocabulary head) contribute a constant 11.76 ms/tok independent of L, while evaluating the threshold predictor τ_t^ℓ and updating the tail block metadata $(\boldsymbol{\mu}_B, \boldsymbol{\sigma}_B, M_B)$ adds 0.26 ms/tok ($< 0.6\%$ of step time). Because KV-cache attention accounts for 78%–93% of total latency at $L \geq 64$K, ETA delivers up to a $\mathbf{1.40}\times$ end-to-end speedup at 35.5% union density and $\mathbf{2.15}\times$ at 16.0% union density ($L = 256$K).

Table 20: Full 22-layer 1.45B model decode latency ($B = 64$, `float16`, H100 80GB). Total step time sums 22-layer attention, the constant non-attention floor (11.76 ms), and ETA predictor/metadata updates (0.26 ms).

Context L	Method ($B = 64$)	22-Layer Attn (ms)	Total Step (ms/tok)	E2E Speedup
64K	Dense FlashAttention-2	42.14	53.90	1.00×
	ETA 51.2% union (38% head)	40.33	52.35	1.03×
	ETA 35.5% union (25% head)	31.57	43.59	1.24×
	ETA 16.0% union (10% head)	**20.61**	**32.63**	**1.65×**
128K	Dense FlashAttention-2	82.92	94.68	1.00×
	ETA 35.5% union (25% head)	70.93	82.95	1.14×
	ETA 16.0% union (10% head)	**53.54**	**65.56**	**1.44×**
256K	Dense FlashAttention-2	167.37	179.13	1.00×
	ETA 35.5% union (25% head)	116.16	128.18	1.40×
	ETA 16.0% union (10% head)	**71.22**	**83.24**	**2.15×**

Prefill Indexing and Pretraining Throughput. During prompt prefill ($B = 1$), constructing the dual block metadata $(\boldsymbol{\mu}_B, \boldsymbol{\sigma}_B, M_B)$ requires a single $\mathcal{O}(Ld_h)$ pass over K, whereas attention scales as $\Theta(L^2 d_h)$; consequently, relative prefill overhead decays as $\mathcal{O}(1/L)$, dropping from +2.16% (+1.33 ms) at $L = 8$K to +0.51% (+2.43 ms) at 32K and $+\mathbf{0.14}\%$ (+8.31 ms out of 5,911 ms) at 128K. Similarly, during 22-layer pretraining at $L = 2048$ ($B = 4$, 8,192 tokens/step), linear projections and SwiGLU MLPs account for 83.2% of step time (7.21 ms/layer vs. 1.45 ms/layer for FlashAttention-2). Thus, although ETA's fused Triton training kernel takes 3.83 ms/layer, full-model step latency increases by only $+\mathbf{27.4}\%$ (242.8 ms vs. 190.6 ms), sustaining $\mathbf{293.5}$ TFLOPs/GPU (33,739 tok/s/GPU).