BLOCK SPARSE ATTENTION WITH LOG-LINEAR COMPLEXITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling language models to long contexts is limited by the quadratic cost of self-attention. Block sparse attention offers an efficient alternative, but selecting the retained blocks remains a bottleneck. Conventional block selection requires scoring all query–block pairs and therefore remains quadratic in sequence length. To address this issue, we propose PISA, a block-sparse attention mechanism that employs a pyramid Top-K selection strategy. The main idea is to gradually narrow down the candidates across different levels, making it more efficient to find the most relevant keys. Specifically, we construct a coarse-to-fine hierarchy of keys and perform selection from the coarsest level. At each level, LogSumExp scoring is applied to a bounded candidate set to select candidates for the next finer level, continuing until the finest level is reached. Through pooling, we construct $O(\log N)$ levels of keys, yielding an overall complexity of $O(N \log N)$, where N denotes the sequence length. We develop hardware-aware Triton kernels for both training and inference, fusing hierarchical routing and LogSumExp scoring without materializing the query–key score matrix. We further evaluate our method on language modeling tasks. Compared with the baseline, our method achieves comparable performance on benchmarks such as commonsense reasoning while delivering better results on retrieval tasks.

1 INTRODUCTION

Scaling language models to longer contexts has become increasingly important, but the computational complexity of full self-attention grows quadratically with sequence length, making it prohibitively expensive for long sequences (Vaswani et al., 2017). Sparse attention reduces this cost by restricting each query to a subset of keys (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Block-sparse attention groups keys and values into contiguous blocks of size C, allowing the sparse computation to use efficient tiled kernels (Yuan et al., 2025; Lu et al., 2025; Gao et al., 2025). Block-sparse attention generally consists of two stages: Top-K block selection and attention computation over the selected blocks. Once the Top-K blocks are selected, attention is computed using the original keys and values within these blocks. Since each query attends to at most KC keys, the second stage scales linearly with sequence length.

However, conventional Top-K block selection computes a relevance score between each query and every key block, and then selects the K highest-scoring blocks. For a sequence of length N with a block size of C, there are approximately N/C candidate key blocks. Since each query must scan all candidate key blocks, the selection stage still incurs a total complexity of $O(N^2/C)$ over the entire sequence. Therefore, although attention computation over the selected blocks scales linearly with sequence length, the Top-K block-selection stage remains a major computational bottleneck in long-context modeling.

To address this bottleneck, we propose **PISA** (**P**yram**i**d **S**parse **A**ttention), a block-sparse attention method based on hierarchical Top-K block selection. PISA first constructs a multilevel representation of key blocks through pooling. The finest level consists of fixed-size key blocks, while each coarser level is formed by aggregating adjacent representations from the level below. As the hierarchy becomes coarser, each block covers a larger portion of the sequence and the number of candidate blocks decreases, resulting in a coarse-to-fine pyramid of key representations.

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.

Selection starts at the coarsest level of the hierarchy. At each level, PISA computes LogSumExp (LSE) scores over a bounded set of candidate blocks and retains the Top-K candidates. Each selected block is then expanded into its finer-grained sub-blocks at the next level, while the unselected blocks are discarded. This coarse-to-fine process continues until the finest level is reached. In this way, each query only evaluates a small number of promising blocks at each level, avoiding an exhaustive comparison with all fine-grained blocks.

The pooled key hierarchy contains $O(\log N)$ levels. With fixed Top-K budget, branching factor, and leaf-block size, each query evaluates a bounded number of candidate blocks at every level, resulting in a per-query routing complexity of $O(\log N)$. Routing all queries in a sequence of length N therefore yields an overall block-selection complexity of $O(N \log N)$.

To execute this process efficiently on modern hardware, we develop hardware-aware Triton kernels for both training and inference (Tillet et al., 2019). Our implementation fuses multiple levels of hierarchical routing and LSE scoring into a single kernel, allowing intermediate candidates to be directly expanded and filtered at the next level without repeatedly reading and writing intermediate results across routing levels. The kernel maintains only a bounded candidate set at the current level and does not materialize or store a dense query–key score matrix, thereby reducing memory traffic and intermediate-tensor overhead for long sequences.

We evaluate our method on language modeling tasks. Compared with the conventional block-sparse attention baseline BSA, our method achieves comparable performance on commonsense reasoning benchmarks while delivering better results on retrieval tasks. These results indicate that coarse-to-fine candidate search reduces selection complexity while preserving blocks relevant to downstream modeling. Top-K selection benchmarks with sequence lengths of up to 256K tokens further show that hierarchical routing is more efficient than conventional single-level selection for long sequences.

2 Related Work

Table 1: Training settings and computational complexity of selected sparse attention methods. N denotes the sequence length. Complexities include selection and attention computation. ✓ denotes trainable, and × denotes a training-free method. For LLSA (Zhou et al., 2026), which is designed for diffusion models, the listed cost is per attention pass rather than autoregressive prefill.

Method	Trainable	Prefill complexity	Decode complexity
HiP (Lee et al., 2025)	×	$O(N \log N)$	$O(\log N)$
HISA (Xu et al., 2026)	×	$O(N^2)$	$O(N)$
MoBA (Lu et al., 2025)	✓	$O(N^2)$	$O(N)$
NSA (Yuan et al., 2025)	✓	$O(N^2)$	$O(N)$
HiLS (Hu et al., 2026)	✓	$O(N^2)$	$O(N)$
LLSA (Zhou et al., 2026)	✓	$O(N \log N)$	—
BSA (Yuan et al., 2025)	✓	$O(N^2)$	$O(N)$
PISA (ours)	✓	$O(N \log N)$	$O(\log N)$

Sparse attention reduces attention computation by limiting the keys and values that each query attends to. Existing methods either apply sparsity to pretrained models without updating their weights or incorporate sparse attention during model training. We review these training-free and trainable methods below. Table 1 summarizes the training settings and computational complexity of selected methods closely related to ours.

Training-Free Sparse Attention. Training-free sparse attention reduces inference cost without retraining the model. During prefill, methods use head-specific sparse patterns (Jiang et al., 2024) or estimate block importance through sampling and online filtering (Xu et al., 2025; Zhang et al., 2025). During decoding, methods reduce KV-cache accesses by selecting relevant tokens or pages before loading their full keys and values. Selection scores are computed from a subset of key dimensions (Ribar et al., 2024) or page-level minimum and maximum key values (Tang et al., 2024). To reduce the number of token scores, two-stage methods first select blocks or clusters using their summaries, then score tokens only within the selected groups (Xu et al., 2026; Ni et al., 2026). HiP (Lee et al., 2025) repeats selection across multiple levels: it divides retained regions into smaller ones and keeps a fixed number at each step, giving log-linear complexity for a fixed selection budget.

Trainable Sparse Attention. Trainable methods train or adapt models with sparse attention. Early methods combine local attention with selected long-range connections (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Query-dependent methods such as MoBA (Lu et al., 2025) and InfLLM-V2 (Zhao et al., 2025) use pooled block summaries, while NSA (Yuan et al., 2025) and HiLS (Hu et al., 2026) use learned block summaries to select relevant blocks. Other methods learn gates or use differentiable rules for block selection (Gao et al., 2025; Huang et al., 2026). IndexCache (Bai et al., 2026) and LongCat (Zan et al., 2026) use cross-layer distillation to enable index reuse across layers. SSA and H-SSA (Mao et al., 2026) use continued pretraining (CPT) to learn single-level and hierarchical gist summaries, respectively, for chunk selection. LLSA (Zhou et al., 2026) performs hierarchical Top-K selection for diffusion transformers and includes both original and compressed keys and values in the final attention computation.

3 METHOD

3.1 BLOCK SPARSE ATTENTION

Block sparse attention (BSA) (Yuan et al., 2025; Lu et al., 2025) partitions the key sequence into contiguous blocks and selects a subset of these blocks for each query. The values at the same positions are selected together with each key block.

Let $q_t, k_j, v_j \in \mathbb{R}^d$ denote the query at position t and the key and value at position j. For block size C, the i-th key block is $K_i = \left[k_{(i-1)C+1}, \ldots, k_{\min(iC,N)}\right]$, where $i \in \{1, \ldots, M\}$ and $M = \lceil N/C \rceil$. BSA first applies a block mapping f, such as mean pooling (Lu et al., 2025) or a learned linear projection (Yuan et al., 2025), to obtain a summary vector $\bar{k}_i = f(K_i) \in \mathbb{R}^d$.

The summary vector is used for selection, for query q_t, BSA computes one score for every block:

$$\mathbf{s}_t = [s_{t,1}, \ldots, s_{t,M}], \qquad s_{t,i} = \frac{q_t^\top \bar{k}_i}{\sqrt{d}}. \tag{1}$$

BSA retains the indices of the K largest scores, $\mathcal{I}_t = \mathrm{TopK}_K(\mathbf{s}_t)$.

Let $\mathcal{T}_t = \bigcup_{i \in \mathcal{I}_t} \{(i-1)C + 1, \ldots, \min(iC, N)\}$ contain the token positions covered by the selected blocks. The attention output is then computed from the original keys and values at these positions:

$$o_t = \sum_{j \in \mathcal{T}_t} \exp\left(q_t^\top k_j / \sqrt{d}\right) v_j \Big/ \left(\sum_{j \in \mathcal{T}_t} \exp\left(q_t^\top k_j / \sqrt{d}\right)\right). \tag{2}$$

The final attention reads the original keys and values from the selected blocks. It therefore accesses at most KC keys per query, but scoring all $M = \lceil N/C \rceil$ block summaries for every query still costs $O(N^2 d/C)$.

3.2 PYRAMID TOP-K BLOCK SELECTION

PISA performs hierarchical Top-K block selection on a key-block hierarchy constructed from fine to coarse. Selection proceeds in the opposite direction and expands only the blocks retained at the previous level. We first define the hierarchy and candidate sets, and then introduce the LSE score used for selection.

Fine-to-coarse block aggregation. We first construct the key-block hierarchy from fine to coarse. At level $\ell \geq 0$, let M_ℓ denote the number of blocks and $K_i^{(\ell)}$ the i-th block, with $i \in \{1, \ldots, M_\ell\}$. Level 0 contains the original keys, each represented as a singleton block: $M_0 = N$ and $K_i^{(0)} = [k_i]$.

Let g_ℓ be the maximum number of adjacent level-ℓ blocks combined into one level-$(\ell+1)$ block. We use $g_0 = C$ and $g_\ell = g = 2$ for $\ell \geq 1$. Let $\mathrm{Ch}_\ell(i)$ denote the indices of the level-ℓ children of $K_i^{(\ell+1)}$. For $\ell \geq 0$, the hierarchy is constructed as

$$M_{\ell+1} = \left\lceil \frac{M_\ell}{g_\ell} \right\rceil, \quad \mathrm{Ch}_\ell(i) = \{(i-1)g_\ell + 1, \ldots, \min(ig_\ell, M_\ell)\}, \quad K_i^{(\ell+1)} = \mathrm{Concat}_{r \in \mathrm{Ch}_\ell(i)} K_r^{(\ell)}. \tag{3}$$

Here, r indexes the child blocks, and Concat preserves their sequence order. The first aggregation step forms the leaf blocks, with $M_1 = M = \lceil N/C \rceil$ and $K_i^{(1)} = K_i$.

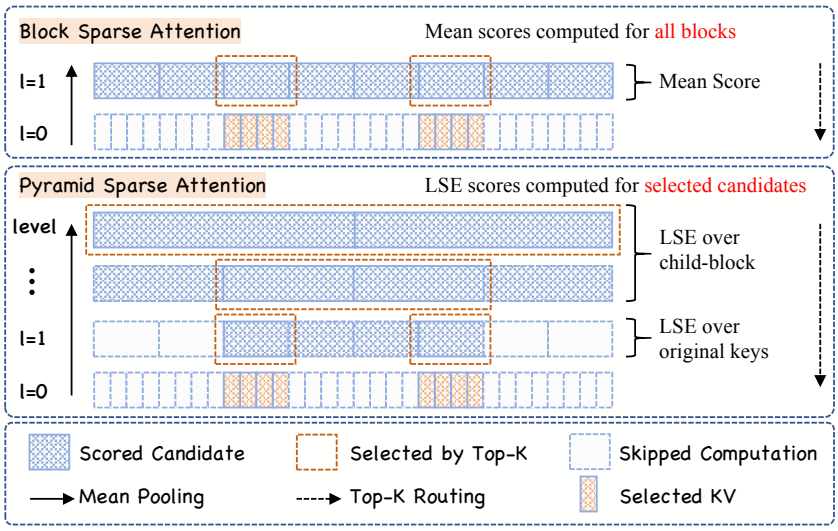


Figure 1: Comparison of two different Top-K selection methods for sparse attention. Top: BSA computes a score for each key block before performing Top-K selection. Bottom: PISA constructs a fine-to-coarse hierarchy of key blocks through mean pooling and expands only the selected candidates from coarse to fine during selection. Intermediate blocks are scored using LSE over their child summaries, while leaf blocks are scored using exact LSE over their original keys.

Each block $K_i^{(\ell)}$ has a summary vector $\bar{k}_i^{(\ell)} \in \mathbb{R}^d$. Starting from $\bar{k}_i^{(0)} = k_i$, we compute the summaries recursively by mean pooling:

$$\bar{k}_i^{(\ell+1)} = \frac{1}{|\operatorname{Ch}_\ell(i)|} \sum_{r \in \operatorname{Ch}_\ell(i)} \bar{k}_r^{(\ell)}, \qquad \ell \geq 0. \tag{4}$$

The denominator counts the valid child blocks. When child blocks are equally sized at every aggregation step, the summary equals the mean of all original keys in the block. Queries remain token-wise throughout selection.

Coarse-to-fine block selection. Selection proceeds from the coarsest level L to the leaf-block level 1. For each query q_t, we initialize the candidate index set at the coarsest level as $\mathcal{A}_t^{(L)} = \{1\}$.

At level ℓ, $\mathcal{A}_t^{(\ell)}$ denotes the candidate index set, $s_{t,i}^{(\ell)}$ denotes the score of candidate block i, and $\mathbf{s}_t^{(\ell)}$ collects the scores of all candidate blocks. The operator Select_K returns the indices of the K highest-scoring blocks, denoted by $\mathcal{I}_t^{(\ell)}$. The candidate set at level $\ell - 1$ is then constructed from the children of the retained blocks in $\mathcal{I}_t^{(\ell)}$. The complete selection and expansion process is defined as

$$\mathbf{s}_t^{(\ell)} = \left[s_{t,i}^{(\ell)}\right]_{i \in \mathcal{A}_t^{(\ell)}}, \qquad \mathcal{I}_t^{(\ell)} = \operatorname{Select}_K\left(\mathcal{A}_t^{(\ell)}, \mathbf{s}_t^{(\ell)}\right), \qquad \mathcal{A}_t^{(\ell-1)} = \bigcup_{i \in \mathcal{I}_t^{(\ell)}} \operatorname{Ch}_{\ell-1}(i), \quad (\ell > 1). \tag{5}$$

If a level contains at most K candidate blocks, all candidates are retained without scoring. Otherwise, Select_K retains at most K blocks. For $\ell > 1$, each retained block has at most $g_{\ell-1} = g$ children, where we set $g = 2$ in practice. Therefore,

$$\left|\mathcal{I}_t^{(\ell)}\right| \leq K, \qquad \left|\mathcal{A}_t^{(\ell-1)}\right| \leq g\left|\mathcal{I}_t^{(\ell)}\right| \leq gK. \tag{6}$$

The final set $\mathcal{I}_t = \mathcal{I}_t^{(1)}$ identifies the original key and value blocks used in the sparse attention computation. Appendix B gives the complete forced-block policy.

LogSumExp block scoring. For query q_t, PISA assigns candidate $K_i^{(\ell)}$ the LSE score

$$s_{t,i}^{(\ell)} = \log \sum_{r \in \operatorname{Ch}_{\ell-1}(i)} \exp\left(\frac{q_t^\top \bar{k}_r^{(\ell-1)}}{\sqrt{d}}\right), \qquad \ell \geq 1. \tag{7}$$

Each score reduces at most $g_{\ell-1}$ logits: C at the leaf level and g at intermediate levels. Since $\bar{k}_r^{(0)} = k_r$, a completed historical leaf receives its exact raw-key LSE score, whereas an intermediate block is scored from its child summaries. Appendix A gives further scoring details.

3.3 Kernel Implementation

Following NSA (Yuan et al., 2025), query heads within the same GQA group share the same selected key blocks. We design different kernels for training/prefill and decoding. For training and prefill, we adopt a two-stage kernel design to reduce IO overhead. The first stage performs hierarchical selection over the intermediate levels, while the second stage scores the original key blocks. For decoding, we use a single-stage kernel to avoid the overhead of multiple kernel launches.

Algorithm 1 Two-stage block selection for PISA

Input: query vectors $\{q_{t,h'}\}$, key summaries $\{\bar{k}_{i,h}^{(\ell)}\}_{\ell=1}^{L}$, original key blocks $\{K_{i,h}^{(1)}\}$, initial candidate sets $\{\mathcal{A}_{t,h}^{(L)}\}$, block budget K, and maximum query positions per program Q_{tile}

Output: selected leaf-block indices $\{\mathcal{I}_{t,h}^{(1)}\}$

Stage 1: Intermediate-level selection

for all query positions t and KV heads h in parallel **do**
  load $\{q_{t,h'} : h' \in \mathcal{H}(h)\}$ once; initialize $\mathcal{A} \leftarrow \mathcal{A}_{t,h}^{(L)}$
  for $\ell = L, L-1, \ldots, 2$ **do**
    for all $i \in \mathcal{A}$ requiring scores **do**
      $s_{h',t,i}^{(\ell)} \leftarrow \log \sum_{r \in \mathrm{Ch}_{\ell-1}(i)} \exp\left(q_{t,h'}^{\top} \bar{k}_{r,h}^{(\ell-1)} / \sqrt{d}\right)$
      $u_{h,t,i}^{(\ell)} \leftarrow \sum_{h' \in \mathcal{H}(h)} s_{h',t,i}^{(\ell)}$
    end for
    select $\mathcal{I} \leftarrow \mathrm{Select}_K\left(\mathcal{A}, \{u_{h,t,i}^{(\ell)}\}_{i \in \mathcal{A}}\right)$
    expand $\mathcal{A} \leftarrow \bigcup_{i \in \mathcal{I}} \mathrm{Ch}_{\ell-1}(i)$, where $\mathrm{Ch}_{\ell-1}(i)$ indexes block i's children
  end for
  store $\mathcal{A}_{t,h}^{(1)} \leftarrow \mathcal{A}$
end for

Stage 2: Original key-block scoring

For each KV head h and leaf block i, collect the query positions t that require a score for $i \in \mathcal{A}_{t,h}^{(1)}$, divide each query cluster into disjoint groups $I_{h,i,j}$ with $|I_{h,i,j}| \le Q_{\text{tile}}$.

for all groups $I_{h,i,j}$ in parallel **do**
  load $K_{i,h}^{(1)}$ once and $\{q_{t,h'} : t \in I_{h,i,j},\ h' \in \mathcal{H}(h)\}$
  $s_{h',t,i}^{(1)} \leftarrow \log \sum_{r \in \mathrm{Ch}_0(i)} \exp\left(q_{t,h'}^{\top} k_{r,h} / \sqrt{d}\right)$
  store $u_{h,t,i}^{(1)} \leftarrow \sum_{h' \in \mathcal{H}(h)} s_{h',t,i}^{(1)}$ for each query position $t \in I_{h,i,j}$
end for

Final Top-K selection

for all query positions t and KV heads h in parallel **do**
  $\mathcal{I}_{t,h}^{(1)} \leftarrow \mathrm{Select}_K\left(\mathcal{A}_{t,h}^{(1)}, \{u_{h,t,i}^{(1)}\}_{i \in \mathcal{A}_{t,h}^{(1)}}\right)$
end for

return $\{\mathcal{I}_{t,h}^{(1)}\}$

Training/Prefill Stage 1: Intermediate-level selection. In the first stage, we parallelize over the query position t and KV head h. Let $\mathcal{H}(h)$ denote the set of query heads that share KV head h. Each kernel program loads the corresponding query vectors once and sequentially processes the intermediate levels from $\ell = L$ to $\ell = 2$.

At level ℓ, $\mathcal{A}_{t,h}^{(\ell)}$ denotes the candidate block indices. Using the child summaries of each candidate block, the program computes an LSE score for every candidate and query head. It then sums the scores over all query heads sharing KV head h:

$$u_{h,t,i}^{(\ell)} = \sum_{h' \in \mathcal{H}(h)} s_{h',t,i}^{(\ell)}. \tag{8}$$

Here, $s^{(\ell)}_{h',t,i}$ is the LSE score of candidate block $i \in \mathcal{A}^{(\ell)}_{t,h}$ for query head h'. The program applies Select_K to obtain the retained indices $\mathcal{I}^{(\ell)}_{t,h}$ and expands the selected blocks into their children to form $\mathcal{A}^{(\ell-1)}_{t,h}$. The candidate indices remain local to the program throughout this loop. If at most K candidates are valid, all are retained without scoring.

Training/Prefill Stage 2: Original key-block selection. After the first stage, each query has at most gK candidate key blocks. For each KV head, we group queries that need to score the same candidate key block into a query cluster. Each cluster is divided into tiles of at most Q_{tile} queries, and tiles across all clusters are processed in parallel. Each kernel program loads the corresponding key block once and reuses it to jointly score the queries in its tile. In our implementation, we set $Q_{\mathrm{tile}} = 4$.

Let $G_Q = |\mathcal{H}(h)|$ denote the number of query heads sharing the same KV head. Each program loads a key tile of shape $d \times C$ covering one leaf block and a query tile of shape $(Q_{\mathrm{tile}}G_Q) \times d$. The program computes

$$\underbrace{(Q_{\mathrm{tile}}G_Q) \times d}_{\text{query tile}} \times \underbrace{d \times C}_{\text{key tile}} \longrightarrow (Q_{\mathrm{tile}}G_Q) \times C. \tag{9}$$

The program applies LSE over keys for each query head, then sums across the G_Q query heads:

$$(Q_{\mathrm{tile}}G_Q) \times C \xrightarrow{\text{LSE}} Q_{\mathrm{tile}}G_Q \xrightarrow{\text{sum over heads}} Q_{\mathrm{tile}}. \tag{10}$$

After the scores are stored, a final kernel independently applies Select_K for each query and KV head.

Decoding. During decoding, we cache the mean pyramid and update only the current leaf mean and its ancestor path for each new token. Block selection uses a single kernel that follows Stage 1 but continues to $\ell = 1$, directly scoring the original key blocks. Combining all selection levels into a single kernel avoids the overhead of launching multiple kernels for each decoding step.

3.4 Complexity

Computational complexity. Because the pyramid key hierarchy is constructed through downsampling, it contains $L = O(\log N)$ levels. At each level, PISA scores at most gK candidate blocks for each query. Each intermediate candidate uses at most g child summaries, and each leaf candidate uses at most C original keys. With fixed g, K, block size, head counts, and head dimension, the selection complexity for a query is $O(\log N)$. Processing all N queries during training or prefill costs $O(N \log N)$. Decoding costs $O(\log N)$ per step on average, including the occasional cost of expanding the cache.

Why do training/prefill and decoding use different kernel designs? A natural question is why Stage 1 does not continue directly to $\ell = 1$ during training and prefill. We compare the two designs' leaf-level Q/K IO costs for one KV head, excluding the initial query load shared by both.

In the two-stage design, each query is associated with at most gK candidate key blocks. Loading the query vectors for all query–block pairs therefore incurs an IO cost of $O(NgKG_Qd)$. Since query tiles are processed in parallel, each key block is loaded

$$O\left(\frac{NgK}{(N/C)Q_{\mathrm{tile}}}\right) = O\left(\frac{CgK}{Q_{\mathrm{tile}}}\right) \tag{11}$$

times on average. Since there are N/C key blocks and loading one key block costs $O(Cd)$, the total key IO cost is

$$O\left(\frac{CgK}{Q_{\mathrm{tile}}} \cdot \frac{N}{C} \cdot Cd\right) = O\left(\frac{NCgKd}{Q_{\mathrm{tile}}}\right). \tag{12}$$

Under this approximation, the two-stage leaf-level Q/K IO cost is

$$O\left(NgKd\left(G_Q + \frac{C}{Q_{\mathrm{tile}}}\right)\right). \tag{13}$$

In comparison, a single-stage kernel would load at most gK original key blocks per query at $\ell = 1$. With C keys per block, its additional leaf-level IO cost is

$$O(NgKCd). \tag{14}$$

When $G_Q + \frac{C}{Q_{\text{tile}}} < C$, the two-stage design reduces the leaf-level Q/K IO cost by reusing each loaded key block across multiple queries. In our implementation, we set $G_Q = 16$, $Q_{\text{tile}} = 4$, and $C = 64$, which gives

$$G_Q + \frac{C}{Q_{\text{tile}}} = 16 + \frac{64}{4} = 32 < 64 = C. \tag{15}$$

Thus, under these assumptions, the two-stage design has a lower leaf-level Q/K IO cost for our configuration.

During decoding, each cluster contains only one query, so there is no cross-query key reuse. A two-stage kernel would load gKC keys and load the query vectors gK times at $\ell = 1$, giving an IO cost of

$$O(gK(C + G_Q)d)\,. \tag{16}$$

In contrast, the single-stage kernel reuses the query vectors already loaded during intermediate-level selection. Its additional leaf-level IO cost is

$$O(gKCd). \tag{17}$$

The fused selector avoids reloading queries at the leaf level and requires no separate grouping or final-selection launches. Accordingly, we use the two-stage kernel for training and prefill and the single-stage kernel for decoding.

4 Experiments

4.1 Experimental setup

We compare PISA with Full Attention, BSA, NSA (Yuan et al., 2025), and HiLS (Hu et al., 2026) at the 418M, 1.47B, and 2.67B backbone scales, using the same decoder-only backbone configuration at each scale and matched training settings. BSA uses the selected-attention branch of NSA with mean-pooled key summaries. NSA retains its learned compression and selected-attention branches, while HiLS retains its learnable landmark-based routing. We pretrain the models for 100B tokens at a sequence length of 4K, followed by 10B additional tokens of continued pretraining (CPT) at 16K. Sparse methods use block size $C = 64$ throughout, with block budgets $K = 8$ and $K = 32$ for the two stages, respectively. We report training loss, language-model perplexity, multiple-choice accuracy, and containment accuracy, and use RULER (Hsieh et al., 2024) to measure long-context retrieval. Complete training and evaluation protocols are given in Appendix C.

4.2 Language Modeling and Downstream Evaluation

Table 2 compares language-modeling and downstream performance at three model scales. PISA performs comparably to BSA, NSA, and HiLS on language modeling and commonsense reasoning. Across the six containment tasks, PISA achieves the highest average accuracy among sparse methods at all three scales, although Full Attention retains a higher average.

4.3 Long-context evaluation

We evaluate long-context retrieval on four needle-in-a-haystack task families from RULER using the models after continued pretraining at 16K. Table 3 reports accuracy for the 2.67B models at context lengths from 1K to 16K. PISA achieves strong long-context retrieval performance. Training loss and downstream results are provided in Appendix D.1.

4.4 Ablation Studies

Scoring function. We compare PISA with PISA-1 and PISA-2, keeping the backbone and hierarchy unchanged. For logits $z_1, \ldots, z_m$ with mean $\bar{z}$ and variance $\mathrm{Var}(z)$, Taylor expansion gives

$$s_{\text{PISA}} = \log \sum_{j=1}^{m} e^{z_j} = \log m + \bar{z} + \frac{1}{2}\,\mathrm{Var}(z) + \text{higher-order terms}. \tag{18}$$

Omitting $\log m$, PISA-1 and PISA-2 truncate this expansion at first and second order, respectively:

$$s_{\text{PISA-1}} = \bar{z}, \qquad s_{\text{PISA-2}} = \bar{z} + \frac{1}{2}\,\mathrm{Var}(z).$$

Table 2: Language modeling and downstream evaluation after 100B tokens of pretraining at 4K. Baselines include NSA (Yuan et al., 2025) and HiLS (Hu et al., 2026). Loss is the final training loss, and accuracies are reported as percentages. Gray columns show group averages; boldface marks the best sparse loss and group averages at each scale.

Method	Params	Loss ↓	Perplexity			Multiple-choice									Containment						
			Wiki. ppl↓	LMB. ppl↓	Avg ppl↓	BoolQ acc↑	PIQA acc↑	Hella. acc-n↑	Wino. acc↑	ARC-e acc↑	ARC-c acc-n↑	OBQA acc-n↑	SIQA acc↑	Avg acc↑	SWDE acc↑	SQuAD acc↑	FDA acc↑	TQA acc↑	NQ acc↑	DROP acc↑	Avg acc↑
From-scratch training, 418M																					
Full attention	418M	2.5557	20.40	25.60	23.00	53.64	68.93	47.57	55.01	64.35	32.00	32.40	39.00	49.11	66.16	40.95	60.25	56.64	24.33	22.57	45.15
NSA	420M	2.5727	21.46	26.13	23.80	60.43	68.82	47.04	53.51	63.97	31.48	33.80	40.38	49.93	48.51	38.40	41.11	55.33	22.84	18.59	37.46
HiLS	423M	**2.5596**	20.83	26.67	23.75	58.99	70.67	47.28	54.14	65.11	31.48	35.00	40.69	50.42	52.03	39.01	53.27	55.98	23.09	21.37	40.79
BSA	418M	2.5684	21.47	25.02	23.25	57.92	69.59	47.65	53.35	65.32	29.78	34.60	38.79	49.63	52.30	40.48	37.57	53.97	23.34	21.42	38.18
PISA-1	418M	2.5683	21.53	24.94	23.23	55.38	68.72	47.85	54.46	64.23	31.91	33.00	39.30	49.36	52.30	39.11	45.64	55.39	21.79	21.18	39.24
PISA-2	418M	2.5663	21.31	24.78	23.04	53.12	69.26	47.32	55.01	65.66	30.29	37.60	39.66	49.74	53.11	39.21	54.45	54.92	23.95	20.17	40.97
PISA	418M	2.5630	21.17	24.91	**23.04**	58.41	69.80	47.52	55.25	65.19	32.51	36.00	39.51	**50.52**	57.07	38.81	54.81	56.16	23.79	19.98	**41.77**
From-scratch training, 1.47B																					
Full attention	1.47B	2.3155	15.33	13.31	14.32	56.33	73.39	58.37	58.56	71.89	38.48	40.80	42.48	55.04	75.97	43.80	74.68	63.86	30.54	23.91	52.13
NSA	1.47B	2.3142	15.70	11.93	**13.82**	62.60	74.05	58.98	60.14	73.82	38.65	42.40	41.91	**56.57**	68.68	42.29	59.71	62.74	28.25	22.14	47.30
HiLS	1.48B	**2.3084**	15.42	12.72	14.07	60.31	73.99	58.79	59.27	73.32	39.76	39.60	41.61	55.83	68.50	44.77	65.06	63.15	27.78	21.99	48.54
BSA	1.47B	2.3179	15.63	12.95	14.29	58.26	74.54	59.30	59.59	72.35	37.97	41.60	41.04	55.58	68.32	43.10	64.88	63.09	27.65	24.20	48.54
PISA-1	1.47B	2.3215	15.77	13.38	14.58	60.95	73.72	58.46	58.72	72.60	39.08	40.20	41.10	55.60	67.96	43.90	56.53	63.51	29.27	23.24	47.40
PISA-2	1.47B	2.3168	15.59	12.90	14.24	62.97	73.39	58.84	57.06	71.25	38.23	39.20	40.48	55.18	69.67	45.04	67.42	63.03	29.24	24.34	49.79
PISA	1.47B	2.3162	15.58	13.27	14.43	62.94	73.83	58.91	58.33	72.14	40.19	40.00	41.35	55.96	71.83	43.93	69.24	63.27	29.17	21.56	**49.83**
From-scratch training, 2.67B																					
Full attention	2.67B	2.2209	13.74	10.81	12.28	64.50	74.86	62.37	62.12	74.92	43.77	42.00	41.30	58.23	77.23	46.01	75.23	64.75	30.92	23.91	53.01
NSA	2.68B	2.2098	13.76	9.89	**11.83**	62.91	74.86	63.38	61.88	75.21	43.94	42.80	42.73	58.46	74.35	45.54	61.07	66.05	30.73	23.48	50.20
HiLS	2.70B	**2.2047**	13.67	10.19	11.93	62.20	74.92	63.41	60.06	76.09	43.60	42.20	42.53	58.13	70.75	45.17	73.41	66.29	30.85	23.24	51.62
BSA	2.67B	2.2184	13.87	10.34	12.11	64.92	74.76	63.42	61.48	76.14	41.81	42.60	43.71	**58.61**	72.46	45.64	67.24	66.53	30.16	24.72	51.13
PISA-1	2.67B	2.2179	13.96	10.36	12.16	62.75	74.37	63.24	62.12	76.09	41.98	42.00	42.58	58.14	70.66	44.81	68.06	64.87	30.73	26.11	50.87
PISA-2	2.67B	2.2152	13.88	10.30	12.09	62.08	74.81	63.29	60.62	76.18	42.75	42.00	42.94	58.08	72.01	46.62	72.87	65.70	30.98	24.10	52.05
PISA	2.67B	2.2151	13.83	10.36	12.10	61.25	75.19	63.23	60.85	75.51	43.94	42.20	42.63	58.10	75.07	45.95	69.78	65.58	31.42	25.06	**52.14**

Table 3: RULER needle-in-a-haystack results after 10B tokens of continued pretraining at 16K for the 2.67B models. Sparse methods use $C = 64$ and $K = 32$. Scores are accuracies in percent. The gray Avg column averages scores over the four task families at 1K, 2K, 4K, 8K, and 16K, before rounding. Boldface marks the best sparse result in each column.

Size	Method	niah_single					niah_multikey					niah_multiquery					niah_multivalue					Avg ↑
		1K	2K	4K	8K	16K	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K	
2.67B	Full Attention	99.67	99.60	99.47	99.13	98.80	43.33	38.40	34.67	28.00	23.13	85.60	72.45	50.05	51.55	34.85	90.15	84.55	77.05	57.10	57.20	66.24
	NSA	**99.93**	99.53	**99.33**	**97.47**	**91.67**	57.47	52.00	**40.67**	**18.00**	**15.20**	72.60	79.85	32.65	50.10	24.20	83.80	69.55	59.85	48.65	**28.90**	61.07
	BSA	99.20	98.40	93.60	85.33	59.67	37.33	32.20	20.87	13.40	7.13	80.75	69.85	66.85	37.90	15.10	83.80	75.15	63.45	38.15	21.70	54.99
	PISA-1	99.87	**99.73**	97.33	81.87	60.27	66.80	**53.40**	39.40	12.60	9.93	**93.40**	**89.90**	73.75	38.50	21.70	83.30	82.95	**70.50**	38.15	20.45	61.69
	PISA-2	99.47	99.33	99.00	90.00	72.40	51.13	40.00	27.20	13.13	9.33	91.45	86.95	77.85	**59.55**	**25.20**	89.60	81.25	69.60	**52.55**	23.30	**62.92**
	PISA	99.07	94.60	96.33	91.47	71.27	**68.20**	52.33	32.47	13.53	10.00	88.65	85.35	**79.00**	48.50	20.85	**93.40**	**87.10**	58.60	43.95	21.30	62.80

The pretraining results in Table 2 show that PISA has lower training loss and higher average containment accuracy than PISA-1, PISA-2, and BSA at all three scales. Appendix A provides the scoring derivations.

Block-selection quality. We evaluate all block selectors on identical query and key tensors from a Full-Attention checkpoint. For each query position i and KV head h, each method selects K blocks using its scoring rule, and we construct a reference set of K blocks from the full-attention weights. We denote these sets by $I^{i,h}_{\text{sparse}}$ and $I^{i,h}_{\text{full}}$, respectively. Let $P_{i,h}(B)$ denote the full-attention mass assigned to block B, averaged over the query heads sharing KV head h. We report two metrics:

$$\text{Recall@K}_{i,h} = \frac{|I^{i,h}_{\text{sparse}} \cap I^{i,h}_{\text{full}}|}{K}, \qquad \text{MassRatio}_{i,h} = \frac{\sum_{B \in I^{i,h}_{\text{sparse}}} P_{i,h}(B)}{\sum_{B \in I^{i,h}_{\text{full}}} P_{i,h}(B)}. \tag{19}$$

Recall@K measures the overlap between the selected and reference blocks; attention mass ratio measures the attention mass retained by the selected blocks relative to the reference set, not total full-attention mass. In this experiment, PISA achieves the highest average Recall@8 and attention mass

ratio (Figure 2). The complete evaluation setup and aggregate results are provided in Appendix D.2. This indicates that LSE-based scoring enables more accurate Top-K selection.

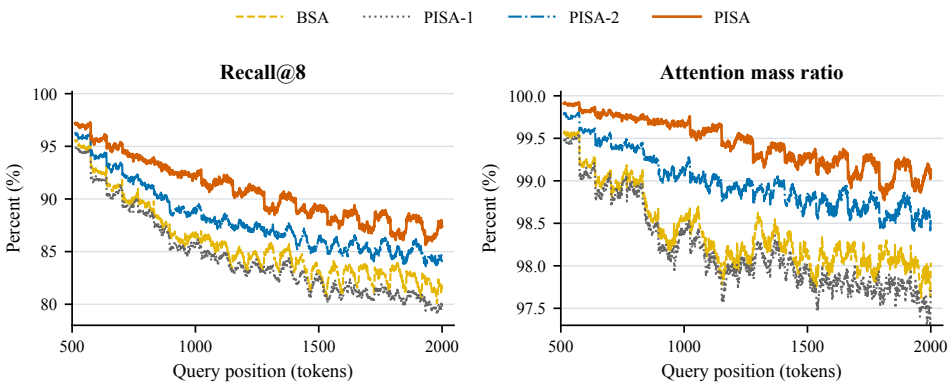


Figure 2: Block-selection quality on 100 FDA prompts using identical Full-Attention queries and keys. Curves show mean Recall@8 and attention mass ratio by query position. All selectors and the reference share $K = 8$ and the forced-block policy; Recall@8 includes forced blocks.

4.5 Block-selection efficiency

Figure 3 compares prefill block-selection latency for BSA and the Q4 implementation of PISA ($Q_{\text{tile}} = 4$). BSA is faster from 4K to 16K, whereas PISA achieves $2.86\times$, $5.31\times$, and $9.95\times$ speedups over BSA at 64K, 128K, and 256K, respectively. At the same lengths, Q4 achieves $1.35\times$, $1.30\times$, and $1.33\times$ speedups over the per-query implementation, respectively. Detailed timings are provided in Appendix B (Table 4).

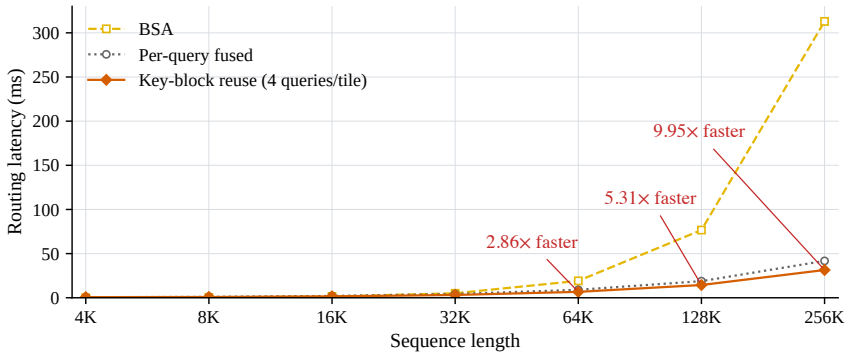


Figure 3: Prefill block-selection latency from 4K to 256K with $C = 64$ and $K = 8$. Timings include mean-summary construction and all selection stages, but exclude attention over the selected blocks. The key-block reuse implementation of PISA uses $Q_{\text{tile}} = 4$; annotations show speedups over BSA. Complete settings are provided in Appendix B.

5 Conclusion

We introduced PISA, a block-sparse attention method that combines pyramid Top-K selection with LogSumExp scoring. By scoring a bounded candidate set at each level, PISA reduces prefill complexity to $O(N \log N)$ for fixed model dimensions and selection parameters. Our Triton kernels fuse intermediate selection levels and reuse leaf-key blocks across queries. Experiments across three model scales show comparable language-modeling and commonsense-reasoning performance to sparse baselines, with higher average containment accuracy after pretraining. Block-selection benchmarks further demonstrate lower latency than BSA at long sequence lengths.

AI USE STATEMENT

Generative AI tools assisted with searching and organizing relevant literature, translating text, and reviewing manuscript drafts to provide writing suggestions. LLM-based agents automated the submission of experiment jobs in batches. The authors take full responsibility for all claims and artifacts.

ETHICS STATEMENT

This work studies efficient sparse attention through language-model training and benchmark evaluation. The proposed method improves attention computation and does not target any specific downstream application.

REPRODUCIBILITY STATEMENT

The appendices provide the scoring analysis, kernel implementation details, training configurations, evaluation protocols, and additional experimental results (Appendices A–D).

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. Simple linear attention language models balance the recall-throughput tradeoff. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1763–1840. PMLR, 21–27 Jul 2024a. URL https://proceedings.mlr.press/v235/arora24a.html.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models, 2024b. URL https://arxiv.org/abs/2407.05483.

Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. Indexcache: Accelerating sparse attention via cross-layer index reuse, 2026. URL https://arxiv.org/abs/2603.12201.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2026. URL https://arxiv.org/abs/2405.14782.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. Seerattention: Learning intrinsic sparse attention in your llms, 2025. URL https://arxiv.org/abs/2410.13276.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Xiang Hu, Xinyu Wei, Hao Gu, Minshen Zhang, Tian Liang, Huayang Li, Lei Zhu, Yan Wang, Sirui Han, Yushi Bai, Kewei Tu, Haitao Mi, and Leo Liang. Hierarchical sparse attention done right: Toward infinite context modeling, 2026. URL https://arxiv.org/abs/2607.02980.

Yuxiang Huang, Nuno M. T. Gonçalves, Federico Alvetreti, Lei Li, Xu Han, Edoardo M. Ponti, André F. T. Martins, and Marcos V. Treviso. Dashattention: Differentiable and adaptive sparse hierarchical attention, 2026. URL https://arxiv.org/abs/2605.18753.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 52481–52515. Curran Associates, Inc., 2024. doi: 10.52202/079017-1663. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5dfbe6f5671e82c76841ba687a8a9ecb-Paper-Conference.pdf.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Heejun Lee, Geon Park, Youngwan Lee, Jaduk Suh, Jina Kim, Wonyong Jeong, Bumsik Kim, Hyemin Lee, Myeongjae Jeon, and Sung Ju Hwang. A training-free sub-quadratic cost transformer model serving framework with hierarchically pruned attention. In Y. Yue, A. Garg, N. Peng, F. Sha, and

R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 1280–1325, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/043f0503c4f652c737add3690aa5d12c-Paper-Conference.pdf.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17790–17815. Curran Associates, Inc., 2025. doi: 10.52202/085713-0602. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19eae75beed66321d62272e794a9c2ac-Paper-Conference.pdf.

Yuzhen Mao, Michael Y. Li, and Emily B. Fox. Simplified sparse attention via gist tokens, 2026. URL https://arxiv.org/abs/2604.20920.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Wentao Ni, Kangqi Zhang, Zhongming Yu, Oren Nelson, Mingu Lee, Hong Cai, Fatih Porikli, Jongryool Kim, Zhijian Liu, and Jishen Zhao. Double-p: Hierarchical top-p sparse attention for long-context llms, 2026. URL https://arxiv.org/abs/2602.05191.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42558–42583. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ribar24a.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of

Proceedings of Machine Learning Research, pp. 47901–47911. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

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*, pp. 10–19, June 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *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.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. XAttention: Block sparse attention with antidiagonal scoring. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69819–69831. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/xu25ag.html.

Yufei Xu, Fanxu Meng, Fan Jiang, Yuxuan Wang, Ruijie Zhou, Zhaohui Wang, Jiexi Wu, Zhixin Pan, Xiaojuan Tang, Wenjie Pei, Tongxuan Liu, Di Yin, Xing Sun, and Muhan Zhang. Hisa: Efficient hierarchical indexing for fine-grained sparse attention, 2026. URL https://arxiv.org/abs/2603.28458.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

Wen Zan, Jiaqi Zhang, Jianchao Tan, Hong Liu, Cunguang Wang, Xiang Li, Duyue Ma, Guanyu Wu, Yifan Lu, Fengcun Li, Yerui Sun, Peng Pei, Yuchen Xie, and Xunliang Cai. Longcat sparse attention: Taming the lightning via streaming-aware hierarchical cross-layer indexing, 2026. URL https://arxiv.org/abs/2608.01662.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. SpargeAttention: Accurate and training-free sparse attention accelerating any model inference. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76397–76413. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhang25ch.html.

Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, Ao Sun, Xu Han, and Zhiyuan Liu. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation, 2025. URL https://arxiv.org/abs/2509.24663.

Yifan Zhou, Zeqi Xiao, Tianyi Wei, Shuai Yang, and Xingang Pan. Trainable log-linear sparse attention for efficient diffusion transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9424–9433, June 2026.

APPENDIX CONTENTS

Appendix A Analysis of Block Scoring 15
Appendix B Implementation Details 16
Appendix C Experimental setup 17
Appendix D Additional Experiments 18
Appendix E Limitations 20

A ANALYSIS OF BLOCK SCORING

For a fixed query head, a visible block's full-attention mass is proportional to the exponential of its raw-key LSE score, so both give the same block ranking. Intermediate blocks are instead scored from child mean summaries, which do not retain all raw-key information.

A.1 INTERMEDIATE BLOCK SCORING

At intermediate levels, PISA avoids reading all keys in a parent block by scoring its child mean summaries. We compare LSE over these summaries with averaging their logits, using the raw-key LSE as the reference.

Fix a query q and one query head. Let B be a fully visible block with g equally sized children $C_1, \dots, C_g$. Assume that every preceding pooling step also combines equally sized groups, so each stored child summary equals the mean of its original keys. Writing $z_t = q^\top k_t/\sqrt{d}$, define the child mean logits and the parent's normalized raw-key LSE as

$$m_j = \frac{1}{|C_j|}\sum_{t\in C_j} z_t, \qquad F_q(B) = \log\left(\frac{1}{|B|}\sum_{t\in B} e^{z_t}\right). \tag{20}$$

The scaled dot product between q and child j's mean summary equals m_j. Under these conditions,

$$\frac{1}{g}\sum_{j=1}^{g} m_j \;\le\; \widehat{F}(B) := \log\left(\frac{1}{g}\sum_{j=1}^{g} e^{m_j}\right) \;\le\; F_q(B). \tag{21}$$

The left term is the intermediate mean score used by PISA-1. The middle term is the PISA score minus $\log g$, and the right term is the parent raw-key LSE minus $\log|B|$. These normalizations remove the input-count terms so that the scores can be compared on the same scale; they do not change rankings among candidates with the same input counts.

For the left inequality in Equation 21, the convexity of the exponential function allows Jensen's inequality to be applied to the g child mean logits:

$$\exp\left(\frac{1}{g}\sum_{j=1}^{g} m_j\right) \le \frac{1}{g}\sum_{j=1}^{g} e^{m_j}.$$

Since the logarithm is increasing, taking logarithms on both sides gives $g^{-1}\sum_j m_j \le \widehat{F}(B)$.

For the right inequality, apply Jensen separately to the raw-key logits within each child C_j:

$$e^{m_j} = \exp\left(\frac{1}{|C_j|}\sum_{t\in C_j} z_t\right) \le \frac{1}{|C_j|}\sum_{t\in C_j} e^{z_t}.$$

The children partition B and each has $|C_j| = |B|/g$ tokens. Averaging the inequalities over the g children therefore gives

$$\frac{1}{g}\sum_{j=1}^{g} e^{m_j} \le \frac{1}{g}\sum_{j=1}^{g}\frac{1}{|C_j|}\sum_{t\in C_j} e^{z_t} = \frac{1}{|B|}\sum_{t\in B} e^{z_t}.$$

Taking logarithms gives $\widehat{F}(B) \le F_q(B)$, completing the two inequalities.

Thus, for the same block, normalized LSE over child means is at least as close to the normalized raw-key LSE as the mean score is. It uses differences among child means, but cannot recover variation within each child. When all logits within each child are equal, $\widehat{F}(B) = F_q(B)$. This comparison concerns the numerical error of a block score; it does not guarantee more accurate block rankings or recovery of the globally highest-mass leaf blocks.

A.2 First- and second-order approximations

PISA-1 uses the first-order Taylor approximation of normalized LSE, while PISA-2 also retains the second-order term. For one candidate and query head, let $z_1, \dots, z_m$ be the valid logits entering its score, with $m \ge 1$. Define their mean and population variance by

$$\bar{z} = \frac{1}{m}\sum_{j=1}^{m} z_j, \qquad v_z = \mathrm{Var}(z) = \frac{1}{m}\sum_{j=1}^{m}(z_j - \bar{z})^2. \tag{22}$$

To derive the approximations, introduce

$$\phi(\lambda) = \log\left(\frac{1}{m}\sum_{j=1}^{m} e^{\lambda z_j}\right). \tag{23}$$

Here, $\phi(1) = s_{\mathrm{PISA}} - \log m$ is normalized LSE. Direct differentiation gives $\phi(0) = 0$, $\phi'(0) = \bar{z}$, and $\phi''(0) = v_z$. Taylor's theorem about $\lambda = 0$, evaluated at $\lambda = 1$, gives

$$\phi(1) = \bar{z} + \frac{1}{2}v_z + \frac{1}{2}\int_0^1 (1-\lambda)^2 \phi'''(\lambda)\, d\lambda. \tag{24}$$

Keeping only the mean gives PISA-1; adding half the variance gives PISA-2. In contrast, PISA computes LSE without truncating the expansion:

$$s_{\mathrm{PISA}} = \log\sum_{j=1}^{m} e^{z_j}, \qquad s_{\text{PISA-1}} = \bar{z}, \qquad s_{\text{PISA-2}} = \bar{z} + \tfrac{1}{2}v_z. \tag{25}$$

A.3 Grouped-head scoring

In grouped-query attention (Ainslie et al., 2023), PISA selects one shared block set for query heads that share a KV head. For a fixed query position, let $\mathcal{H}(h)$ denote the query heads sharing KV head h, with $G_Q = |\mathcal{H}(h)|$. PISA sums their LSE scores:

$$U_B = \sum_{h' \in \mathcal{H}(h)} s_{h',B}, \tag{26}$$

where $s_{h',B}$ is the score of block B for query head h'. The full-attention reference in the diagnostic instead averages their block masses:

$$P_h(B) = \frac{1}{G_Q}\sum_{h' \in \mathcal{H}(h)} p_{h'}(B), \tag{27}$$

where $p_{h'}(B)$ is the total attention probability assigned to block B by head h'. Even with exact raw-key LSE scores, these two aggregation rules can rank blocks differently.

B Implementation Details

Causal masking. Candidates outside the sequence or starting after the query are excluded. We retain the first, previous, and current leaf blocks and their ancestor paths whenever distinct and eligible, following the NSA implementation in Flash Linear Attention (Yang & Zhang, 2024). Prefill summaries on the current block's path can contain future keys, but these nodes are retained without using their scores for ranking. All other eligible nodes lie entirely in the query's past, so future keys cannot affect selection. The attention kernel separately masks future tokens within selected blocks.

During backpropagation, the selected block indices are held fixed: gradients flow through attention on the selected entries, not through the discrete selection decisions.

Decode cache. The cached pyramid stores $O((N/C)d)$ elements per KV head, and each level's capacity grows by a fixed fraction when more space is needed. Each level therefore expands only a constant number of times as the sequence length doubles; even if every expansion copies the entire summary buffer, the total copying cost through length N is $O((N/C)d \log N)$ per KV head. At fixed model and selection parameters, these occasional expansions preserve the $O(\log N)$ average decoding cost per generated token.

Table 4: Fixed-length block-selection latency from 4K to 256K. Q1, Q2, and Q4 reuse one leaf-key tile across up to one, two, and four query references, respectively. Boldface marks the lowest latency at each sequence length.

Block-selection latency (ms)					
N	Per-query	Q1	Q2	Q4	BSA
4K	0.573228	0.716264	0.702543	0.701181	**0.167007**
8K	1.028081	1.175723	0.908852	0.864647	**0.462596**
16K	2.049632	2.295943	1.696250	1.581351	**1.461185**
32K	4.243254	4.684515	3.429051	**3.184846**	5.098899
64K	9.040538	9.716966	7.183495	**6.704480**	19.143526
128K	18.813094	20.466775	15.466555	**14.452411**	76.681923
256K	41.732992	43.365215	33.521616	**31.440747**	312.955505

Block-selection benchmarks. Table 4 reports fixed-length block-selection timings for Figure 3. We use random BF16 queries and keys with a fixed seed, batch size 1, 32 query heads, two KV heads, $d = 64$, $C = 64$, $K = 8$, and $g = 2$, retaining the first, previous, and current blocks whenever eligible. Dot products accumulate in FP32, as do score reductions. The per-query variant fuses all selection levels per query and KV head; Q1, Q2, and Q4 instead use separate leaf-scoring programs with one, two, and four query slots, respectively. Timings use Triton's benchmarking utility after warm-up and include summary construction and all selection stages, but exclude sparse attention.

Q4 is slower than per-query at 4K but faster from 8K onward, reaching $1.35\times$, $1.30\times$, and $1.33\times$ speedups at 64K, 128K, and 256K, respectively. BSA is fastest through 16K; Q4 has the lowest reported latency from 32K to 256K.

C Experimental setup

C.1 Model and training configuration

All models use the GPT-2 BPE tokenizer (Radford et al., 2019) with a vocabulary of 50,257 tokens; the embedding and output matrices are padded to 50,432 rows. Decoder blocks use pre-RMSNorm, SiLU-gated feed-forward layers, and no dropout. RoPE is applied to the high-frequency half of each attention head with a base of 10,000. Table 5 summarizes the architecture dimensions and from-scratch training settings.

At each model scale, the training data, decoder backbone, and attention projections are matched across methods. Sparse methods use $C = 64$ and $K = 8$. We follow the default policy in the NSA implementation of Flash Linear Attention (Yang & Zhang, 2024), retaining the first, previous, and current blocks whenever eligible. PISA, PISA-1, and PISA-2 share the same hierarchy, candidate-expansion rule, and selected-attention code; they differ only in the block score. BSA uses NSA's selected-attention branch with non-hierarchical block selection. Unlike PISA and its variants, it normalizes candidate scores within each query head before summing across heads.

Continued pretraining. We use $C = 64$ and $K = 32$ for sparse methods during 10B tokens of continued pretraining at 16K. Each run uses the same block budget during training and evaluation.

Each continued-pretraining run starts from the corresponding 4K pretrained checkpoint, restoring model parameters and Adam optimizer states. We extend the sequence length to 16K and increase the RoPE base from 10,000 to 80,000. Each run uses a new learning-rate schedule: linear warm-up to

Table 5: From-scratch training configuration.

Field	418M	1.47B	2.67B
Layers	24	24	32
Model dimension	1024	2048	2560
MLP dimension	2816	5632	6912
Query heads / KV heads	32 / 2	32 / 2	32 / 2
Head dimension	64	128	128
Sequence length	4096	4096	4096
Per-device batch / accumulation	8 / 2	4 / 2	4 / 2
Training steps / warm-up	100K / 1K	100K / 1K	100K / 1K
Training budget (B tokens)	100	100	100
Peak LR	3×10^{-4}	3×10^{-4}	3×10^{-4}
Seed	42	42	42
Optimizer	AdamW, $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1		
LR schedule	1K linear warm-up; 90K plateau; 9K square-root decay to $0.1\times$ peak		
Gradient clipping	global norm 1.0		
FSDP precision	bfloat16 parameters, FP32 reductions		

3×10^{-5} over the first 10% of training steps, followed by cosine decay to 3×10^{-6}. Packed inputs preserve document boundaries, with no cross-document attention.

C.2 Evaluation details

Language modeling and multiple-choice evaluation. We use the Language Model Evaluation Harness (Biderman et al., 2026) for WikiText word perplexity (Merity et al., 2016), LAMBADA perplexity (Paperno et al., 2016), and eight multiple-choice tasks: BoolQ, PIQA, HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge, OpenBookQA, and SocialIQA (Clark et al., 2019; Bisk et al., 2019; Zellers et al., 2019; Sakaguchi et al., 2019; Clark et al., 2018; Mihaylov et al., 2018; Sap et al., 2019). WikiText and LAMBADA use the standard `wikitext` and `lambada_openai` tasks; their reported average is the arithmetic mean of the two perplexities. For multiple-choice evaluation, we report acc-n for HellaSwag, ARC-Challenge, and OpenBookQA, and acc for the other five tasks, as indicated in Table 2. The multiple-choice average (Acc-8) gives equal weight to these eight task scores. These tasks and metrics are unchanged after continued pretraining.

Containment evaluation. We evaluate six retrieval tasks: SWDE, SQuAD Completion, FDA, TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019), and DROP (Dua et al., 2019). We use the task implementations from BASED and JRT (Arora et al., 2024a;b). Contains counts a prediction as correct if it includes any gold answer as a case-insensitive literal substring. All six tasks use zero-shot greedy decoding on the validation split, generating at most 48 tokens and stopping at a newline or EOS.

RULER evaluation. We evaluate the continued-pretraining models on four needle-in-a-haystack families from RULER (Hsieh et al., 2024): single-key, multi-key, multi-query, and multi-value. We use greedy decoding at context lengths of 1K, 2K, 4K, 8K, and 16K, as reported in Table 3. The single-key and multi-key scores each average three task templates, while multi-query and multi-value each use one template. Avg is the mean over the four families and five context lengths, computed before rounding.

D Additional Experiments

D.1 Downstream evaluation after continued pretraining

Table 6 reports language modeling and downstream performance after 10B tokens of continued pretraining at 16K. PISA has slightly lower training loss than BSA and PISA-1 at both scales, with perplexity and average multiple-choice accuracy comparable to BSA. Its average containment accuracy is lower than BSA at both scales.

Table 6: Training loss and downstream performance for the 1.47B and 2.67B models after 10B tokens of continued pretraining at 16K. Sparse methods, including NSA (Yuan et al., 2025), use $C = 64$ and $K = 32$. Loss is the final training loss. Gray columns show group averages; the containment average covers SWDE, SQuAD Completion, and FDA. The block budget does not apply to Full Attention.

				Perplexity			Multiple-choice									Containment			
Method	Params	K	Loss ↓	Wiki. ppl↓	LMB. ppl↓	Avg ppl↓	BoolQ acc↑	PIQA acc↑	Hella. acc-n↑	Wino. acc↑	ARC-e acc↑	ARC-c acc-n↑	OBQA acc-n↑	SIQA acc↑	Avg acc↑	SWDE acc↑	SQuAD acc↑	FDA acc↑	Avg acc↑
10B CPT, 1.47B																			
Full Attention	1.47B	–	1.6993	13.33	10.04	11.69	51.65	73.99	58.28	57.85	71.55	38.05	41.00	40.28	54.08	76.87	46.68	75.14	66.23
NSA	1.47B	32	1.6990	13.42	9.67	11.55	63.82	73.61	58.47	59.67	72.47	37.97	42.60	40.23	56.11	74.44	44.84	72.23	63.84
BSA	1.47B	32	1.7039	13.36	9.92	11.64	56.45	74.32	58.64	59.35	71.51	37.37	40.00	40.84	54.81	76.06	44.54	79.04	66.55
PISA-1	1.47B	32	1.7042	13.45	9.95	11.70	59.57	73.39	58.32	58.88	71.04	37.20	40.00	40.17	54.82	74.44	44.27	71.05	63.25
PISA-2	1.47B	32	1.7007	13.38	9.81	11.59	63.39	73.39	58.77	57.06	70.71	38.31	40.60	39.76	55.25	72.19	46.45	73.32	63.99
PISA	1.47B	32	1.7010	13.37	9.99	11.68	61.59	74.32	58.38	57.93	71.30	39.93	41.40	40.99	55.73	74.89	45.41	75.68	65.33
10B CPT, 2.67B																			
Full Attention	2.67B	–	1.6312	11.96	8.12	10.04	64.98	75.19	62.71	61.25	74.54	42.49	41.80	40.28	57.91	77.23	48.19	75.41	66.94
NSA	2.68B	32	1.6243	11.77	7.70	9.73	65.90	75.03	63.31	60.46	73.57	41.98	42.00	41.56	57.98	79.84	48.83	75.68	68.12
BSA	2.67B	32	1.6324	11.91	7.99	9.95	63.39	75.19	63.20	61.48	74.07	40.10	42.60	40.48	57.56	77.68	47.65	79.13	68.15
PISA-1	2.67B	32	1.6331	11.96	8.03	10.00	63.61	74.97	63.31	61.96	74.49	41.30	42.40	41.30	57.92	77.68	48.39	73.96	66.68
PISA-2	2.67B	32	1.6302	11.90	7.91	9.91	57.19	75.03	62.91	61.01	75.21	42.92	43.20	40.07	57.19	75.61	48.76	80.85	68.41
PISA	2.67B	32	1.6297	11.88	7.95	9.91	59.45	75.73	63.31	61.64	73.91	43.94	41.20	40.17	57.42	77.50	49.10	72.60	66.40

D.2 Block-selection diagnostic

We replay all selectors on identical position-encoded query and key tensors from a 418M Full Attention model pretrained for 100B tokens at 4K. The model weights are fixed, and no additional training is performed. We use all 24 layers, $C = 64$, and $K = 8$. We sample 100 FDA prompts with seed 0, restricting prompt lengths to 513–4096 tokens; the sampled prompts contain 618–2001 tokens. We evaluate zero-based query positions $i \geq 512$, where more than eight blocks are eligible.

Section 4.4 defines the full-attention reference set, Recall@K, and attention mass ratio. For KV head h, let $\mathcal{H}(h)$ contain its query heads and let $\mathcal{E}_i$ contain the causally available leaf blocks at query position i. We define the log mass of block B for query head h' and its grouped mass as

$$\begin{aligned} a_{h',i,h}(B) &= \log \sum_{\substack{t \in B \\ t \leq i}} \exp\left(\frac{q_{i,h'}^{\top} k_{t,h}}{\sqrt{d}}\right), \\ P_{i,h}(B) &= \frac{1}{|\mathcal{H}(h)|} \sum_{h' \in \mathcal{H}(h)} \frac{e^{a_{h',i,h}(B)}}{\sum_{B' \in \mathcal{E}_i} e^{a_{h',i,h}(B')}}. \end{aligned} \tag{28}$$

Here, $q_{i,h'}$ and $k_{t,h}$ are the query and key vectors at positions i and t, respectively. Thus, $P_{i,h}(B)$ is the share of full-attention mass assigned to B, averaged over the query heads that share KV head h.

The selector and full-attention reference use the same eligible blocks, budget, and forced-block policy, retaining the first, previous, and current blocks. The reference fills the remaining slots with the eligible blocks of highest full-attention mass. Recall@8 therefore includes a guaranteed three-block overlap. The reference's average of normalized attention masses differs from the sum of query-head LSE scores used by PISA (Appendix A.3); the two can rank blocks differently. We also report captured mass:

$$\mathrm{CapturedMass}_{i,h} = \sum_{B \in I^{i,h}_{\mathrm{sparse}}} P_{i,h}(B). \tag{29}$$

Captured mass is the fraction of total full-attention mass retained. Attention mass ratio divides it by the mass captured by the reference set $I^{i,h}_{\mathrm{full}}$, not by total full-attention mass. Aggregate scores assign equal weight to each $(\mathrm{prompt}, \mathrm{position}, \mathrm{layer}, \mathrm{KV\ head})$ decision, so longer prompts contribute more decisions. At each query position, Figure 2 averages over the available prompts, layers, and KV heads without binning or smoothing.

Table 7: Block-selection quality on 100 FDA prompts using shared Full-Attention queries and keys, with $C = 64$ and $K = 8$. Scores are percentages, averaged over query rows with more than eight eligible blocks. Recall@8 includes the three forced blocks. Boldface marks the best result in each column.

Selector	Recall@8 ↑	Captured mass ↑	Attention mass ratio ↑
BSA	85.91	85.67	98.42
PISA-1	84.75	85.50	98.22
PISA-2	88.24	86.13	99.02
PISA	**90.95**	**86.47**	**99.46**

Table 7 shows that PISA has the highest Recall@8, captured mass, and attention mass ratio in this diagnostic. The selected blocks capture nearly as much attention mass as the full-attention reference.

E LIMITATIONS

Computational constraints limit the model sizes and pretraining data budgets explored in our experiments. Both factors can substantially affect performance and relative gains over baselines. Within the evaluated settings, our central finding remains that pyramid block selection reduces selection complexity while maintaining competitive model performance.