# PROBEREFINE: HEAD-ADAPTIVE QUERY PROBING AND RESIDUAL REFINEMENT FOR SPARSE ATTENTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large language models (LLMs) face substantial computational overhead during long-context prefilling due to the quadratic complexity of self-attention. Dynamic sparse attention alleviates this cost by selecting only a subset of tokens or blocks for computation, while proxy-based methods further reduce selection overhead by sharing block rankings across attention heads. However, head-specific budgets only determine how many blocks each head retains from a shared ranking, leaving limited flexibility in which blocks are selected when head preferences differ. In this paper, we introduce **ProbeRefine**, a training-free sparse attention method that preserves efficient shared proxy estimation while enabling head-specific block selection. ProbeRefine achieves this through two key designs: 1) **Query-Guided Head-Specific Importance Estimation**, which selects compact query probes for each head to estimate block importance and determine head-specific budgets; and 2) **Gated Residual Block Replacement**, which revisits a bounded set of blocks beyond the shared cutoff and performs one-for-one replacement when a candidate receives sufficient head-specific importance and proxy support. Together, these designs allow different heads to refine their block selections without increasing the allocated block count or constructing full per-head attention maps. Experiments across Qwen2.5, Llama3.1, and Yi on RULER and InfiniteBench show that ProbeRefine consistently improves over ProxyAttn on both benchmarks and outperforms FlexPrefill across all three backbones on RULER while remaining competitive on InfiniteBench.

## 1 INTRODUCTION

Long-context large language models (LLMs) support increasingly extensive inputs (Team et al., 2024; Grattafiori et al., 2024), enabling applications in document understanding (Bai et al., 2024; Zhang et al., 2024) and repository-level software development (Liu et al., 2024b; Jimenez et al., 2024). However, larger context windows do not guarantee reliable use of relevant information (Liu et al., 2024a; Hsieh et al., 2024), while the quadratic cost of self-attention makes long-prompt prefill expensive (Vaswani et al., 2017). Efficient inference must therefore reduce computation while preserving access to task-relevant evidence.

Existing approaches improve attention kernels (Dao et al., 2022; Dao, 2024), optimize KV-cache management (Kwon et al., 2023), or restrict retained history for streaming inference (Xiao et al., 2024). Dynamic sparse attention directly reduces query–key computations through input-dependent selection (Jiang et al., 2024; Lai et al., 2025; Xu et al., 2025), but requires accurate importance estimates at low cost. ProxyAttn (Wang et al., 2026) addresses this trade-off by pooling attention heads, computing token-level interactions, and aggregating them into shared block scores. Separate per-head budgets then accommodate differences in attention concentration.

Nevertheless, inter-head redundancy coexists with head specialization (Michel et al., 2019; Voita et al., 2019). For a fixed query block, per-head budgets change the cutoff in a shared ranking, but cannot express different relative preferences among blocks. A block important to one head may consequently fall below its selection threshold; recovering it by expanding the shared prefix also retains potentially less useful blocks. We call this restriction the shared-ranking bottleneck. It motivates adapting which blocks each head selects, in addition to how many, while retaining the efficiency of proxy-based estimation.



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.

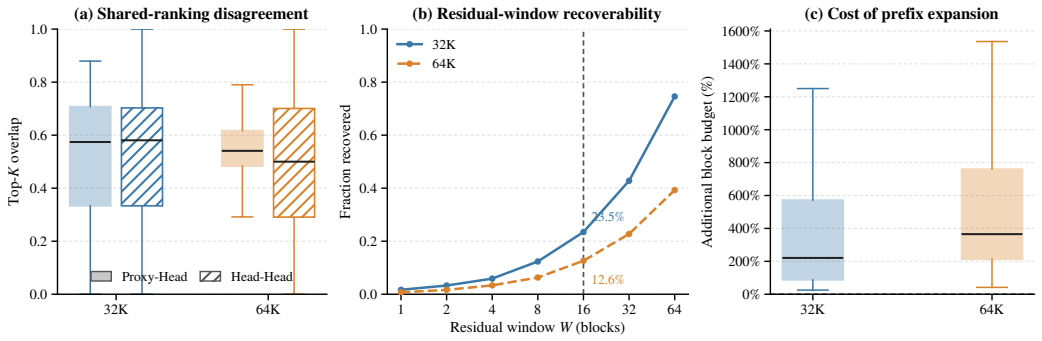


Figure 1: Diagnostics of the shared-ranking bottleneck. (a) Top-$K$ overlap between shared proxy and head-specific reference selections (Proxy–Head), together with the overlap between head-specific reference selections within the same proxy group (Head–Head). (b) Fraction of missed head-specific blocks recoverable within different residual windows beyond the shared cutoff; the dashed line marks the default $W = 16$. (c) Additional block budget required to recover missed head-specific blocks through prefix expansion.

We introduce ProbeRefine, a training-free sparse-prefill method that preserves the efficiency of shared proxy estimation while recovering head-specific block preferences beyond a shared ranking. ProbeRefine first performs query-guided head-specific importance estimation, selecting a compact set of query probes independently for each head to estimate its block-importance distribution. The number and composition of these probes adapt to the context scale, providing head-specific evidence for both budget allocation and block preference estimation. ProbeRefine then applies gated residual block replacement, which examines a bounded set of candidates beyond the shared selection cutoff and performs a one-for-one replacement only when the candidate provides sufficient head-specific gain while retaining adequate proxy support. This refinement allows each head to adjust which blocks are selected while preserving its allocated block count, without constructing full per-head attention maps or training an auxiliary selector.

We evaluate ProbeRefine across three long-context model families Qwen2.5-7B-Instruct-1M (Yang et al.), Llama3.1-8B-Instruct (Grattafiori et al., 2024), and Yi-9B-200K (Young et al., 2024) on RULER (Hsieh et al., 2024) and InfiniteBench (Zhang et al., 2024). ProbeRefine consistently outperforms other methods across all three backbones on RULER while remaining competitive on InfiniteBench. Further ablations support the effectiveness of head-specific query probing and budget-preserving residual refinement.

Our contributions can be summarized as follows:

- We identify the shared-ranking bottleneck in proxy-guided sparse attention: head-specific budgets adapt how many blocks each head retains, while a shared ranking restricts which blocks it can prefer.
- We propose ProbeRefine, which combines query-guided head-specific importance estimation with gated residual block replacement to recover head-specific block preferences beyond the shared cutoff through budget-preserving one-for-one correction, without training an auxiliary selector.
- We evaluate ProbeRefine across three long-context model families on RULER and InfiniteBench. ProbeRefine consistently improves over ProxyAttn across all three backbones on both benchmarks and outperforms FlexPrefill across all three backbones on RULER, with ablations further validating its key design choices.

## 2 Motivation

**Problem formulation.** Consider a self-attention layer with $H$ query heads, sequence length $L$, and head dimension $d$. We partition the sequence into $N(L) = \lceil L/B \rceil$ blocks of size $B$. Let $\mathcal{B}_i$ denote

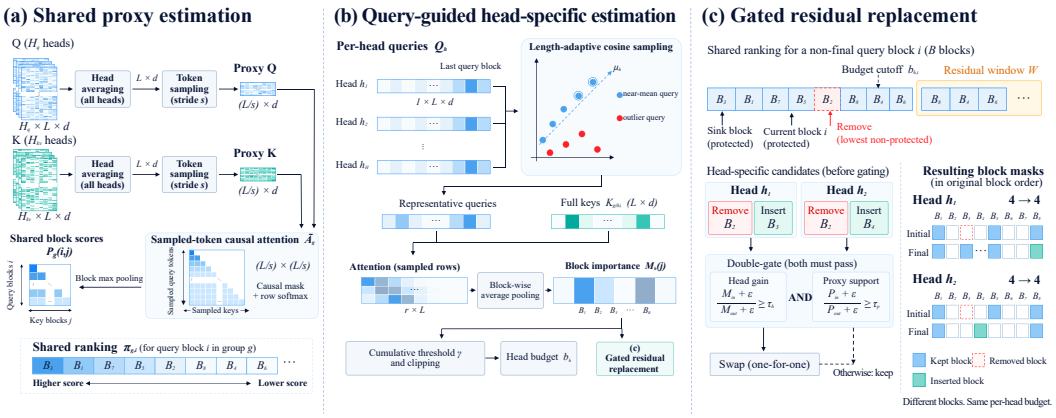


Figure 2: Overview of the proposed ProbeRefine framework.

query block $i$, and $\mathcal{S}_{h,i} \subseteq \{1, \ldots, i\}$ denote the key blocks selected for head $h$. Sparse attention restricts computation to $\mathcal{S}_{h,i}$, making its construction central to the trade-off between efficiency and attention coverage.

**Shared ranking vs. head-specific preference.** Proxy-based sparse attention reduces selection overhead by sharing a block ranking across multiple query heads while assigning head-specific budgets. However, different budgets only change *how far* each head truncates the shared ranking, not *which* blocks it prefers. As shown in Fig. 1(a), the Proxy–Head overlap remains far from complete at both 32K and 64K, revealing substantial disagreement between shared proxy and head-specific selections. The Head–Head overlap further shows that heads within the same proxy group can prefer different key blocks. Thus, adapting *how many* blocks each head retains does not fully adapt *which* blocks it selects.

**Recovering blocks beyond the shared cutoff.** Rather than reconstructing a complete ranking for every head, we examine whether a bounded search beyond the shared cutoff can recover missed head-specific blocks. As shown in Fig. 1(b), a residual window of $W = 16$ covers 23.5% and 12.6% of the missed blocks at 32K and 64K, respectively, providing a practical opportunity for lightweight head-specific correction. Simply expanding the shared prefix to reach these blocks, however, requires retaining all intervening blocks and can incur substantial additional budget, as shown in Fig. 1(c). This motivates selective replacement rather than prefix expansion.

**Design implication.** These observations motivate retaining the shared proxy ranking as an efficient initial estimate while allowing each head to selectively correct its block selection. Accordingly, ProbeRefine uses compact head-specific query evidence to identify preferences underrepresented by the shared ranking and performs gated one-for-one replacement within a bounded residual window. This design adapts *which* blocks are selected without increasing the allocated block count or reconstructing a full ranking for every head.

## 3 METHOD

ProbeRefine consists of three stages: shared proxy importance estimation, query-guided head-specific importance estimation, and gated residual block replacement. The second stage further adapts the number and composition of query probes to the context scale $N(L)$ (see Appendix A). The resulting head-specific estimates guide both budget allocation and residual correction.

### 3.1 SHARED PROXY IMPORTANCE ESTIMATION

Following ProxyAttn (Wang et al., 2026), we partition the $H$ query heads into $G$ proxy groups. Let $\mathcal{G}_g$ denote group $g$ and $g(h)$ the group containing head $h$. We average query and key representations

within each group:

$$\bar{Q}_g = \frac{1}{|\mathcal{G}_g|} \sum_{h \in \mathcal{G}_g} Q_h, \qquad \bar{K}_g = \frac{1}{|\mathcal{G}_g|} \sum_{h \in \mathcal{G}_g} K_h. \tag{1}$$

Groups can include all query heads or follow the model's native KV-sharing structure, where query heads sharing the same KV head form one group. This grouping only determines how proxy scores are shared and leaves the original attention structure unchanged.

Let $\mathcal{D}_i$ denote positions sampled from block $\mathcal{B}_i$ with stride $s$. We compute causal attention probabilities $\bar{A}_g$ between sampled proxy queries and keys, and obtain block-level proxy scores by max pooling:

$$P_g(i,j) = \max_{\substack{t \in \mathcal{D}_i \\ u \in \mathcal{D}_j}} \bar{A}_g(t,u), \qquad j \le i. \tag{2}$$

For each query block $i$, sorting the visible key blocks by $P_g(i,j)$ gives the shared proxy ranking $\pi_{g,i}$. Heads within the same group share this ranking, but retain independent block budgets and subsequent residual corrections.

### 3.2 Query-Guided Head-Specific Importance Estimation

To recover head-specific block preferences beyond the shared ranking, we select a small set of query probes from the last query block of each head and use them to estimate an independent block-importance distribution. This distribution guides both head-specific budget allocation and the residual correction in Section 3.3.

**Scale-aware query probe selection.** For each head, we select query probes from its last query block $\widehat{Q}_h = Q_h[-B:]$. Each query is scored by its cosine similarity to the mean query direction:

$$c_{h,t} = \cos\Big(q_{h,t}, \mathrm{Mean}(\widehat{Q}_h)\Big). \tag{3}$$

Low and high similarity queries form the outlier and representative probes, respectively. Their number and composition adapt to the context scale $N(L)$, using outlier probes at the base scale and both types at longer contexts. The exact allocation rule is provided in Appendix A.

**Block importance and head-specific budget.** For each head, we aggregate the attention probabilities of its query probes over each key block:

$$M_h(j) = \frac{1}{|\mathcal{P}_h(L)|\,|\mathcal{B}_j|} \sum_{t \in \mathcal{P}_h(L)} \sum_{\substack{u \in \mathcal{B}_j \\ u \le t}} A_h(t,u), \tag{4}$$

where $M_h(j)$ measures the head-specific importance of key block $j$. Unlike the shared proxy ranking, $M_h$ is estimated independently for each head and guides both budget allocation and residual correction.

Let $M_h^{\downarrow}$ denote the block importance sorted in descending order. We choose the smallest number of blocks whose cumulative importance exceeds a fraction $\gamma$ of the total importance:

$$k_h = \min\left\{ k : \frac{\sum_{\ell=1}^{k} M_h^{\downarrow}(\ell)}{\sum_{j=1}^{N} M_h(j)} > \gamma \right\}. \tag{5}$$

We then set $b_h = \mathrm{clip}(k_h + 1, b_{\min}, b_{\max})$ and $b_{h,i} = \min(b_h, i)$ to enforce the budget range and causal visibility. Thus, heads sharing the same proxy ranking retain independent importance estimates and block budgets.

### 3.3 Gated Residual Block Replacement

Head-specific budgets determine how many blocks each head retains, but the initial selection still follows the shared proxy ranking. We therefore use the head-specific importance $M_h$ to refine blocks near the selection boundary without increasing the allocated budget.

**Residual candidates and block replacement.** For head $h$ and query block $i$, let $\mathcal{S}^0_{h,i}$ contain the top $b_{h,i}$ causally visible blocks in the shared ranking $\pi_{g(h),i}$. We form a residual candidate set $\mathcal{U}_{h,i}$ from the next $W$ unselected blocks, excluding protected blocks $\mathcal{A}_i$ such as the sink and current blocks. We select the lowest-ranked removable block for removal and the candidate with the highest head-specific importance for insertion:

$$\begin{aligned} j_{\text{out}} &= \operatorname*{arg\,max}_{j \in \mathcal{S}^0_{h,i} \setminus \mathcal{A}_i} \operatorname{rank}_{\pi_{g(h),i}}(j), \\ j_{\text{in}} &= \operatorname*{arg\,max}_{j \in \mathcal{U}_{h,i}} M_h(j). \end{aligned} \tag{6}$$

This bounded search preserves the shared proxy ranking as the initial selection while allowing head-specific corrections near its cutoff.

**Dual-gated correction.** Since $M_h$ is estimated from a small set of query probes, we accept a replacement only when the incoming block has sufficient head-specific gain and proxy support:

$$\frac{M_h(j_{\text{in}}) + \epsilon}{M_h(j_{\text{out}}) + \epsilon} \geq \tau_h, \qquad \frac{P_{g(h)}(i, j_{\text{in}}) + \epsilon}{P_{g(h)}(i, j_{\text{out}}) + \epsilon} \geq \tau_p. \tag{7}$$

Here, $\tau_h \geq 1$ enforces head-specific improvement, while $0 < \tau_p \leq 1$ filters candidates with insufficient proxy support. The replacement is applied only when both conditions hold.

ProbeRefine performs at most one one-for-one replacement per head and query block, preserving the allocated block budget. Protected blocks remain unchanged, and query blocks assigned dense attention bypass residual correction.

### 3.4 Computational Complexity

The shared proxy estimation costs $O(GL^2d/s^2)$, where $G$ is the number of proxy groups and $s$ is the sampling stride, compared with $O(HL^2d)$ for dense multi-head attention. For $r$ query probes per head, head-specific importance estimation adds $O(HrLd)$ computation. Budget estimation requires $O(HN \log N)$ due to per-head sorting of block-level importance scores. In the current vectorized implementation, residual refinement operates over candidate tensors of width $C$, resulting in $O(HNC)$ complexity, where $C$ is typically bounded by the allocated candidates and residual window. Residual correction performs at most one one-for-one replacement per head and query block, changing selected block identities without increasing the allocated block count. ProbeRefine requires no additional training or learned selector checkpoint.

## 4 Experiments

We evaluate ProbeRefine on long-context benchmarks to assess its task performance and computational efficiency. We further investigate the contributions of head-adaptive query probing and budget-preserving residual refinement through controlled ablations.

### 4.1 Experimental Setup

**Models.** We evaluate ProbeRefine on four long-context language models: Meta-Llama-3.1-8B-Instruct-128K (Grattafiori et al., 2024), Qwen2.5-7B-Instruct-1M (Yang et al.), Yi-9B-200K (Young et al., 2024), and GLM-4-9B-Chat-1024K (Glm et al., 2024). These models cover different architectures and context-window capacities, allowing us to examine whether the method generalizes across model families.

**Benchmarks.** We evaluate on RULER (Hsieh et al., 2024) and InfiniteBench (Zhang et al., 2024). RULER provides controlled synthetic tasks for evaluating information retrieval, multi-hop tracing, aggregation, and question answering. We evaluate its 13 tasks at six context lengths: 4K, 8K, 16K, 32K, 64K, and 128K. InfiniteBench complements this evaluation with ten synthetic and natural-language tasks, covering long-document question answering and summarization, dialogue understanding, code debugging, mathematical search, and retrieval. We report task-specific benchmark scores and their macro-average. For RULER, we additionally report the average across the six evaluated context lengths, assigning equal weight to each length.

Table 1: Performance comparison across different context lengths on Ruler.

| Method | 4k | 8k | 16k | 32k | 64k | 128k | Avg. |
|---|---|---|---|---|---|---|---|
| *Qwen2.5-7B-Instruct-1M* | | | | | | | |
| FullAttention | 95.19 | 93.03 | 92.31 | 91.11 | 87.50 | 84.62 | 90.63 |
| Streaming LLM | 94.95 | 93.27 | 74.04 | 51.92 | 33.89 | 25.00 | 62.18 |
| MInference | **95.19** | **93.51** | 92.07 | 89.90 | 87.02 | **83.17** | 90.14 |
| FlexPrefill | 94.71 | 92.07 | 92.31 | 89.18 | 83.17 | 81.73 | 88.86 |
| ProxyAttn | 93.04 | 91.55 | 93.02 | 89.45 | 85.49 | 80.77 | 88.89 |
| **Ours** | 94.95 | **93.51** | **93.03** | **91.11** | **88.70** | **83.17** | **90.75** |
| *Llama3.1-8B-Instruct* | | | | | | | |
| FullAttention | 95.67 | 93.99 | 93.51 | 87.50 | 85.10 | 79.33 | 89.18 |
| Streaming LLM | 95.67 | 93.99 | 75.48 | 44.47 | 25.72 | 30.53 | 60.98 |
| MInference | 95.67 | **94.23** | 93.75 | 86.54 | 84.38 | 62.74 | 86.22 |
| FlexPrefill | 95.43 | 93.99 | **94.71** | 89.18 | 82.45 | **79.09** | 89.14 |
| ProxyAttn | **96.16** | 93.18 | 93.76 | 89.70 | 85.97 | 77.95 | 89.45 |
| **Ours** | 95.19 | 93.99 | 94.23 | **92.07** | **87.26** | **79.09** | **90.31** |
| *Yi-9B-200K* | | | | | | | |
| FullAttention | 93.03 | 84.13 | 82.45 | 70.19 | 66.59 | 64.66 | 76.84 |
| Streaming LLM | **93.51** | 81.97 | 64.66 | 51.92 | 42.79 | 34.86 | 61.62 |
| MInference | 92.31 | 83.41 | 80.53 | 71.64 | 64.90 | 61.30 | 75.68 |
| FlexPrefill | 92.55 | **86.78** | 82.45 | 70.91 | 61.54 | 60.34 | 75.76 |
| ProxyAttn | 92.79 | 82.45 | 82.69 | **74.76** | 66.11 | 62.98 | 76.96 |
| **Ours** | 93.27 | 86.06 | **83.17** | 72.84 | **67.31** | **64.90** | **77.93** |

**Baselines.** We compare ProbeRefine with five representative attention methods. **FlashAttention** (Dao, 2024) serves as the dense-attention reference. **StreamingLLM** (Xiao et al., 2024) retains attention-sink tokens together with a local context window. **MInference** (Jiang et al., 2024) combines offline attention-pattern selection with online sparse-index construction. **FlexPrefill** (Lai et al., 2025) adapts attention patterns and computational budgets to the input. **ProxyAttn** (Wang et al., 2026) estimates block importance through representative heads and allocates head-specific budgets. Together, these baselines cover dense attention, fixed sparse patterns, and dynamic sparse selection.

**Evaluation protocol.** Within each model, all methods use the same evaluation examples, prompt formatting, context-length limits, truncation policy, and task-specific generation limits. We use greedy decoding with a batch size of one. Following prior sparse-prefill evaluations, sparsification is restricted to the prefilling stage, while decoding uses full attention. In particular, StreamingLLM is evaluated through its prefill attention pattern rather than through persistent KV-cache eviction. The same scoring implementation is used across methods to avoid differences arising from evaluation procedures.

**Implementation details.** Experiments are conducted on NVIDIA A800 GPUs with 80 GB of memory. Our implementation uses PyTorch and Triton with BF16 model inference. ProbeRefine operates on blocks of 128 tokens and combines shared proxy estimation with head-adaptive probing and gated residual refinement. The refinement preserves the allocated block count for each head and query block. Model-specific configurations, including proxy grouping, probe selection, probe count, minimum budget, residual-window size, and gate thresholds, are documented in the appendix, together with baseline configurations and software versions.

### 4.2 Main Result

**RULER** As shown in Table 1, ProbeRefine achieves the best average score on all three models, obtaining 90.75 on Qwen2.5, 90.31 on Llama-3.1, and 77.93 on Yi, while also matching or surpass-

Table 2: Main results on InfiniteBench.

| Methods | En | | | | Zh | Code | Math | Retrieval | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Sum | QA | MC | Dia | QA | Debug | Find | PassKey | Number | KV | |
| *Qwen2.5-7B-Instruct-1M* | | | | | | | | | | | |
| FullAttention | 33.37 | 11.84 | 59.39 | 20.50 | 10.15 | 18.53 | 40.00 | 25.93 | 100.00 | 64.60 | 38.43 |
| Streaming LLM | 32.68 | 6.79 | 30.13 | 13.00 | 8.64 | **19.80** | 40.00 | 0.00 | 15.08 | 5.00 | 17.11 |
| MInference | 33.35 | 10.47 | 56.33 | 15.50 | 9.95 | 11.42 | **42.29** | 21.53 | **100.00** | **64.60** | **36.54** |
| FlexPrefill | 31.52 | 11.04 | 48.47 | 13.50 | 8.86 | 14.47 | 38.86 | **31.19** | **100.00** | 50.60 | 34.85 |
| ProxyAttn | 33.61 | 11.74 | **58.52** | 13.00 | 9.89 | 13.96 | 39.14 | 13.90 | **100.00** | 58.20 | 35.20 |
| **Ours** | **33.70** | **11.87** | 57.21 | **16.00** | **10.03** | 15.48 | 38.57 | 12.37 | **100.00** | 57.80 | 35.30 |
| *Llama3.1-8B-Instruct* | | | | | | | | | | | |
| FullAttention | 32.57 | 26.25 | 69.43 | 21.50 | 33.02 | 18.53 | 24.57 | 99.32 | 99.66 | 60.60 | 48.55 |
| Streaming LLM | 30.00 | 10.51 | 40.61 | 8.50 | 23.04 | 21.07 | 17.71 | 2.71 | 5.93 | 0.00 | 16.01 |
| MInference | 31.23 | 21.32 | 64.63 | 15.50 | 27.77 | 20.81 | 25.43 | 61.69 | 76.78 | 21.60 | 36.68 |
| FlexPrefill | 32.38 | 25.74 | 69.43 | 20.50 | 33.05 | **21.57** | **30.86** | 98.81 | 99.66 | 45.00 | **47.70** |
| ProxyAttn | 31.87 | **26.98** | 69.00 | 19.00 | **33.71** | 20.30 | 24.57 | **99.15** | **99.83** | 41.40 | 46.58 |
| **Ours** | **32.67** | 26.37 | **71.18** | **21.00** | 32.20 | 20.30 | 24.00 | 98.14 | **99.83** | **45.20** | 47.09 |
| *Yi-9B-200K* | | | | | | | | | | | |
| FullAttention | 7.84 | 11.20 | 62.88 | 1.00 | 16.74 | 23.10 | 24.86 | 100.00 | 100.00 | 15.40 | 36.30 |
| Streaming LLM | 6.08 | 10.61 | 41.92 | 2.50 | 15.74 | 20.30 | **28.00** | 13.05 | 48.47 | 3.40 | 19.01 |
| MInference | 7.16 | 10.69 | 58.52 | 2.00 | 16.90 | **25.38** | 24.86 | **100.00** | **100.00** | **15.80** | 36.13 |
| FlexPrefill | 7.79 | **11.93** | **66.81** | 1.50 | 16.51 | 19.04 | 25.43 | **100.00** | 96.10 | 14.40 | 35.95 |
| ProxyAttn | 7.18 | 11.42 | 63.32 | **4.00** | 17.29 | 23.35 | 22.00 | **100.00** | 99.83 | 14.40 | 36.28 |
| **Ours** | **8.59** | 11.60 | 62.45 | 1.00 | **17.45** | 23.35 | 24.00 | **100.00** | **100.00** | 15.60 | **36.40** |

ing dense attention. Averaged across the three models, ProbeRefine improves over FlexPrefill and ProxyAttn by 1.74 and 1.23 points.

These results are consistent with the motivation of ProbeRefine. As the number of candidate blocks increases, a shared or coarse block ranking becomes more likely to miss blocks that are important to individual heads. Scale-aware representative queries provide head-specific importance estimates over the enlarged context. Gated one-for-one replacement then recovers selected missing blocks without increasing the allocated block budget. Together, these components deliver consistent gains at long context lengths and remain competitive on shorter inputs. Additional results on GLM are provided in Appendix B.

**InfiniteBench** Table 2 reports the results on InfiniteBench. ProbeRefine achieves competitive performance across all three model families, obtaining average scores of 35.30, 47.09, and 36.40 on Qwen2.5-7B-Instruct-1M, Llama3.1-8B-Instruct, and Yi-9B-200K, respectively. Compared with FlexPrefill, ProbeRefine improves the average score on both Qwen2.5 and Yi, while remaining competitive on Llama3.1. It also consistently outperforms ProxyAttn across all three models. The gains on tasks such as MC, Dialogue, and Retrieve.Number further indicate that head-specific probing and residual refinement can recover useful long-context information missed by shared block rankings. These results demonstrate that ProbeRefine generalizes across diverse long-context tasks and model architectures.

We further report end-to-end attention speedup and stage-wise latency breakdowns in Appendix E.

### 4.3 Ablations

**Effect of query selection.** We ablate different query selection strategies on Llama3.1-8B-Instruct with the 32K RULER setting to examine how query evidence affects head-specific block estimation. As shown in Table 3, directly averaging the last query block (Avg-Q) yields a score of only 80.29, suggesting that a single averaged representation can obscure informative query-level variations. Us-

Table 3: Ablation of query selection strategies. Random-Q16 reports the mean and standard deviation over three runs with seeds 42, 43, and 44. $\Delta$ denotes the score difference relative to Outlier-Q16.

| Query Evidence | Score | Δ Score |
|---|---|---|
| Avg-Q | 80.29 | $-11.78$ |
| Random-Q16 | $90.07 \pm 0.28$ | $-2.00$ |
| Representative-Q16 | 90.39 | $-1.68$ |
| **Outlier-Q16** | **92.07** | **0.00** |

Table 4: Effect of residual exchange on Llama3.1-8B-Instruct under RULER. Recall measures the retention of remote Top-2 blocks identified by Outlier-Q16 queries. Residual exchange improves this diagnostic recall without increasing the sparse-attention budget.

| Length | Variant | Score | Active Ratio | Final Recall | Δ Recall (pp) |
|---|---|---|---|---|---|
| 32K | No Swap | 91.83 | 0.4196 | 58.56% | +0.00 |
| 32K | **Swap** | **92.07** | 0.4193 | **63.20%** | +4.64 |
| 64K | No Swap | 87.26 | 0.3653 | 60.88% | +0.00 |
| 64K | **Swap** | **87.26** | 0.3649 | **63.83%** | +2.95 |

ing multiple real queries substantially improves performance: Random-Q16 reaches $90.07 \pm 0.28$, while selecting queries closest to the mean direction (Representative-Q16) further improves the score to 90.39. In contrast, Outlier-Q16, which selects queries with the lowest cosine similarity to the mean query direction, achieves the best score of 92.07, outperforming Representative-Q16 and Random-Q16 by 1.68 and 2.00 points, respectively. These results indicate that queries deviating from the dominant direction provide complementary evidence that is not sufficiently captured by averaged or representative queries. Such evidence is particularly useful for exposing head-specific block preferences, supporting our design of outlier-based query probing.

**Effect of residual exchange.** We evaluate residual exchange at 32K and 64K with a fixed residual window of $W = 16$. Here, recall is a mechanism-level diagnostic that measures whether the final sparse layout retains remote Top-2 blocks identified by the Outlier-Q16 queries. As shown in Table 4, residual exchange improves the final recall from 58.56% to 63.20% at 32K and from 60.88% to 63.83% at 64K, corresponding to gains of 4.64 and 2.95 percentage points over the no-swap variants, respectively. Meanwhile, the active ratio remains nearly unchanged, confirming that these gains arise from one-for-one block replacement rather than an increased sparse-attention budget. These results show that residual exchange improves the retention of representative remote blocks under a fixed budget, while such mechanism-level improvements do not necessarily translate into proportional gains in downstream task performance.

**Effect of residual window size.** We vary the residual window $W$ to examine how the candidate search range affects diagnostic block recall and downstream performance. As shown in Fig. 3, enlarging $W$ consistently increases both window recall and final recall, indicating that a broader residual search exposes and retains more remote Top-2 blocks identified by the Outlier-Q16 queries. However, this improvement does not translate monotonically into downstream performance. The RULER score peaks at 92.07 with $W = 16$, whereas $W = 32$ achieves the highest window and final recall but a lower score of 91.59. This divergence shows that maximizing diagnostic recall alone is insufficient for maximizing downstream performance. We therefore adopt $W = 16$, which provides the best empirical trade-off between residual candidate coverage and task performance.

## 5 Related Work

**Efficient Attention for Long Contexts.** Long-context Transformers are constrained by the quadratic complexity of self-attention. FlashAttention (Dao et al., 2022) accelerates exact attention through IO-aware kernels, while Sparse Transformer (Child et al., 2019), Reformer (Kitaev et al., 2020), Longformer (Beltagy et al., 2020), and BigBird (Zaheer et al., 2020) reduce computa-

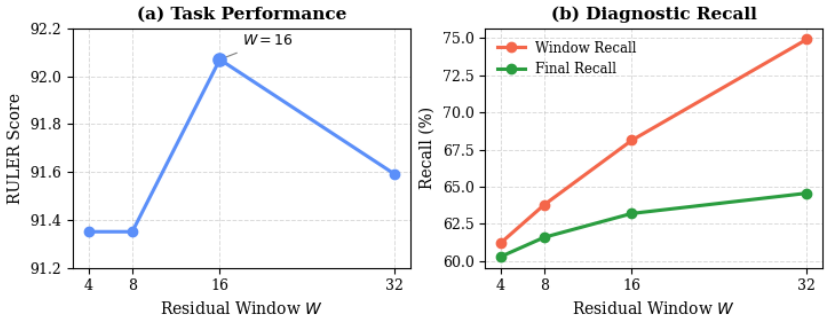


Figure 3: Effect of residual window size $W$ on Llama3.1-8B-Instruct at 32K, showing increasing diagnostic recall but peak RULER performance at $W = 16$.

tion using structured sparse connectivity. These patterns, however, are largely predefined. Learned approaches such as SeerAttention (Gao et al., 2024) and Native Sparse Attention (Yuan et al., 2025) provide greater adaptivity through sparsity-aware training. Our work instead targets training-free acceleration of pretrained long-context LLMs.

**Dynamic Sparse Attention.** Recent methods construct sparse indices online. MInference (Jiang et al., 2024) assigns recurring sparse patterns to attention heads and dynamically builds their indices. FlexPrefill (Lai et al., 2025) adapts both attention patterns and budgets to each input and head. Quest (Tang et al., 2024) performs query-aware KV-page selection for decoding, while XAttention (Xu et al., 2025) and SpargeAttention (Zhang et al., 2025) develop lightweight block-importance estimators. These methods mainly improve dynamic importance estimation along the token or block dimension; we study the complementary problem of information loss caused by sharing sparse rankings across heads.

**Inter-Head Redundancy and Sharing** Multi-head attention exhibits both redundancy (Michel et al., 2019) and task-relevant head specialization (Voita et al., 2019). MQA (Shazeer, 2019) and GQA (Ainslie et al., 2023) exploit this structure by sharing key-value heads to reduce inference cost. ProxyAttn (Wang et al., 2026) instead compresses heads during sparse-index estimation, using one block ranking per proxy group and a separate budget for each original head. Although the budgets are head-specific, heads within a group still select from the same ranking, potentially suppressing blocks important to only a subset of heads. Our method preserves the efficient shared ranking but corrects such omissions using query-diverse, head-specific residual estimation and budget-preserving block replacement.

## 6 Conclusion

We presented ProbeRefine, a training-free sparse attention method that addresses the shared-ranking bottleneck in long-context prefilling. ProbeRefine combines query-guided head-specific importance estimation with gated one-for-one block replacement, allowing each head to refine *which* blocks are selected without increasing its allocated block count. Across Qwen2.5, Llama3.1, and Yi, ProbeRefine consistently improves over ProxyAttn on RULER and InfiniteBench, and outperforms FlexPrefill across all three backbones on RULER while remaining competitive on InfiniteBench. Our analyses further support the effectiveness of head-specific query probing and budget-preserving residual refinement. Overall, ProbeRefine enables sparse attention to adapt not only how many blocks each head retains, but also which blocks it selects beyond a shared ranking.

**Limitations** ProbeRefine introduces additional selection overhead for head-specific importance estimation and residual refinement. Its probe allocation, residual window, and gating thresholds are empirically designed and may require adaptation across models and context scales. Moreover, bounded residual search cannot recover all head-specific blocks missed by the shared ranking. Broader evaluation across architectures, context lengths, and tasks would further assess the generality of the method.

# 7 STATEMENT

## AI USE STATEMENT

Generative AI tools were used to assist with language polishing and improving the clarity of the manuscript. All AI-assisted text was reviewed and revised by the authors. The research methodology, experimental design, implementation, data analysis, and scientific conclusions were developed and verified by the authors. The authors take full responsibility for the final content of this work.

## ETHICS STATEMENT

This work studies efficient sparse attention for long-context language models and does not involve human subjects, personal or sensitive data, or the release of new datasets. ProbeRefine modifies the attention computation of existing language models without introducing additional training data or model capabilities. It may therefore inherit the general ethical risks of the underlying language models, while we do not identify additional ethical concerns specific to the proposed method.

## REPRODUCIBILITY STATEMENT

We provide detailed descriptions of ProbeRefine and its experimental evaluation throughout the main paper and appendix. The methodology, experimental setup, evaluated models and benchmarks, as well as supporting ablation studies are documented to facilitate reproduction of our results. Additional implementation details will be made available to further support reproducibility.

# REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 4895–4901, 2023.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew 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.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766*, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024b.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 5797–5808, 2019.

Yixuan Wang, Huang He, Siqi Bao, Haifeng Wang, Qingfu Zhu, Wanxiang Che, et al. Proxyattn: Guided sparse attention via representative heads. In *International Conference on Learning Representations*, volume 2026, pp. 18603–18617, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. *arXiv preprint arXiv:2503.16428*, 2025.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. Qwen2. 5-1m technical report, 2025. *URL https://arxiv. org/abs/2501.15383*.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping 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)*, pp. 23078–23097, 2025.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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. *arXiv preprint arXiv:2502.18137*, 2025.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024.

**APPENDIX**

# A SCALE-AWARE QUERY ALLOCATION

Our query probing strategy adapts the probe composition to the context scale. For contexts up to 64K, we use 16 outlier queries per attention head. When the context expands to 128K, simply increasing the number of outlier queries is insufficient to simultaneously capture atypical query directions and the dominant query mode. We therefore adopt a simple and deterministic scale-aware allocation rule that expands the probe budget while introducing representative queries at the next context scale. The rule is independent of the model, task, and dataset, and its hyperparameters are determined once during development and kept fixed throughout the main experiments.

**Query characterization.** Let a sequence of length $L$ be partitioned into blocks of size $B = 128$, such that the number of blocks is

$$N(L) = \left\lceil \frac{L}{B} \right\rceil. \tag{8}$$

For each attention head $h$, we compute the mean direction of the queries in the last query block:

$$\boldsymbol{\mu}_h = \frac{1}{T_q} \sum_{t=1}^{T_q} \boldsymbol{q}_{h,t}, \tag{9}$$

where $T_q = 128$ in our implementation. We then measure the cosine similarity between each query and the mean direction:

$$s_{h,t} = \cos\left(\boldsymbol{q}_{h,t}, \boldsymbol{\mu}_h\right). \tag{10}$$

Queries with the lowest similarity scores capture directions that deviate most from the dominant query direction and are referred to as outlier queries. Conversely, queries with the highest similarity scores are referred to as representative queries, as they characterize the dominant query mode.

**Scale-aware allocation.** We use $N_0 = 512$ blocks, corresponding to 64K tokens, as the reference context scale and define the scale-transition variable

$$r(L) = \mathrm{clip}\left(\left\lceil \log_2 \frac{N(L)}{N_0} \right\rceil, 0, 1\right). \tag{11}$$

Starting from a base probe budget of $K_0 = 16$, the numbers of outlier and representative probes are respectively defined as

$$K_{\mathrm{out}}(L) = K_0 + \frac{K_0}{2} r(L), \qquad K_{\mathrm{rep}}(L) = \frac{K_0}{2} r(L). \tag{12}$$

The resulting probe set for attention head $h$ is

$$\mathcal{Q}_h(L) = \mathrm{Bottom}_{K_{\mathrm{out}}(L)}(s_h) \cup \mathrm{Top}_{K_{\mathrm{rep}}(L)}(s_h). \tag{13}$$

Under the context lengths considered in our experiments, this rule yields two operating regimes. For $L \leq 64$K, $r(L) = 0$, and we use 16 outlier probes (*Outlier-Q16*). When the context enters the next scale at 128K, $r(L) = 1$ and the probe set expands to 32 queries. Rather than assigning the entire additional capacity to increasingly atypical queries, we allocate half to extending outlier coverage and half to representative queries that capture the dominant query mode. This yields 24 outlier and 8 representative queries (*Bimodal-Q32, 24:8*), with the total number of probes bounded at 32.

The ablation in Table 5 supports this scale-dependent design. At 64K, Outlier-Q16 achieves the best score of 87.26, compared with 86.30 for Outlier-Q32 and 84.62 for Bimodal-Q32, indicating that the compact outlier set is already sufficient at this scale and that additional probes do not provide further benefit. At 128K, simply doubling the number of outlier probes is also insufficient: Outlier-Q16 and Outlier-Q32 both obtain 78.61. In contrast, Bimodal-Q32 improves the score to 79.09. Since Outlier-Q32 and Bimodal-Q32 use the same number of probes, this comparison isolates the effect of probe composition from probe count. The results suggest that, as the context and corresponding key-block search space expand, incorporating representative queries provides complementary evidence that cannot be recovered by merely increasing outlier coverage. Notably, this improvement is obtained without increasing the active attention ratio, which slightly decreases from 0.3161 with Outlier-Q16 to 0.3092 with Bimodal-Q32.

Table 5: Ablation of scale-aware query allocation on RULER. AR denotes the active attention ratio. All other sparse-attention configurations are kept fixed.

| | Probe Allocation | | | 64K | | 128K | |
|---|---|---|---|---|---|---|---|
| **Variant** | **Out.** | **Rep.** | **Total** | **Score** | **AR** | **Score** | **AR** |
| Outlier-Q16 | 16 | 0 | 16 | **87.26** | 0.3649 | 78.61 | 0.3161 |
| Outlier-Q32 | 32 | 0 | 32 | 86.30 | 0.3569 | 78.61 | 0.3110 |
| Bimodal-Q32 (24:8) | 24 | 8 | 32 | 84.62 | 0.3564 | **79.09** | 0.3092 |

Table 6: Additional RULER results on GLM-4-9B-Chat-1M across context lengths.

| **Method** | **4K** | **8K** | **16K** | **32K** | **64K** | **128K** | **Avg.** |
|---|---|---|---|---|---|---|---|
| FlexPrefill | 95.43 | **93.99** | **94.71** | 89.18 | 82.45 | **79.09** | 89.14 |
| ProxyAttn | **96.16** | 93.18 | 93.76 | 89.70 | 85.97 | 77.95 | 89.45 |
| **ProbeRefine** | 95.19 | **93.99** | 94.23 | **92.07** | **87.26** | **79.09** | **90.31** |

**Design choice.** The scale-aware allocation should be viewed as a principled empirical design rather than a theoretically optimal allocation. The reference scale $N_0 = 512$ and base probe budget $K_0 = 16$ are selected once during development and subsequently fixed across models, tasks, and datasets. The motivation is to avoid increasing query probes unnecessarily at shorter contexts, while providing additional query diversity when the key-block search space expands substantially. In particular, the bimodal allocation at 128K is intended to complement the outlier-focused probes with evidence from the dominant query mode, rather than merely increasing the number of outlier probes.

To validate this design, we compare Outlier-Q16, Outlier-Q32, and the 24:8 bimodal allocation at both 64K and 128K. As reported in Table 5, these experiments separate the effect of increasing the probe budget from that of introducing representative queries.

## B Additional Results on GLM

We further evaluate ProbeRefine on GLM-4-9B-Chat-1M (Glm et al., 2024) to examine its generalization across different attention organizations and KV-sharing patterns. GLM-4-9B-Chat-1M supports a context length of up to 1M tokens and employs 32 query attention heads with 4 multi-query groups, resulting in substantial KV sharing across query heads. This provides a complementary architecture to the model families evaluated in the main text and allows us to examine whether head-specific query probing remains effective under a different KV-sharing configuration.

As shown in Table 6, ProbeRefine achieves an average RULER score of 90.31, compared with 89.14 for FlexPrefill and 89.45 for ProxyAttn. The gains are particularly pronounced at 32K and 64K, where ProbeRefine reaches 92.07 and 87.26, respectively, compared with 89.18 and 82.45 for FlexPrefill and 89.70 and 85.97 for ProxyAttn. At 128K, the scale-aware bimodal allocation improves ProbeRefine from 78.85 with Outlier-Q16 to 79.09, matching FlexPrefill while outperforming ProxyAttn.

These results provide additional evidence that the effectiveness of ProbeRefine is not tied to a specific attention organization or KV-sharing pattern. Together with the Llama, Qwen, and Yi results in the main text, the GLM evaluation supports the transferability of head-specific query probing and budget-preserving residual refinement across heterogeneous long-context model architectures.

## C Algorithm Details

This section summarizes the complete procedure of ProbeRefine. The method consists of three stages: shared proxy importance estimation, query-guided head-specific importance estimation, and gated residual block replacement. The resulting sparse block indices are then passed to the sparse attention kernel.

**ProbeRefine Configuration.** We use $B = 128$, $s = 4$, and $W = 16$ in our experiments. Residual refinement performs at most one one-for-one replacement per head and query block, while mandatory anchors and dense terminal rows are handled separately. Although Algorithm 4 considers a residual window of size $W$, the vectorized implementation operates on candidate tensors of width $C$, yielding $O(HNC)$ implementation complexity. No additional training or learned selector checkpoint is required.

**Algorithm 1** ProbeRefine Sparse Prefill

**Require:** Query states $Q$, key states $K$; block size $B$; proxy stride $s$; coverage threshold $\gamma$; budget range $[b_{\min}, b_{\max}]$; residual window $W$; gating thresholds $\tau_h, \tau_p$
**Ensure:** Sparse block selections $\{\mathcal{S}_{h,i}\}$
$N \leftarrow \lceil L/B \rceil$
$\{P_g, \pi_{g,i}\} \leftarrow \textsc{SharedProxyImportance}(Q, K, B, s)$
**for** each query head $h$ **do**
  $(\mathcal{P}_h, M_h, b_h) \leftarrow \textsc{HeadSpecificImportance}(Q_h, K_{g(h)})$
  **for** each query block $i$ **do**
    $b_{h,i} \leftarrow \min(b_h, i)$
    $\mathcal{S}^0_{h,i} \leftarrow \textsc{TopBlocks}(\pi_{g(h),i}, b_{h,i})$
    $\mathcal{S}_{h,i} \leftarrow \textsc{ResidualReplace}(\mathcal{S}^0_{h,i}, h, i)$
  **end for**
**end for**
Add mandatory anchors and handle dense terminal rows
**return** $\{\mathcal{S}_{h,i}\}$

**Algorithm 2** Shared Proxy Importance Estimation

**Require:** Query states $Q$, key states $K$; proxy groups $\{\mathcal{G}_g\}_{g=1}^{G}$; block size $B$; stride $s$
**Ensure:** Proxy scores $\{P_g\}$ and shared rankings $\{\pi_{g,i}\}$
**for** each proxy group $g$ **do**
  $\bar{Q}_g \leftarrow |\mathcal{G}_g|^{-1} \sum_{h \in \mathcal{G}_g} Q_h$
  $\bar{K}_g \leftarrow \textsc{GroupKeys}(K, \mathcal{G}_g)$
  $\widetilde{Q}_g \leftarrow \textsc{StrideSample}(\bar{Q}_g, s)$
  $\widetilde{K}_g \leftarrow \textsc{StrideSample}(\bar{K}_g, s)$
  $\bar{A}_g \leftarrow \textsc{CausalAttention}(\widetilde{Q}_g, \widetilde{K}_g)$
  **for** each query block $i$ **do**
    $P_g(i, \cdot) \leftarrow \textsc{BlockMaxPool}(\bar{A}_g, i, B)$
    $\pi_{g,i} \leftarrow \textsc{DescendingSort}(P_g(i, \cdot))$
  **end for**
**end for**
**return** $\{P_g, \pi_{g,i}\}$

**Algorithm 3** Query-Guided Head-Specific Importance Estimation

**Require:** Head queries $Q_h$, corresponding keys $K_{g(h)}$; block size $B$; number of blocks $N$; coverage threshold $\gamma$; budget range $[b_{\min}, b_{\max}]$
**Ensure:** Probe set $\mathcal{P}_h$, block importance $M_h$, budget $b_h$

$\widehat{Q}_h \leftarrow Q_h[-B:]$, $\bar{q}_h \leftarrow \textsc{Mean}(\widehat{Q}_h)$
**for** each $q_{h,t} \in \widehat{Q}_h$ **do**
    $c_{h,t} \leftarrow \textsc{CosineSimilarity}(q_{h,t}, \bar{q}_h)$
**end for**
$(K_{\text{out}}, K_{\text{rep}}) \leftarrow \textsc{ScaleAwareAllocation}(N)$
$\mathcal{P}_h \leftarrow \textsc{BottomK}(c_h, K_{\text{out}}) \cup \textsc{TopK}(c_h, K_{\text{rep}})$
$M_h \leftarrow \textsc{ProbeBlockImportance}(\mathcal{P}_h, K_{g(h)}, B)$
$M_h^{\downarrow} \leftarrow \textsc{DescendingSort}(M_h)$
$k_h \leftarrow \min\left\{k : \frac{\sum_{\ell=1}^{k} M_h^{\downarrow}(\ell)}{\sum_{j=1}^{N} M_h(j)} > \gamma\right\}$
$b_h \leftarrow \text{clip}(k_h + 1, b_{\min}, b_{\max})$
**return** $\mathcal{P}_h, M_h, b_h$

**Algorithm 4** Gated Residual Block Replacement

**Require:** Initial selection $\mathcal{S}^0_{h,i}$; shared ranking $\pi_{g(h),i}$; head importance $M_h$; proxy scores $P_{g(h)}(i,\cdot)$; residual window $W$; thresholds $\tau_h, \tau_p$; protected set $\mathcal{A}_i$
**Ensure:** Refined selection $\mathcal{S}_{h,i}$

$\mathcal{S}_{h,i} \leftarrow \mathcal{S}^0_{h,i}$
$\mathcal{U}_{h,i} \leftarrow \textsc{NextUnselected}(\pi_{g(h),i}, \mathcal{S}^0_{h,i}, W) \setminus \mathcal{A}_i$
**if** $\mathcal{U}_{h,i} = \emptyset$ **then**
    **return** $\mathcal{S}_{h,i}$
**end if**
$j_{\text{out}} \leftarrow \arg\max_{j \in \mathcal{S}^0_{h,i} \setminus \mathcal{A}_i} \text{rank}_{\pi_{g(h),i}}(j)$
$j_{\text{in}} \leftarrow \arg\max_{j \in \mathcal{U}_{h,i}} M_h(j)$
$R_h \leftarrow \frac{M_h(j_{\text{in}})+\epsilon}{M_h(j_{\text{out}})+\epsilon}$, $R_p \leftarrow \frac{P_{g(h)}(i,j_{\text{in}})+\epsilon}{P_{g(h)}(i,j_{\text{out}})+\epsilon}$
**if** $R_h \geq \tau_h$ **and** $R_p \geq \tau_p$ **then**
    $\mathcal{S}_{h,i} \leftarrow (\mathcal{S}^0_{h,i} \setminus \{j_{\text{out}}\}) \cup \{j_{\text{in}}\}$
**end if**
**return** $\mathcal{S}_{h,i}$

# D Implementation Details

Unless otherwise specified, all main results use the `length_adaptive_v1` configuration summarized below. ProbeRefine uses a shared set of hyperparameters across model families, while proxy grouping and minimum budgets are adjusted according to the model architecture and context scale. The confidence gate is disabled in the main evaluation; the progressive gating variant is considered separately as an efficiency optimization.

Table 7: Shared configuration of ProbeRefine used in the main experiments.

| Configuration | Setting |
|---|---|
| Block size $B$ | 128 tokens |
| Coverage threshold $\gamma$ | 0.95 |
| Proxy stride $s$ | 4 |
| Context boundary | 512 blocks ($\approx$64K tokens) |
| Probes ($\leq$64K) | Outlier-Q16 |
| Probes ($>$64K) | Bimodal-Q32 (24:8) |
| Candidate ratio / maximum | 2.0 / 128 blocks |
| Residual window $W$ | 16 blocks |
| Residual refinement | Fixed one-for-one replacement |
| Protected blocks | Sink and recent blocks |
| Dense recent blocks | 2 |
| Refinement mass | Shared with budget mass |
| Confidence gate | Disabled |
| Attention backend | FlexPrefill block-sparse |

Table 8: Model-specific configurations of ProbeRefine.

| Model | Group ($\leq$64K) | Group ($>$64K) | Min. Budget |
|---|---|---|---|
| Qwen2.5-7B | GQA ($G = 4$) | GQA ($G = 4$) | 2048 / 2048 |
| Llama3.1-8B | All ($G = 1$) | GQA ($G = 8$) | 0 / 2048 |
| Yi-9B | All ($G = 1$) | GQA | 0 / 2048 |

Minimum budgets are reported as $\leq$64K / $>$64K in tokens. Under GQA grouping, $G$ follows the number of KV heads. Yi-9B-200K adopts the Llama profile without Yi-specific tuning.

**Length-dependent proxy grouping.** Proxy grouping controls the trade-off between shared estimation cost and selection granularity. We use model- and length-dependent grouping: Llama3.1 and Yi share a single proxy group up to 64K tokens and switch to their native GQA grouping at longer contexts, whereas Qwen2.5 uses native GQA grouping at all evaluated context lengths. Under GQA grouping, the number of proxy groups follows the number of KV heads. Yi-9B-200K directly adopts the Llama profile without Yi-specific tuning.

## E Efficiency Analysis

We further evaluate the runtime efficiency of ProbeRefine on an NVIDIA A800-SXM4-80GB GPU. We report the end-to-end attention-pipeline speedup relative to FlashAttention and further decompose the sparse-attention pipeline into selection, layout construction, and sparse execution.

**End-to-end speedup.** As shown in Fig. 4, the additional selection operations of ProbeRefine are not amortized at short context lengths, resulting in speedups below $1\times$ from 4K to 16K. As the context length increases, the benefit of sparse attention becomes more pronounced. ProbeRefine reaches $1.7\times$, $2.3\times$, and $2.5\times$ speedup over FlashAttention at 32K, 64K, and 128K, respectively. ProxyAttn reaches $1.4\times$, $2.7\times$, and $3.6\times$ at the same lengths. Thus, ProbeRefine retains substantial long-context acceleration over dense attention, while its head-specific probing and residual refinement introduce additional selection cost relative to the simpler proxy-based selection pipeline.

**Stage-wise latency.** Figure 5 further identifies where the additional runtime cost arises. The main overhead of ProbeRefine is concentrated in the selection stage, which includes head-specific query probing, importance estimation, and residual refinement. This overhead becomes particularly visible at 128K, consistent with the additional head-specific computation introduced by ProbeRefine.

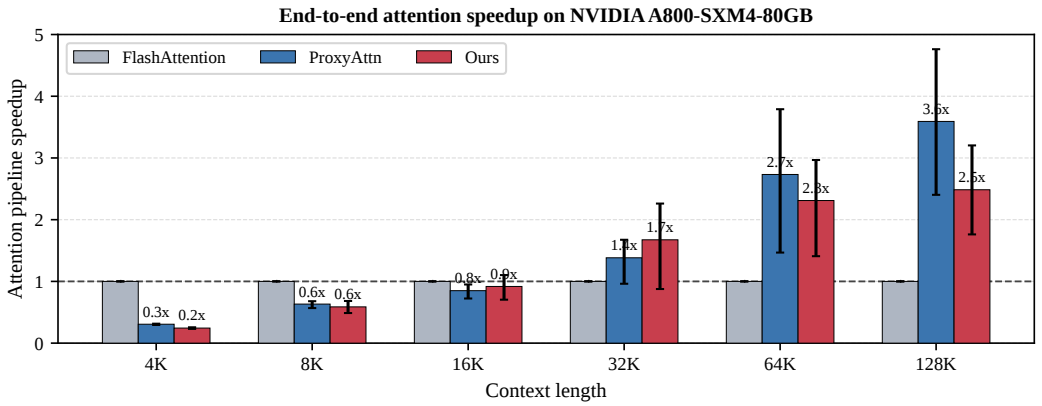


Figure 4: End-to-end attention-pipeline speedup on NVIDIA A800-SXM4-80GB across different context lengths. FlashAttention is normalized to $1\times$.

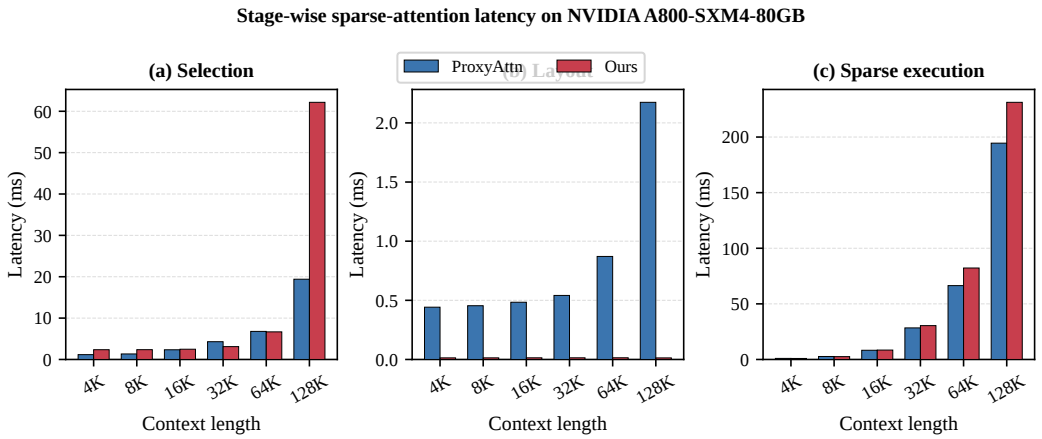


Figure 5: Stage-wise sparse-attention latency of ProxyAttn and ProbeRefine on NVIDIA A800-SXM4-80GB. The pipeline is decomposed into block selection, layout construction, and sparse attention execution.

In contrast, layout construction contributes only a small fraction of the overall latency in our implementation. At longer contexts, sparse execution accounts for a substantial portion of the pipeline cost, and the reduction in sparse attention computation is sufficient to offset the additional selection overhead relative to dense FlashAttention.

Overall, these results illustrate the efficiency–quality trade-off of ProbeRefine. Compared with ProxyAttn, ProbeRefine spends additional computation on head-specific block selection rather than relying solely on a shared proxy ranking. This additional selection cost limits its speedup relative to ProxyAttn at the longest contexts, but ProbeRefine still achieves up to $2.5\times$ end-to-end attention-pipeline speedup over FlashAttention while providing the accuracy improvements reported in the main experiments.