SAS: Simple Attention Sparsification via End-to-End Optimization of Context Ranking

Zhiwei Li[1,2*], **Lei Zhu**[1*†], **Hao Gu**[1,3], **Xiang Hu**[1], **Yan Wang**[1],
Haitao Mi[1], **Sirui Han**[3], **Leowei Liang**[1], **Zhijiang Guo**[2,3]
[1]Tencent HY LLM Frontier
[2]Hong Kong University of Science and Technology (Guangzhou)
[3]Hong Kong University of Science and Technology

https://github.com/Tencent-Hunyuan/Simple-Attention-Sparsification

Abstract

Post-training attention sparsification reduces the quadratic cumulative attention cost of pretrained Transformers by selecting a small set of context units (tokens or blocks) for each query. Existing trainable methods usually rely on a lightweight selector to score context units followed by a hard Top-K selection, which blocks gradients from the language modeling loss. As a result, these methods commonly resort to distilling layer-wise dense attention distributions. While this approach encourages the selector to rank context units according to dense attention weights in the original model, such a ranking is not directly aligned with their impact on the model's final predictions under a fixed attention budget (*i.e.*, the number of attended context units per query), which can waste the limited attention budget on less useful units. To address this ranking misalignment, we propose **S**imple **A**ttention **S**parsification (SAS), a gated sparse attention mechanism that optimizes context ranking *end-to-end* with the language modeling loss. The key idea is to inject the selector's continuous scores into the attention logits during training, allowing the language modeling loss to update the selector through standard back-propagation. We identify several choices that are crucial to make this simple design work well in practice: placing the gate inside the attention $\mathrm{softmax}$ in log form, using normalized $\mathrm{softmax}$ gates to calibrate historical context against the always-retained current block, and preserving continuous selector scores so the model learns relative priorities rather than only hard selections. To support long-sequence training, we implement a memory-efficient Triton kernel that integrates our method into FlashAttention-style computation. Across reasoning, long-context understanding, and agentic tasks, SAS consistently outperforms trainable sparse attention baselines across attention budgets, with especially large gains under tight budgets, demonstrating more effective context ranking for downstream tasks.

1 Introduction

Long-context inference has become a critical efficiency bottleneck for LLMs, as autoregressive generation requires dense attention over all preceding context tokens when generating each new token. This makes the cumulative attention cost grow quadratically with context length. Since many high-performing LLMs are already deployed with dense attention, practice favors inference-efficient methods that avoid architectural changes or retraining. This motivates post-training attention sparsification[1] [1–4], which adapts dense models into sparse-attention models after pretraining.

*Equal Contribution.
†Corresponding Author.
[1]We use "post-training" relative to dense pretraining: it includes methods applied after a dense model has been pretrained, such as mid-training adaptation, post-training fine-tuning, and training-free inference-time sparsification.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

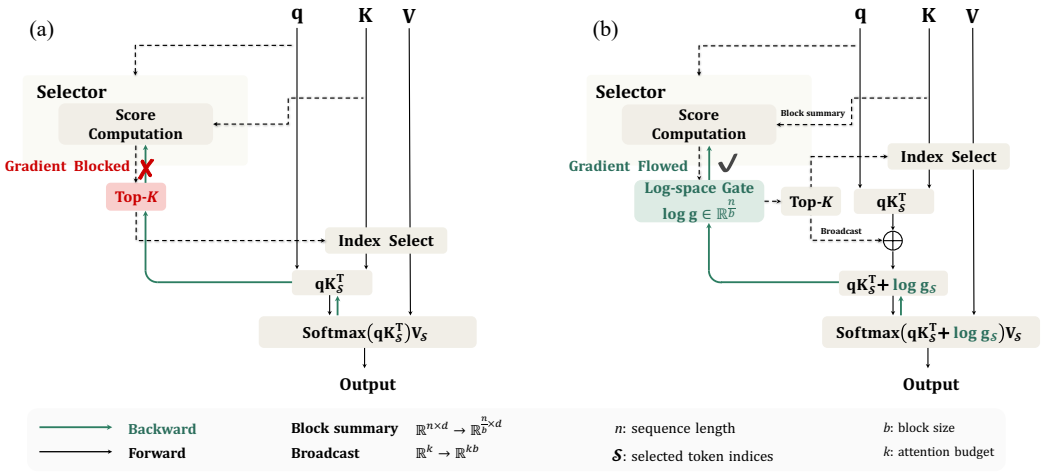


Figure 1: (a) Gradient Blockage in Discrete Selection. Sparse attention with **discrete Top-K selection** prevents gradients propagation from language modeling loss to selector. Existing approaches bypass this issue using *auxiliary distillation* [5, 4] or *heuristics* [6, 7]. (b) Differentiable Continuous Gating. SAS keeps discrete Top-K selection but attaches a continuous soft gate to each selected block, enabling gradient flow from the language modeling loss to the selector.

In post-training attention sparsification for LLMs, the central question is how to select the most useful context units (tokens or blocks) for each query under a limited attention budget, where a unit can be an individual token or a coarse-grained block depending on the sparsification scheme. Many training-free methods address this problem with heuristic or empirical rules [8, 1, 9–11]. While these training-free methods are attractive for their simplicity, their hand-crafted selection rules cannot adapt the selection strategy to the model's own prediction behavior. More recently, trainable sparsification has emerged as a promising direction by introducing learnable selectors [3, 4, 2]. A selector is a lightweight module that scores context units for each query, after which a hard Top-K step chooses the attended units, enabling a higher performance ceiling than hand-crafted selection rules. Because hard Top-K selection is non-differentiable, these methods commonly train selectors through layer-wise distillation of dense attention distributions: the selector in each layer is supervised to match the attention mass produced by the original dense model. However, this surrogate supervision can induce a ranking misalignment: it teaches the selector to rank context units according to where the original dense model attends, rather than by their impact on the model's final predictions under a limited attention budget. This distillation strategy has two limitations. First, per-layer targets can overlook cross-layer complementarity in context usage. Second, attention matching constrains only attention weights, ignoring how attended *values* (the $\mathbf{V}$ matrix in Figure 1) affect the final prediction.

In this work, we explore *end-to-end* optimization of context selection with the language modeling loss. Our key idea is to make the selector's scores part of the differentiable attention computation during training, so that the standard language modeling loss can update the selector through backpropagation. We instantiate this idea in the open-source block sparse setting of SeerAttention-R [4], where layer-wise selectors score context blocks for each query. Instead of using layer-wise attention distillation as in SeerAttention-R, we optimize the selectors end-to-end via a simple design: during training, blockwise scores from selector are interpreted as log-space gates, and added to the $\mathbf{qK}^\top$ attention logits, allowing gradients to flow from the attention computation to the selectors (Figure 1b).

The effectiveness of this simple design, however, depends on four key choices in how the selector is trained from the language modeling loss. *(1)* The gate should be added inside the attention $\mathrm{softmax}$ in log form, so selector scores directly modulate attention mass rather than merely rescaling attention outputs. *(2)* The gate activation should normalize scores across context blocks (*e.g.*, with a $\mathrm{softmax}$), so the gates encode calibrated relative importance rather than unnormalized scores. *(3)* The selected gates should stay continuous rather than being collapsed into binary Top-K indicators, because the model must learn not only which context blocks are selected but also their relative priorities. *(4)* Training should update only the selected blocks rather than all context blocks, since it is much cheaper

and still reaches comparable final performance. Together, the first three choices make end-to-end gradients from the language modeling loss informative for learning which context blocks actually support the model's prediction, while the last keeps training efficient.

Besides the design choices above that make the end-to-end optimization effective, we also implement specialized Triton [12] kernels that make long-context training practical. Since a naive implementation would materialize the full attention matrix before adding the log-space gates, which is prohibitive for long-context training, our kernel fuses the gate addition into the tile-level $\mathbf{q}\mathbf{K}^\top$ computation in FlashAttention [13]. As a result, attention sparsification with SAS is simplified to replacing the attention kernel with our gated-attention kernel and optimizing the standard language modeling loss, without auxiliary distillation.

Despite its simplicity, extensive experiments demonstrate that SAS consistently outperforms existing post-training attention sparsification baselines across reasoning, long-context understanding, and agentic tasks. We further provide initial evidence that the same end-to-end optimization applies to continued pretraining, where the backbone and selector are trained jointly. The improvements are most pronounced under low attention budgets, where accurate context ranking is most critical: on reasoning tasks at a 1024 token budget, SAS improves over SeerAttention-R [4] by 6.0–7.7 points on MATH500 and 10.6–15.5 points on GPQA-Diamond across Qwen3-4B, 8B, and 14B. The gains extend beyond reasoning: on long-context understanding (LongBench) SAS leads SeerAttention-R at every budget, with the largest margins on the longest inputs (*e.g.*, +3.2 on the 8K+ bucket of Qwen3-4B at budget 2048), and on agentic tasks it improves BFCL by up to +3.5 and remains ahead on VitaBench, nearly recovering full attention at budget 4096. Our contributions are:

- We propose SAS, an end-to-end post-training attention sparsification paradigm that trains context selectors with the language modeling loss, removing the need for teacher attention or auxiliary distillation. In controlled comparisons with matched backbones, selector architectures, and training data, SAS learns more effective sparse routing than layer-wise attention distillation.
- We identify the design choices that make simple gated relaxation effective and practical, including log-space gate injection, normalized gate activation, preservation of fine-grained score differences, an efficient sparse training scope, and a memory-efficient kernel for long-context training.
- SAS consistently improves performance across reasoning (MATH500, GPQA-Diamond, AIME24, AIME25), long-context understanding (LongBench), and agentic tasks (BFCL, VitaBench), with gains of over 10% under low attention budgets.

2 Related Work

Inference time attention sparsification / KV-cache selection. A practical way to reduce long-context attention cost is to sparsify attention at inference time by selecting a subset of context units without updating the backbone LLM [8, 14, 1, 11, 15–19]. Closely related are KV-cache methods that dynamically select relevant cached tokens without irreversible eviction, which is more suitable for long reasoning generation where future token importance is hard to predict [1, 20, 21, 16, 22–24]. These methods are attractive because they can be applied to existing dense LLMs with little or no additional training. However, their sparse locations are usually determined by heuristic rules, post-hoc statistics, or external retrieval/indexing procedures, leaving the selection rule largely fixed rather than learned from the model's prediction objective. In contrast, our method keeps the post-training sparsification setting but learns a lightweight block selector directly from the language modeling loss.

Trainable sparse attention. Another line of work makes sparse attention patterns trainable. Early sparse Transformers impose fixed layouts, such as local windows, global tokens, strided connections, or block-wise patterns [25–27], but typically require training or adapting models around prescribed sparse patterns. Recent methods use learnable routing, gating, or dense-sparse switching modules [28, 29, 6, 30, 3, 4, 7, 2]. However, hard Top-K selection prevents language-modeling gradients from directly updating the selector, so existing methods often rely on dense-attention distillation [3, 4, 2] or proxy selection signals [29, 6]. In contrast, we relax hard block selection into continuous log-space gates inside attention, enabling end-to-end selector optimization with the language modeling loss.

3 Preliminaries

This section reviews attention in autoregressive language models and block sparse attention, which reduces long-context inference cost by restricting each query to a subset of context blocks.

Standard attention. Let $\mathbf{q} \in \mathbb{R}^d$ denote a query vector of the current token, and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ denote the key and value matrices of the n previous tokens, which are referred to as *context tokens*. Standard attention [31] computes the output $\mathbf{o} \in \mathbb{R}^d$ as

$$\mathbf{o} = \mathrm{softmax}(\mathbf{q}\mathbf{K}^\top)\mathbf{V}. \tag{1}$$

Here we omit the scaling factor for notational simplicity. During autoregressive decoding, each new query attends to all previous keys and values. Since the number of previous tokens grows linearly with the decoding step, the cumulative attention cost over a sequence of length n grows as $\mathcal{O}(n^2)$.

Block sparse attention. To mitigate this quadratic cost, block sparse attention [32, 33] restricts the query $\mathbf{q}$ to attend only to a selected subset of context blocks. Let $\mathcal{S} \subseteq \{1, \ldots, n\}$ denote the selected token-index set. The attention output is formulated as

$$\mathbf{o} = \mathrm{softmax}\left(\mathbf{q}\mathbf{K}_{\mathcal{S}}^\top\right)\mathbf{V}_{\mathcal{S}}. \tag{2}$$

For a single query, this reduces the computation cost from $\mathcal{O}(n)$ to $\mathcal{O}(|\mathcal{S}|)$. Over a sequence of length n, the cumulative cost is therefore reduced to $\mathcal{O}(n|\mathcal{S}|)$. To construct $\mathcal{S}$, the n context positions are partitioned into C contiguous index blocks $\{B_1, \ldots, B_C\}$, where each B_m denotes a set of token indices and contains b tokens (i.e., $C = n/b$). A lightweight selector $\mathcal{R}_\theta(\cdot)$ computes relevance scores $\mathbf{s} \in \mathbb{R}^C$ over these blocks. The selector chooses the K most important blocks from all C context blocks, and the query only attends to the union of these selected blocks. Formally, the selected block indices and the corresponding token-index set are

$$\mathcal{I} = \text{Top-}K(\mathbf{s}, K), \qquad \mathcal{S} = \bigcup_{m \in \mathcal{I}} B_m. \tag{3}$$

Under plain Top-K, selected index set is piecewise constant with respect to selector scores. Therefore, language modeling loss provides no useful gradient through selection path to update selector, as illustrated in Figure 1a. To make selector end-to-end trainable, core problem is how to make selector scores affect attention differentiably, so that language modeling loss can train the selector.

4 Method

We present SAS, a **S**imple **A**ttention **S**parsification method that trains a lightweight selector to rank context blocks from the language modeling loss. In this section, we first reformulate sparse block selection as context ranking. We then analyze how to make such rankings learnable from the language modeling loss, focusing on gate position, gate activation, ranking preservation, and training scope. Finally, we introduce a specialized training kernel for the practical implementation of SAS.

4.1 From Sparse Selection to Context Ranking

The starting point is that Top-K selection is determined by the ordering of selector scores over context blocks. Before producing a discrete selected set, the selector first defines a ranking of candidate blocks. A good selector should assign higher ranks to blocks that are more useful for prediction. We therefore view attention sparsification as *context ranking*: during training, the selector learns a continuous ordering over blocks, and this ordering provides the basis for sparse block selection.

Differentiable ranking. To make block ranking learnable, the selector scores must affect the attention computation during training. Following the block partition in Section 3, we single out the always-retained current block as B_0 and treat the remaining C context blocks $\mathcal{H} = \{B_1, \ldots, B_C\}$ as the candidate historical blocks that the selector ranks. The selector produces scores $\mathbf{s} \in \mathbb{R}^C$ only for $\mathcal{H}$, which we convert into positive gates

$$\mathbf{g} = \phi(\mathbf{s}) \in \mathbb{R}_+^C, \qquad g_0 = 1, \tag{4}$$

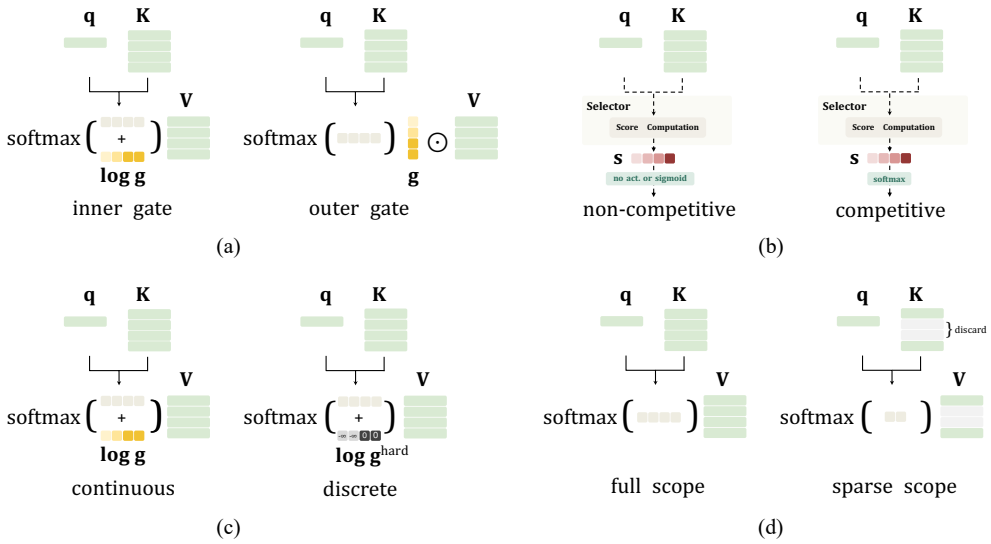


Figure 2: Components of learnable context ranking. (a) Gate position, whether the gates are injected into the softmax or applied outside the softmax. (b) Gate activation, whether historical-block scores are normalized jointly with softmax, activated independently with sigmoid, or injected as raw logits; the current block always has a unit gate. (c) Ranking preservation, whether continuous scores are preserved or collapsed into binary masks. (d) Training scope, whether training receives the gradient from all context blocks or only the selected blocks.

where $\phi(\cdot)$ is an activation function and $g_0 = 1$ leaves the current block unbiased. This transformation allows historical-block scores to modulate attention rather than determine a discrete Top-K set.

For each historical block $B_m \in \mathcal{H}$, the gate g_m is broadcast to all tokens in that block, while tokens in B_0 retain the unit gate. By making the attention computation depend on these broadcast gates, gradients from the language modeling loss can flow back to $\mathbf{g}$ and then to the selector. The key design question is how to *integrate gates into the attention computation so that gradients provide a reliable signal for block ranking*.

Components of learnable ranking. Merely allowing language modeling gradients to reach the selector does not guarantee effective block ranking: different gated formulations shape the resulting ranking signals differently. As illustrated in Figure 2, we investigate four design elements: gate position, gate activation, ranking information preservation, and training scope.

- *Gate position*: whether the gates are applied into the softmax, i.e., $\mathbf{o}_{\text{inner}} = \text{softmax}(\mathbf{q}\mathbf{K}^\top + \log \mathbf{g})\mathbf{V}$, or applied outside the softmax, i.e., $\mathbf{o}_{\text{outer}} = \text{softmax}(\mathbf{q}\mathbf{K}^\top)(\mathbf{g} \odot \mathbf{V})$. For the inner placement, $\log(\cdot)$ ensures that the gates act as multiplicative weights on the attention probabilities after the softmax.
- *Gate activation*: how block scores are transformed before being injected into attention. We compare normalized gates $\mathbf{g} = \text{softmax}(\mathbf{s})$, independent gates $\mathbf{g}_\sigma = \text{sigmoid}(\mathbf{s})$, and unnormalized logit injection, which adds $\mathbf{s}$ directly to attention logits. In all variants, current block remains unbiased with $g_0 = 1$.
- *Ranking preservation*: whether training uses a soft gate $\mathbf{g} = \phi(\mathbf{s})$, or a hard Top-$K$ gate using STE, i.e., $\hat{\mathbf{g}} = \text{stopgrad}(\mathbf{g}^{hard} - \mathbf{g}) + \mathbf{g}$ with $g_m^{hard} = \mathbb{I}[m \in \text{Top-}K(\mathbf{s}, K)]$.
- *Training scope*: whether training uses the full scope in Equation 1, the sparse scope in Equation 2, or a stochastic approximation to full scope by adding noise to the scores before activation.

4.2 Learning Reliable Block Ranking

We next use controlled ablations to evaluate how the four design elements introduced above affect the effectiveness and stability of block-ranking optimization.

Table 1: Ablation of four gating design choices on GPQA-Diamond using Qwen3-4B with a 2048-token budget. We report avg@16 accuracy (%), with average generation length; standard deviations (2.0–2.8) are omitted. $\mathbf{g}_\sigma$ denotes sigmoid-activated gates, and full* simulates full-scope training by adding noise before Top-K. Formulations show only historical-context terms; the current block always uses a unit gate.

Setting	Gate Pos.	Gate Act.	Rank Pres.	Train Sco.	10 step	100 step	1000 step	1 epoch
Baseline								
(1) $\mathrm{softmax}(\mathbf{qK}^\top)\mathbf{V}$	–	–	–	–	–	–	–	56.1 8547
I. Gate Position (inner softmax *vs. outer* softmax*)*								
(2) $\mathrm{softmax}(\mathbf{qK}^\top + \log \mathbf{g})\mathbf{V}$	inner	softmax	✓	full	30.8 25277	**52.2** 8696	**53.7** 6977	54.4 7109
(3) $\mathrm{softmax}(\mathbf{qK}^\top)(\mathbf{g} \odot \mathbf{V})$	outer	softmax	✓	full	28.3 25594	38.9 16969	43.2 12801	41.6 13807
*II. Gate Activation (*softmax, sigmoid, *or none)*								
(4) $\mathrm{softmax}(\mathbf{qK}^\top + \log \mathbf{g})\mathbf{V}$	inner	softmax	✓	full	30.8 25277	**52.2** 8696	**53.7** 6977	54.4 7109
(5) $\mathrm{softmax}(\mathbf{qK}^\top + \log \mathbf{g}_\sigma)\mathbf{V}$	inner	sigmoid	✓	full	19.6 27717	20.4 27997	17.6 27989	17.0 28444
(6) $\mathrm{softmax}(\mathbf{qK}^\top + \mathbf{s})\mathbf{V}$	inner	–	✓	full	23.4 27540	19.9 27418	18.9 26703	18.8 26401
III. Ranking Preservation (continuous vs. discrete)								
(7) $\mathrm{softmax}(\mathbf{qK}^\top + \log \mathbf{g})\mathbf{V}$	inner	softmax	✓	full	30.8 25277	**52.2** 8696	**53.7** 6977	54.4 7109
(8) $\mathrm{softmax}(\mathbf{qK}^\top + \log \hat{\mathbf{g}})\mathbf{V}$	inner	softmax	✗	full	**46.5** 9821	42.4 8438	49.6 7135	46.0 7289
IV. Training Scope (full scope vs. sparse scope)								
(9) $\mathrm{softmax}(\mathbf{qK}^\top + \log \mathbf{g})\mathbf{V}$	inner	softmax	✓	full	30.8 25277	**52.2** 8696	**53.7** 6977	54.4 7109
(10) $\mathrm{softmax}(\mathbf{qK}_{\mathcal{S}}^\top + \log \mathbf{g}_{\mathcal{S}})\mathbf{V}_{\mathcal{S}}$	inner	softmax	✓	sparse	24.8 27504	51.0 8087	53.6 7401	**54.8** 7410
(11) $\mathrm{softmax}(\mathbf{qK}_{\tilde{\mathcal{S}}}^\top + \log \mathbf{g}_{\tilde{\mathcal{S}}})\mathbf{V}_{\tilde{\mathcal{S}}}$	inner	softmax	✓	full*	26.3 27350	51.2 7732	52.6 7424	52.2 7552

4.2.1 Experimental setup

We conduct controlled experiments using the AttnGate selector from SeerAttention-R [4]. For our experiments, we set the block size to 64 and Top-K of 32. The experiments are performed on Qwen3-4B [34] trained on 93.7K examples from OpenR1-Math-220k [35]. Both the training and generation lengths are set to 32,768 tokens. The LLM backbone is frozen during training. We evaluate GPQA-Diamond [36] with inference consistently performed under the block sparse attention formulation in Equation 2, and report the accuracy averaged over 16 runs and generation length.

4.2.2 Ablation analysis

Overall observation. Table 1 and Figure 8 compare four design dimensions for learning reliable block rankings. First, inner $\mathrm{softmax}$ gating performs better than outer gating, showing that gates should participate in attention normalization instead of only rescaling *values* ($\mathbf{V}$ matrix). Second, normalized gating is important: $\mathrm{softmax}$ consistently outperforms independent $\mathrm{sigmoid}$ gates and unnormalized raw-logit injection because it calibrates historical context against the unit-gated current block. Third, preserving ranking information through soft gating improves training stability, while achieving better final performance. Unlike the first three, training scope mainly affects efficiency: sparse scope training converges more slowly at first but gradually approaches full scope, reaching comparable final performance at lower cost.

Gate position. The key distinction is whether gates participate in attention *normalization*. Outside the $\mathrm{softmax}$, the attention probabilities are already fixed, and the gate only rescales the *value* contribution of each block. Inside the $\mathrm{softmax}$ as log-space biases, they directly change how attention mass is allocated across blocks. Let p_i denote the attention probability without gating, $\tilde{p}_i$ the attention probability with inner gating. This difference is reflected in the gate gradients:

$$dg_m^{inner} = \sum_{i \in B_m} \frac{\tilde{p}_i}{g_m}\, d\mathbf{o}^\top(\mathbf{v}_i - \mathbf{o}), \qquad dg_m^{outer} = \sum_{i \in B_m} p_i\, d\mathbf{o}^\top \mathbf{v}_i. \tag{5}$$

Outer gating uses fixed attention probabilities, while inner gating provides a relative signal through $\mathbf{v}_i - \mathbf{o}$ for reallocating attention mass, enabling gates to learn relative importance among blocks.

Activation function. The activation function determines how the selector calibrates historical context against the always-retained current block. For historical blocks, $\mathbf{g} = \mathrm{softmax}(\mathbf{s})$ gives

$$\log g_m = s_m - \mathrm{LSE}(\mathbf{s}), \qquad B_m \in \mathcal{H}, \tag{6}$$

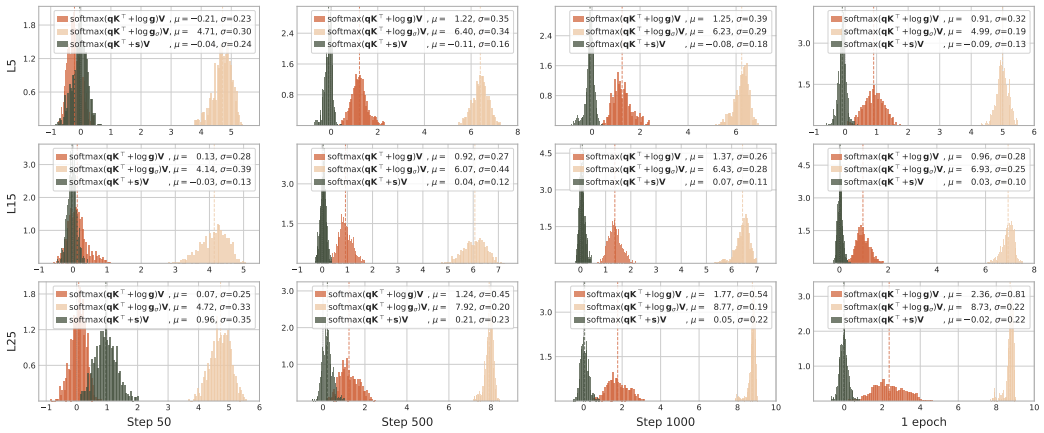


Figure 3: Evolution of selector logits $\mathbf{s}$ under softmax, sigmoid gating and raw-logit injection in different layers. Sigmoid gating saturates historical gates toward one, while raw injection collapses logits toward zero with reduced variance. Thus, both transformations approach effectively ungated attention and fail to provide a discriminative signal for learning block rankings. These observations highlight the importance of the competitive nature of softmax normalization in avoiding such trivial solutions and preserving discriminative block selection.

while current block has $g_0 = 1$ and hence zero additive bias. Although $-\mathrm{LSE}(\mathbf{s})$ is shared by all historical blocks, it does not cancel in the subsequent attention $\mathrm{softmax}$ because it is not applied to current block. It therefore calibrates the aggregate attention mass assigned to historical context relative to current block. This normalization also makes the gates invariant to a global shift $\mathbf{s} \mapsto \mathbf{s} + c$. By contrast, directly injecting unnormalized logits $\mathbf{s}$ changes the historical-to-current attention balance under the same shift. Empirically, Figure 3 shows that $\mathrm{sigmoid}$ gates gradually saturate toward 1, whereas unnormalized logits collapse toward 0 with reduced variance. Both behaviors diminish the distinction between historical blocks and unit-gated block, weakening the learned gating signal.

Ranking preservation. The key difference is whether the forward pass preserves the continuous ranking or collapses it into a binary Top-K mask. Let $z_i = \mathbf{q}\mathbf{k}_i^\top$ denote the attention logit of token i, where $\mathbf{k}_i$ is its key vector, and let $b(i)$ index the block that token i belongs to. Soft gating keeps every block under a single normalizer that sums over all tokens, so each attention weight stays bounded. Hard gating instead relies on STE for a differentiable backward: its forward normalizer sums only over the selected set $\mathcal{S}$, and a dropped block $m \notin \mathcal{I}$ is ranked by the attention mass its tokens would receive under this restricted normalizer, from which its own tokens are absent:

$$\tilde{p}_i^{soft} = \frac{g_{b(i)} \exp(z_i)}{\sum_j g_{b(j)} \exp(z_j)} \le 1, \qquad \tilde{p}_i^{hard} = \frac{\exp(z_i)}{\sum_{j \in \mathcal{S}} \exp(z_j)} \le \exp\left(z_i - \max_{j \in \mathcal{S}} z_j\right). \tag{7}$$

The soft weight $\tilde{p}_i^{soft}$ is bounded, while hard weight $\tilde{p}_i^{hard}$ is unbounded and grows exponentially once a dropped block outscores the selected set. Since $dg_m \propto p_i$ (Equation 5), these unbounded weights are directly translated into large gradients. Early in training, the randomly initialized selector frequently leaves high-score blocks outside Top-K. Contributions from dropped blocks and heads then accumulate, yielding gradient norms orders of magnitude larger than those under soft gating.

Training scope. The key difference is whether non-Top-K blocks receive *independent gradient signals*. Under sparse scope, only the current Top-K blocks participate in attention. A block outside the selected set therefore has no direct gate gradient, even if its score may still change indirectly through the $\mathrm{softmax}$ normalization over selector scores. This indirect update does not evaluate the content of the unselected block itself. For $\phi(\cdot) = \mathrm{softmax}(\cdot)$, consider an unselected block B_m with $m \notin \mathcal{I}$. Gradients under sparse and full scope are

$$ds_m^{sparse} = -g_m \cdot \underbrace{\sum_{\ell \in \mathcal{I}} g_\ell dg_\ell}_{\text{Top-}K\text{ blocks}}, \qquad ds_m^{full} = -g_m \cdot \underbrace{\left(-dg_m + \sum_{\ell=1}^{C} g_\ell dg_\ell \right)}_{\text{own signal + full normalization}}. \tag{8}$$

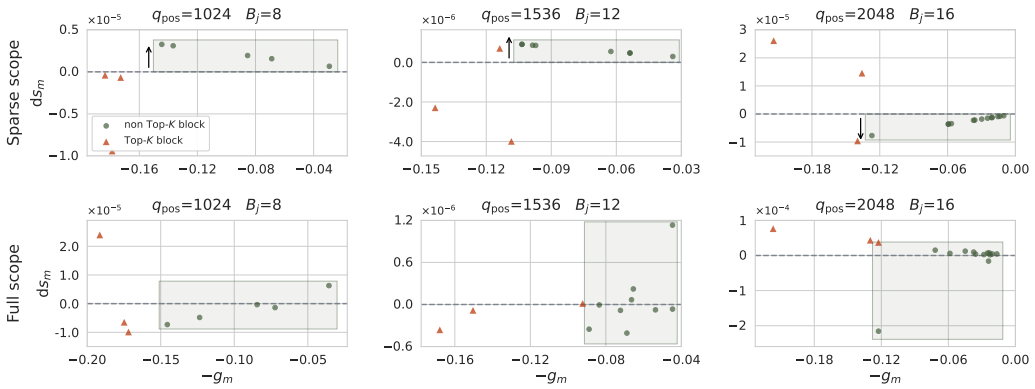


Figure 4: Gradient behavior under sparse and full training scopes with Top-K of 3. For non Top-K blocks, sparse scope training yields noisy and highly correlated gradients, with gradients of these unselected blocks simultaneously being either positive or negative across blocks, ignoring their own importance scores (x-axis). This provides a less informative optimization signal and leads to less efficient gate optimization than full scope training.

Algorithm 1 SAS Training Step

Require: $\mathbf{q}, \mathbf{K}, \mathbf{V}$, selector $\mathcal{R}_\theta$, budget K

$\mathbf{s} = \mathcal{R}_\theta\left(\mathbf{q}, \{\mathbf{K}_{B_m}\}_{m=1}^{C}\right)$

$\mathbf{g} = \mathrm{softmax}(\mathbf{s})$

$\mathcal{I} = \text{Top-}K(\mathbf{g}, K)$, $\mathcal{S} = B_0 \cup \bigcup_{m\in\mathcal{I}} B_m$

$\mathbf{o} = \mathrm{softmax}(\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top} + \log \mathbf{g}_{\mathcal{S}})\mathbf{V}_{\mathcal{S}}$

Update θ with $\nabla_\theta \mathcal{L}_{\mathrm{LM}}$

return $\mathcal{R}_\theta$

Algorithm 2 SAS Inference Step

Require: $\mathbf{q}, \mathbf{K}, \mathbf{V}$, selector $\mathcal{R}_\theta$, budget K

$\mathbf{s} = \mathcal{R}_\theta\left(\mathbf{q}, \{\mathbf{K}_{B_m}\}_{m=1}^{C}\right)$

$\mathcal{I} = \text{Top-}K(\mathbf{s}, K)$

$\mathcal{S} = B_0 \cup \bigcup_{m\in\mathcal{I}} B_m$

$\mathbf{o} = \mathrm{softmax}\left(\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top}\right)\mathbf{V}_{\mathcal{S}}$

Forward to obtain next token logits

return next token

As shown in Figure 4, sparse scope updates an unselected block only through selected blocks, while full scope gives it its own gradient dg_m, allowing its score to be adjusted based on its own content.

Empirically, as shown by settings (9) and (10) in Table 1, sparse scope starts worse than full scope but gradually catches up, reaching comparable final performance. This trend holds across model scales and budgets (Appendix F). Given its lower training cost, we therefore rely mainly on sparse scope in our subsequent experiments. We provide the whole gradient derivations in Appendix G.

4.3 Instantiating Simple Attention Sparsification

Based on the above analysis, SAS adopts four design choices: *1)* inner $\mathrm{softmax}$ gate injection, *2)* $\mathrm{softmax}$ gate activation, *3)* soft gate to keep ranking information, *4)* sparse scope during training. Given the query $\mathbf{q}$, the selector computes relevance scores for the historical blocks $\mathbf{s} = \mathcal{R}_\theta\left(\mathbf{q}, \{\mathbf{K}_{B_m}\}_{m=1}^{C}\right)$. The scores are converted into normalized gates $\mathbf{g} = \mathrm{softmax}(\mathbf{s})$, and the Top-$K$ historical blocks are selected as $\mathcal{I} = \text{Top-}K(\mathbf{g}, K)$. We define the attended token set as $\mathcal{S} = B_0 \cup \bigcup_{m\in\mathcal{I}} B_m$, including the always-retained current block B_0, and broadcast the selected block gates to the token-level gate $\mathbf{g}_{\mathcal{S}}$ aligned with the attention logits. The resulting attention computation used during training is given by

$$\mathbf{o}_{SAS} = \mathrm{softmax}\left(\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top} + \log \mathbf{g}_{\mathcal{S}}\right)\mathbf{V}_{\mathcal{S}}. \tag{9}$$

Thus, the selected historical blocks receive normalized log-gate biases, whereas the current block remains unbiased with a unit gate, which instantiates the differentiable continuous gating illustrated in Figure 1b. At inference time, the learned ranking is discretized into Top-K block indices. Algorithms 1 and 2 summarize the training and inference procedures, respectively.

Kernel Design. SAS training computes sparse scope attention, $\mathrm{softmax}(\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top} + \log \mathbf{g}_{\mathcal{S}})\mathbf{V}_{\mathcal{S}}$, where $\log \mathbf{g}_{\mathcal{S}}$ contains normalized log-gate biases for selected historical blocks and zero bias for the always-retained current block. The selected historical set is encoded by a per-query threshold on the gate: a block is active when its log gate reaches the threshold, which realizes the Top-K selection $\mathcal{I}$ without an explicit sort. FlashAttention [13] kernels neither support injecting such per-block gates inside the tiled $\mathrm{softmax}$ nor skipping blocks according to a learned gate. A naive implementation materializes the gated attention matrix, incurring prohibitive memory overhead.

We implement a FlashAttention-style Triton [12] kernel that fuses the block gate into the tile-level $\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top}$ computation. During the streaming scan over key–value tiles, the kernel adds the normalized log gate to the attention logits of each selected historical block, masks out non-selected historical blocks, leaves the always-retained current block unbiased, and performs the standard online $\mathrm{softmax}$ update. The backward pass accumulates block-level log-gate gradients by summing the attention-logit gradients within each selected historical block. This kernel keeps the sparse training scope while avoiding attention-matrix materialization and retaining FlashAttention-style memory efficiency.

5 Experiments

We primarily evaluate SAS in the *post-training sparsification* setting (Section 5.2), where the backbone LLM is frozen and only the selector is trained, allowing a controlled comparison against distillation-based methods. We further verify that the same end-to-end formulation extends beyond this setting to *continued pretraining* (Appendix B), where backbone and selector are trained jointly.

5.1 Experimental Setup

Baselines, models, and evaluation. We first compare our method with full attention and SeerAttention-R [4], which is a strong post-training sparse attention baseline that freezes the backbone LLM, uses a carefully designed AttnGate selector, and trains through dense attention distillation. We keep the same selector structure, but train block selection from the language modeling loss. On reasoning tasks, we also compare with Sliding Window Attention, StreamingLLM [37], and Quest [1]. We conduct experiments on Qwen3 [34] backbones (4B, 8B, 14B). We evaluate on three types of tasks: reasoning, including MATH500 [38], GPQA-Diamond [36], AIME24, and AIME25 [39]; long-context understanding, using LongBench [40]; and agentic tasks, using BFCL (Multi-Turn) [41] and VitaBench [42]. For reasoning tasks, we follow the same evaluation protocol as SeerAttention-R. For long-context understanding, we evaluate LongBench in the non-thinking mode. For agentic tasks, we use YaRN [43] to extend the context window to 65,536 during evaluation.

Training setup. In all settings we train only the AttnGate selector with the LLM backbone frozen, using AdamW [44] with learning rate 1e-3, cosine decay, and block size 64. The selector is trained on OpenR1-MATH-220K[2] for one epoch with global batch size 32 and a maximum sequence length of 32,768, and the same trained selector is used across reasoning, long-context, and agentic evaluations. Our implementation builds on VeOmni [45] with FSDP distributed training.

5.2 Post-Training Results

Reasoning task performance. As shown in Table 2, SAS outperforms all sparse baselines across nearly all budgets, backbones, and tasks. The training-free static-sparsity baselines (Sliding Window, StreamingLLM) and the query-aware Quest all degrade sharply at tight budgets: the fixed local (plus optional sink) patterns of Sliding Window and StreamingLLM discard reasoning-critical context, and Quest collapses entirely (e.g., 0 on AIME24/25 at budget 2048), whereas SAS stays strong. Against SeerAttention-R, which uses the identical AttnGate but is trained by attention distillation, SAS still wins clearly on harder tasks (e.g., +13.0 on AIME24 at budget 2048, Qwen3-4B); since the two differ only in the training signal, this directly shows the language modeling loss is a more effective supervision for sparse selection. At budget 4096, SAS matches or exceeds full attention in several cases (e.g., 71.72 vs. 71.25 on AIME24, Qwen3-4B), recovering full attention reasoning while attending to only a small fraction of the KV blocks.

[2] `https://huggingface.co/datasets/open-r1/OpenR1-Math-220k`

Table 2: Evaluation results on reasoning benchmarks. Quest* denotes results extracted from Figure 5 of SeerAttention-R using the WebPlotDigitizer tool, while SeerAttn-R# denotes our reproduced results. Standard deviations are shown in parentheses.

Budget	Method	MATH500			GPQA-Diamond			AIME24			AIME25		
		4B	8B	14B	4B	8B	14B	4B	8B	14B	4B	8B	14B
Full	Full Attn	93.93	94.43	95.22	56.19	60.54	65.25	71.25	74.48	78.91	66.41	67.86	70.21
1024	Sliding Window	20.18 (1.3)	-	-	1.52 (0.5)	-	-	-	-	-	-	-	-
	Quest*	10.38	31.65	54.45	4.05	8.76	18.15	-	-	-	-	-	-
	StreamingLLM	72.72 (1.8)	72.17 (1.8)	73.47 (1.8)	13.86 (1.7)	13.04 (1.8)	19.07 (2.2)	-	-	-	-	-	-
	SeerAttn-R	84.67	83.57	86.12	39.84	39.43	45.64	-	-	-	-	-	-
	SAS	**90.65** (1.1)	**91.27** (1.1)	**92.93** (1.0)	**50.41** (2.8)	**53.17** (2.9)	**61.14** (2.9)	-	-	-	-	-	-
2048	Sliding Window	50.72 (1.8)	-	-	8.93 (1.6)	-	-	5.47 (2.4)	-	-	2.19 (1.1)	-	-
	Quest	40.74	68.66	81.52	12.17	24.83	41.01	0	12.98	25.12	0	12.66	27.22
	StreamingLLM	83.65 (1.5)	83.40 (1.5)	84.97 (1.4)	27.18 (2.4)	27.46 (2.4)	34.06 (2.7)	29.69 (7.2)	25.89 (7.1)	28.85 (7.4)	15.94 (6.1)	15.94 (6.3)	15.68 (6.2)
	SeerAttn-R	91.85	91.67	93.02	49.94	54.41	61.68	55.83	58.23	63.65	45.16	43.30	48.70
	SAS	**93.47** (1.0)	**93.17** (1.0)	**93.54** (1.0)	**54.86** (2.8)	**58.74** (2.8)	**65.09** (2.8)	**68.85** (6.6)	**70.73** (6.7)	**77.76** (6.6)	**56.38** (7.3)	**58.41** (7.2)	**64.19** (7.1)
4096	Sliding Window	75.33 (1.6)	-	-	23.74 (2.6)	-	-	19.74 (5.7)	-	-	16.35 (5.7)	-	-
	Quest	71.59	86.68	90.76	23.21	43.18	58.02	12.41	43.99	46.67	12.41	32.14	42.13
	StreamingLLM	91.03 (1.1)	91.12 (1.1)	92.07 (1.1)	40.40 (2.8)	42.27 (2.8)	47.35 (2.9)	45.94 (7.9)	46.35 (8.0)	49.32 (8.1)	31.56 (7.3)	31.15 (7.4)	32.60 (7.5)
	SeerAttn-R	**94.10**	94.00	95.12	**55.40**	**60.48**	63.83	69.32	71.35	75.73	58.59	57.81	64.79
	SeerAttn-R#	93.00 (1.1)	**94.25** (0.9)	94.83 (0.8)	54.86 (2.8)	59.03 (2.9)	63.67 (2.8)	69.16 (6.9)	69.43 (6.9)	75.10 (6.8)	56.59 (7.6)	57.66 (7.4)	64.69 (7.0)
	SAS	93.67 (1.0)	94.23 (1.0)	**95.23** (0.8)	55.11 (2.9)	60.42 (2.8)	**65.03** (2.8)	**71.72** (6.8)	**73.49** (6.7)	**78.28** (6.4)	**59.97** (7.3)	**63.41** (7.2)	**67.29** (6.9)

Table 3: Evaluation results on LongBench, grouped by input length (0-4K, 4-8K, and 8K+ tokens).

Budget	Method	Qwen3-4B				Qwen3-8B				Qwen3-14B			
		0-4K	4-8K	8K+	Avg.	0-4K	4-8K	8K+	Avg.	0-4K	4-8K	8K+	Avg.
Full	Full Attn	53.9	52.4	50.5	52.2	57.4	54.1	51.7	54.4	59.2	55.6	55.0	56.6
2048	SeerAttn-R	53.5	51.6	48.4	51.2	**57.0**	53.1	47.6	52.6	58.9	54.3	51.5	54.9
	SAS	**53.6**	**52.1**	**48.8**	**51.5**	**57.0**	**53.9**	**49.3**	**53.4**	**59.2**	**54.4**	**53.9**	**55.8**
4096	SeerAttn-R	**54.0**	52.1	49.7	51.9	**57.4**	53.9	50.0	53.8	**59.0**	**54.8**	53.5	55.7
	SAS	53.8	**52.4**	**49.8**	**52.0**	**57.4**	**54.1**	**50.4**	**53.9**	58.9	**54.8**	**54.8**	**56.2**

Table 4: Evaluation results on BFCL.

Budget	Method	Qwen3-4B	Qwen3-8B	Qwen3-14B
Full	Full Attn	35.75	42.75	44.50
2048	SeerAttn-R	29.00	34.38	38.88
	SAS	**32.50**	**36.75**	**39.50**
4096	SeerAttn-R	33.38	39.25	43.88
	SAS	**34.38**	**41.25**	**44.00**

Table 5: Evaluation results on VitaBench with Qwen3-14B.

Budget	Method	Delivery			Instore			OTA		
		Avg@4	Pass@4	Pass^4	Avg@4	Pass@4	Pass^4	Avg@4	Pass@4	Pass^4
Full	Full Attn	29.0 (3.3)	62.0 (4.8)	8.0 (2.7)	27.2 (2.9)	64.0 (4.8)	5.0 (2.2)	12.5 (2.1)	37.0 (4.8)	1.0 (1.0)
2048	SeerAttn-R	**32.2** (3.0)	**67.0** (4.7)	4.0 (2.0)	19.0 (2.5)	48.4 (5.1)	2.1 (1.5)	4.2 (1.4)	14.9 (4.3)	0.0 (0.0)
	SAS	30.0 (3.3)	61.0 (4.9)	**9.0** (2.9)	**19.2** (2.5)	**51.0** (5.0)	**3.0** (1.7)	**10.2** (1.8)	**28.0** (4.5)	0.0 (0.0)
4096	SeerAttn-R	32.5 (3.1)	63.0 (4.8)	**4.0** (2.0)	26.7 (2.8)	56.0 (4.9)	1.0 (1.0)	11.7 (2.2)	**29.0** (4.5)	1.0 (1.0)
	SAS	**34.2** (3.0)	**68.0** (4.6)	**4.0** (2.0)	**28.7** (3.0)	**61.0** (4.9)	**5.0** (2.2)	**12.3** (2.3)	27.0 (4.4)	**2.0** (1.4)

Long context understanding performance. Table 3 reports LongBench results with selector trained only on math data and evaluated on long-context inputs. Despite the distribution shift, SAS outperforms SeerAttention-R at nearly every budget, with the gap widening on longer inputs where selection quality matters most: at budget 2048 on Qwen3-14B, SAS improves the 8K+ bucket by +2.4 (53.9 vs. 51.5). At budget 4096, SAS nearly matches full attention (e.g., 56.2 vs. 56.6 average on Qwen3-14B), showing that a math-trained selector still transfers effectively to long-context task.

Agentic task performance. On BFCL multi-turn (Table 4), SAS beats SeerAttention-R across all backbones and budgets, e.g., +3.5 on Qwen3-4B at budget 2048, and nearly closes the gap to full attention at budget 4096 (e.g., 44.00 vs. 44.50 on Qwen3-14B). The advantage carries over to VitaBench (Table 5): at budget 4096, SAS leads SeerAttention-R on most metrics across the Delivery, Instore, and OTA scenarios and approaches full attention. showing that end-to-end selection remains reliable in realistic long-horizon tool-use settings.

6 Conclusion

We presented SAS, a simple yet effective paradigm for end-to-end post-training attention sparsification. By replacing layer-wise attention distillation with direct optimization of language modeling loss, SAS overcomes the limitations of surrogate supervision and better captures cross-layer dependencies. Our empirical analysis identifies three key choices for stable, informative gradient flow in context selection: log-space gate injection, normalized gate activation, and preservation of continuous rankings. Fused Triton kernels make SAS practical for long-context training. Extensive evaluations show that SAS outperforms baselines across reasoning, long-context understanding, and agentic tasks.

References

[1] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. In *International Conference on Machine Learning*, pages 47901–47911, 2024.

[2] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

[3] 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. In *Advances in Neural Information Processing Systems*, 2025.

[4] Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, et al. Seerattention-r: Sparse attention adaptation for long reasoning. In *International Conference on Learning Representations*, 2026.

[5] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[6] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[7] Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, et al. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. In *International Conference on Learning Representations*, 2026.

[8] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

[9] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025.

[10] Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. In *Advances in Neural Information Processing Systems*, 2025.

[11] 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.

[12] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[13] 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.

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

[15] Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. Lserve: Efficient long-sequence llm serving with unified sparse attention. *Proceedings of Machine Learning and Systems*, 7, 2025.

[16] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. In *Advances in Neural Information Processing Systems*, 2025.

[17] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. In *International Conference on Machine Learning (ICML)*, 2025.

[18] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *International Conference on Machine Learning*, pages 69819–69831. PMLR, 2025.

[19] 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. In *Third Conference on Language Modeling*, 2026.

[20] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025.

[21] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32631–32652, 2025.

[22] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Anima Anandkumar, Abedelkadir Asi, and Junjie Hu. R-kv: Redundancy-aware kv cache compression for reasoning models. In *Advances in Neural Information Processing Systems*, 2025.

[23] Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. Omnikv: Dynamic context selection for efficient long-context llms. In *The Thirteenth International Conference on Learning Representations*, 2025.

[24] Pierre-Emmanuel Mazaré, Gergely Szilvasy, Maria Lomeli, Francisco Massa, Naila Murray, Hervé Jégou, and Matthijs Douze. Inference-time sparse attention with asymmetric indexing. *arXiv preprint arXiv:2502.08246*, 2025.

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

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

[27] 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.

[28] Lei Zhu, Xinjiang Wang, Zhanghan Ke, Wayne Zhang, and Rynson WH Lau. Biformer: Vision transformer with bi-level routing attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10323–10333, 2023.

[29] 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)*, pages 23078–23097, 2025.

[30] MiniCPM Team, Chaojun Xiao, Yuxuan Li, Xu Han, Yuzhuo Bai, Jie Cai, Haotian Chen, Wentong Chen, Xin Cong, Ganqu Cui, et al. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

[31] 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.

[32] Jintao Zhang, Rundong Su, Chunyu Liu, Jia Wei, Ziteng Wang, Pengle Zhang, Haoxu Wang, Huiqiang Jiang, Haofeng Huang, Chendong Xiang, et al. A survey of efficient attention methods: Hardware-efficient, sparse, compact, and linear attention. `https://attention-survey.github.io`, 2025.

[33] Yutao Sun, Zhenyu Li, Yike Zhang, Tengyu Pan, Bowen Dong, Yuyi Guo, and Jianyong Wang. Efficient attention mechanisms for large language models: A survey. *arXiv preprint arXiv:2507.19595*, 2025.

[34] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[35] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025.

[36] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[37] 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, pages 21875–21895, 2024.

[38] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

[39] Art of Problem Solving. Aime problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

[40] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[41] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[42] Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi GU, Hui Su, and Xunliang Cai. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. In *The Fourteenth International Conference on Learning Representations*, 2026.

[43] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024.

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[45] Qianli Ma, Yaowei Zheng, Zhelun Shi, Zhongkai Zhao, Bin Jia, Ziyue Huang, Zhiqi Lin, Youjie Li, Jiacheng Yang, Yanghua Peng, et al. Veomni: Scaling any modality model training with model-centric distributed recipe zoo. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

[46] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

[47] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

[48] Xiang Hu, Xinyu Wei, Hao Gu, Minshen Zhang, Tian Liang, Huayang Li, Lei Zhu, Yan Wang, Sirui Han, Yushi Bai, et al. Hierarchical sparse attention done right: Toward infinite context modeling. *arXiv preprint arXiv:2607.02980*, 2026.

[49] Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026.

[50] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[51] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[52] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

[53] 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. *arXiv preprint arXiv:1803.05457*, 2018.

[54] 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 *NAACL*, 2019.

[55] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[56] Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. Cmath: Can your language model pass chinese elementary school math test?, 2023.

[57] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[58] Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. In *International Conference on Machine Learning*, pages 16568–16621, 2024.

[59] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[60] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[61] Lin Niu, Xin Luo, Linchuan Xie, Yifu Sun, Guanghua Yu, Jianchen Zhu, and S Kevin Zhou. Stem: Rethinking causal information flow in sparse attention. In *International Conference on Machine Learning*, 2026.

[62] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

A Inference Implementation

We implement our method as a native attention backend in SGLang [46], built on top of its paged KV cache and FlashInfer [47] attention kernels. Following the standard prefill/decode pipeline of LLM serving, the prefill phase uses dense attention, while the decode phase uses block sparse attention. At each decoding step, the trained selector scores all cached blocks, selects the Top-K blocks for each query group, and gathers only the corresponding KV blocks for the FlashInfer sparse decode kernel. This reduces the decoding complexity from $O(n)$ to $O(|\mathcal{S}|)$, independent of the total context length n. The backend supports grouped query attention (GQA) with per-group block selection, CUDA graph capture for the decode path, and batched concurrent decoding. Since our method and SeerAttention-R share the same AttnGate architecture and inference-time block selection procedure, differing only in how the gate is trained, both run through the same backend by simply loading their respective gate checkpoints; the dense baseline uses the same FlashInfer paged-attention implementation on an identical model configuration.

B Extension to Continued Pretraining

Baselines, model, and evaluation. We follow HiLS-Attention [48] to conduct a continued pretraining experiment. We start from the OLMo3-7B [49] stage-1 base checkpoint and apply SAS with block size 64 and Top-K 32. Instead of the AttnGate used in post-training, here we design a lightweight selector with a dedicated retrieval query/key projection, initialized from the attention projections. All query heads within a group share the same selected blocks. We compare with three baselines built on the same checkpoint: *OLMo3-Base*, the original dense model; *OLMo3-512SWA*, a sliding-window baseline with continued pretraining; and *HiLS-Attn-RoPE*, the RoPE variant of HiLS-Attention (HiLS-Attn RoPE-Q-Cal in their paper). We evaluate on general downstream tasks, spanning general knowledge (MMLU [50], GPQA [51], HellaSwag [52], ARC-c [53], BoolQ [54], Race [55]), mathematics (CMath [56], GSM8K [57]), and code (CRUX [58], HumanEval+ [59], MBPP+ [60]), and on long-context understanding with LongBench.

Training setup. Here we train the whole model, both the OLMo3 backbone and the selector, jointly. Following HiLS-Attention, we train on the OLMo3 pretraining corpus with a maximum sequence length of 8,192 for 13,000 steps with a global batch size of 512, corresponding to about 50B tokens, using AdamW with the learning rate warmed up to 2e-4 and cosine decayed to 2e-5.

Table 6: Evaluation results on general downstream tasks for OLMo3-7B continued pretraining.

Task	Olmo3-Base	Olmo3-512SWA	HiLS-Attn RoPE	SAS-RoPE
General Knowledge				
MMLU (5-shot)	59.90	**59.12**	56.69	58.58
GPQA (5-shot)	29.29	**31.31**	24.75	26.77
Hellaswag (10-shot)	44.17	42.96	33.17	**50.63**
ARC-c (25-shot)	53.56	**55.59**	54.92	52.54
BoolQ (5-shot)	61.01	**64.22**	63.43	62.87
Race (3-shot)	73.89	72.97	69.50	**74.05**
Mathematics				
CMath	41.53	39.98	**42.44**	42.17
GSM8K	37.00	33.43	**35.71**	34.42
Code				
CRUX	24.62	24.50	**25.62**	19.25
HumanEval+	20.10	19.50	18.90	**20.10**
MBPP+	37.60	32.30	33.30	**34.60**
Average	43.88	43.24	41.68	**43.28**

Downstream task and long-context performance. Table 6 shows that, among the continued-pretrained models, SAS achieves the best average score (43.28), performing comparably to the sliding-window baseline OLMo3-512SWA (43.24), clearly ahead of HiLS-Attn-RoPE (41.68), and nearly matching the dense OLMo3-Base (43.88). SAS leads on knowledge tasks such as HellaSwag (50.63) and Race (74.05), and remains competitive on mathematics and code. Table 7 further shows that SAS attains a tied-best LongBench average (30.0), clearly surpassing the dense base (29.0) and

Table 7: Evaluation results on LongBench for OLMo3-7B continued pretraining.

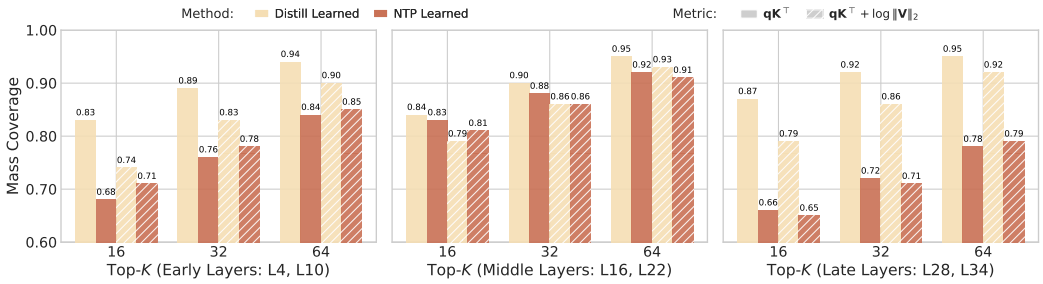
Model	<8K						>8K						Average
	SDoc	MDoc	Summ.	Few-shot	Synth.	Code	SDoc	MDoc	Summ.	Few-shot	Synth.	Code	
Olmo3-Base	36.9	33.9	23.4	63.8	5.8	61.8	11.4	14.4	12.1	36.0	3.3	41.0	29.0
Olmo3-512SWA	**37.4**	27.8	22.1	**64.1**	4.1	59.4	10.0	14.2	12.9	33.6	3.2	32.9	28.0
HiLS-Attn-RoPE	37.2	33.1	22.2	61.0	**5.4**	**60.6**	**17.5**	**17.3**	14.0	**42.0**	**4.3**	47.1	**30.0**
SAS-RoPE	34.4	**34.7**	**22.8**	**64.1**	4.1	58.8	16.1	16.3	**14.8**	41.4	3.4	**48.3**	**30.0**

the sliding-window continued-pretraining baseline (28.0). The gains concentrate on long inputs (>8K), where accurate block selection matters most, providing initial evidence that end-to-end sparse selection also transfers to the continued pretraining stage.

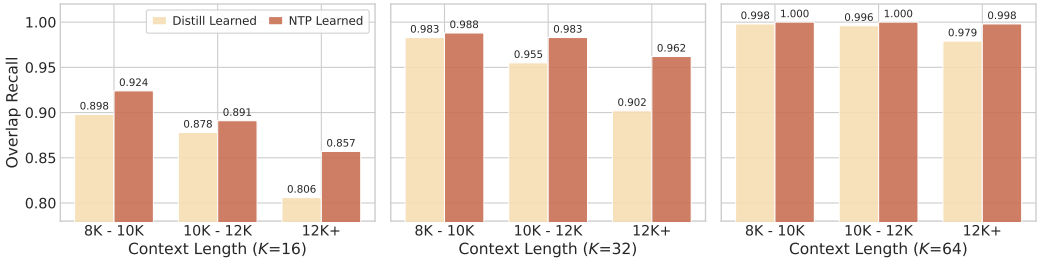
C Analysis

This section analyzes *why* end-to-end training yields a better sparse selector. We first inspect the learned block selection, and find that SAS covers less attention mass per layer yet its cross-layer union better matches the full attention oracle. We then show that this more effective selection lets SAS reach answers with shorter reasoning traces and fewer truncations. Finally, we quantify the end-to-end decode efficiency and report substantial speedups over full attention.

C.1 Analyzing the Learned Block Selection



(a)



(b)

Figure 5: Analysis of the learned block selection, comparing the block selection learned by distillation (SeerAttention-R) and by end-to-end NTP (next-token prediction) training (SAS). (a) Per-layer attention mass coverage of the selected Top-K blocks, under both raw weight $\mathbf{qK}^\top$, and value-aware weight $\mathbf{qK}^\top + \log\|\mathbf{V}\|_2$ from STEM [61]. SAS covers *less* mass than distillation. (b) Overlap recall of the cross-layer union of selected blocks against the full attention oracle. SAS attains *higher* recall than distillation, consistent with more complementary cross-layer selection.

Per-layer attention mass coverage. A natural way to assess a sparse selector is how much attention mass its selected token set $\mathcal{S}$ covers. Reusing the attention logit $z_i = \mathbf{q}\mathbf{k}_i^\top$, we define the covered mass ratio with a per token weight w_i as

$$\mathrm{Cov}(\mathcal{S}) = \frac{\sum_{i\in\mathcal{S}} w_i}{\sum_j w_j}. \tag{10}$$

We consider two instantiations of w_i. The first is $w_i = \exp(z_i)$, which depends only on the attention weight. The second is $w_i = \exp(z_i + \log \|\mathbf{v}_i\|_2)$, which additionally incorporates the magnitude of the value vector $\mathbf{v}_i$, following [61]. As our objective is the final language modeling loss, which depends on all attention components including $\mathbf{V}$, we expect SAS to favor blocks that contribute larger attention outputs. However, Figure 5a shows that SAS covers *less* mass than distillation under both metrics in almost all layers. One possible explanation is that distillation explicitly optimizes for full attention coverage as its training target, naturally encouraging higher attention mass coverage, whereas SAS is never trained to match the full attention distribution.

Overlap with the full attention oracle. The per-layer coverage above, however, ignores that selection is distributed across layers: a block missed at one layer can still be attended at another. We therefore take, for each method, the union of its selected blocks across all layers and measure how much of the full attention model's attended blocks it recovers. On long training sequences (>8K) with Qwen3-4B, we obtain the oracle set from full attention; since it uses GQA, we aggregate the per-head Top-K within each group into one group-level set by majority vote. Taking the union of these sets over all L layers gives the oracle set $\mathcal{I}^{\star}_{\text{all}}$, and applying the same procedure to a method's own selection gives $\mathcal{I}_{\text{all}}$. We then report the overlap recall

$$\text{Recall}(\mathcal{I}_{\text{all}}, \mathcal{I}^{\star}_{\text{all}}) = \frac{|\mathcal{I}_{\text{all}} \cap \mathcal{I}^{\star}_{\text{all}}|}{|\mathcal{I}^{\star}_{\text{all}}|}, \qquad \mathcal{I}_{\text{all}} = \bigcup_{j=1}^{L} \mathcal{I}_j, \tag{11}$$

where $\mathcal{I}_j$ is the per-layer block set at layer j and $\mathcal{I}^{\star}_{\text{all}}$ is defined analogously. As shown in Figure 5b, SAS attains consistently higher overlap recall across context lengths and budgets. This matches our expectation: distillation fits the full attention layer by layer as a local objective, whereas SAS is trained end-to-end, so its block selection is optimized jointly across layers. As a result, even though SAS retains less attention mass within each individual layer, its per-layer choices are more complementary, and their union attains higher overlap recall against the oracle.

C.2 Analyzing the Generation Behavior

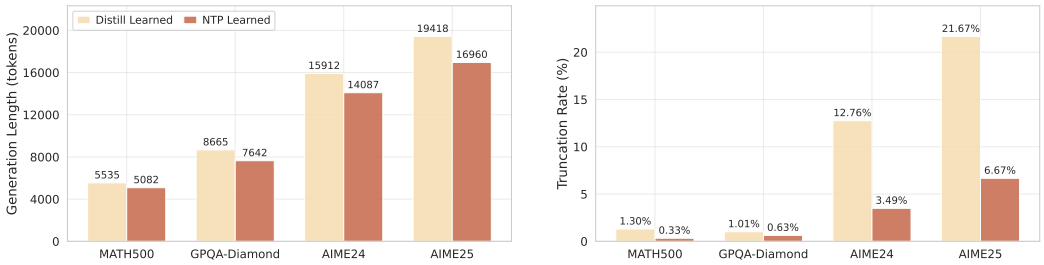
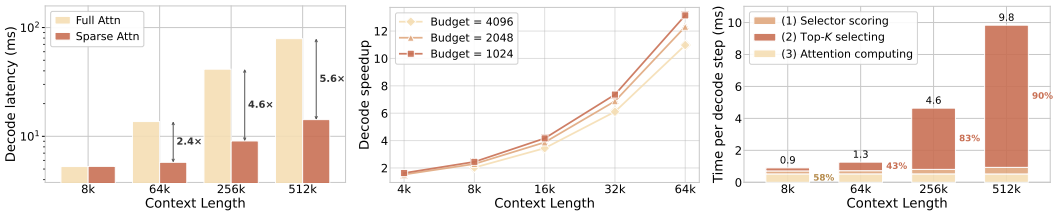


Figure 6: Generation behavior of distillation (SeerAttention-R) and end-to-end NTP (SAS) on Qwen3-4B at a 4096 token budget across four reasoning benchmarks. Left: average generation length; SAS generates shorter traces. Right: truncation rate under a maximum generation length of 32,768; SAS truncates less often. These trends are consistent with more effective context selection, suggesting SAS reaches answers with fewer tokens, with the gap largest on the harder AIME tasks.

Beyond final accuracy, the quality of sparse selection also affects token efficiency, measured here by the generation length needed to reach an answer. When important context is dropped, the model tends to generate longer, less focused reasoning traces and is more likely to hit the maximum generation length before finishing. We therefore compare SeerAttention-R and SAS on Qwen3-4B at a 4096 token budget, reporting the average generation length and the truncation rate under a maximum length of 32,768 (Figure 6). Across all four reasoning benchmarks, SAS produces shorter generations and a lower truncation rate, with the gap widening on the harder AIME tasks where long reasoning is required. These results are consistent with more effective context selection: they suggest that SAS can complete its reasoning with fewer tokens and less frequent truncation, complementing the accuracy gains in Section 5.2.



(a) (b) (c)

Figure 7: End-to-end decode efficiency of SAS versus dense attention on Qwen3-4B in SGLang (single GPU, CUDA graphs, steady-state decode). (a) At batch 1, dense latency grows linearly with context length, while SAS remains nearly constant, achieving up to 5.6× lower latency at 512K. (b) At batch 8, speedup reaches ∼13× and is largely insensitive to the token budget. (c) Sparse-decode latency breakdown: attention cost remains constant, while Top-K selection dominates at long contexts.

C.3 End-to-End Decode Efficiency

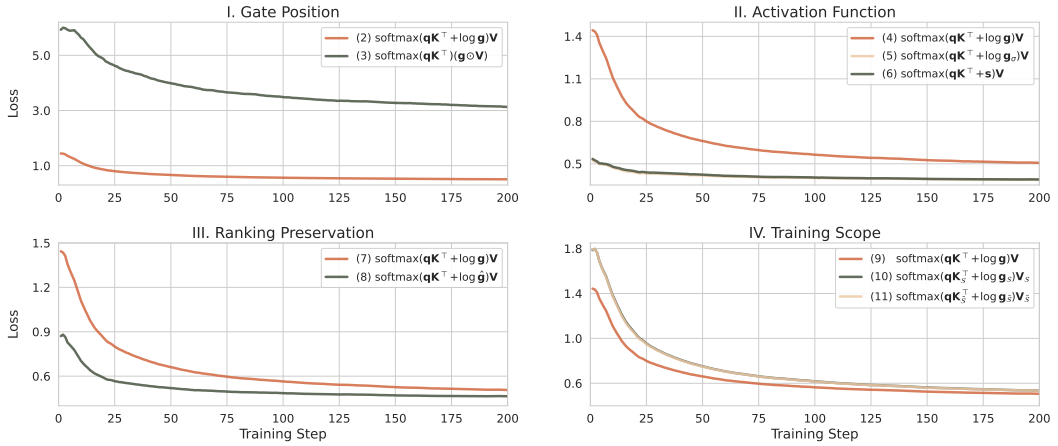
Decode speedup over full attention. We measure wall-clock decode efficiency by serving Qwen3-4B in SGLang with our block sparse attention against full attention under an identical configuration: a single GPU (tensor parallel size 1), CUDA graphs enabled, chunked prefill, and a steady-state decode measurement (the input is prefilled, then 512 generated tokens are timed). Since SAS sparsifies *decode-time* attention only, prefill is unchanged, and the reported numbers isolate the per-token decode cost. Full attention must read the entire KV cache at every step, so its decode latency grows linearly with context length, whereas SAS reads only a fixed token budget of blocks and stays essentially flat. The gap therefore widens with context: at batch 1 (Figure 7a) the two are near parity at 8K but SAS is 2.4×, 4.6×, and 5.6× faster at 64K, 256K, and 512K. Increasing the batch amortizes the fixed per-step overhead and further favors sparsity: at batch 8 (Figure 7b) the speedup reaches ∼13× at 64K. A tighter token budget reads fewer blocks and is thus faster, but the difference across budgets is modest, indicating that the benefit comes primarily from avoiding the full KV read rather than from the exact budget.

Decode step cost breakdown. To understand the scaling behavior, we decompose one sparse attention decode step into its three components: selector scoring (the gate query projection and its dot-product against the block summaries), Top-K block selection, and the attention compute over the selected blocks (Figure 7c). The attention compute is context invariant, since the number of blocks read is capped by the token budget regardless of sequence length. Selector scoring, by contrast, scans all block summaries and thus grows with context in absolute terms, rising from well below the attention compute at 8K to roughly matching it at 512K; extrapolating to longer context, it is expected to surpass the attention compute. The Top-K selection grows even faster, as it must rank all candidate blocks, and comes to dominate the step, shifting from 21% at 8K (where attention compute dominates at 58%) to 90% at 512K. This identifies the selection stage, Top-K ranking together with the growing selector scoring, rather than the attention compute, as the primary bottleneck and the main target for further kernel optimization at very long context.

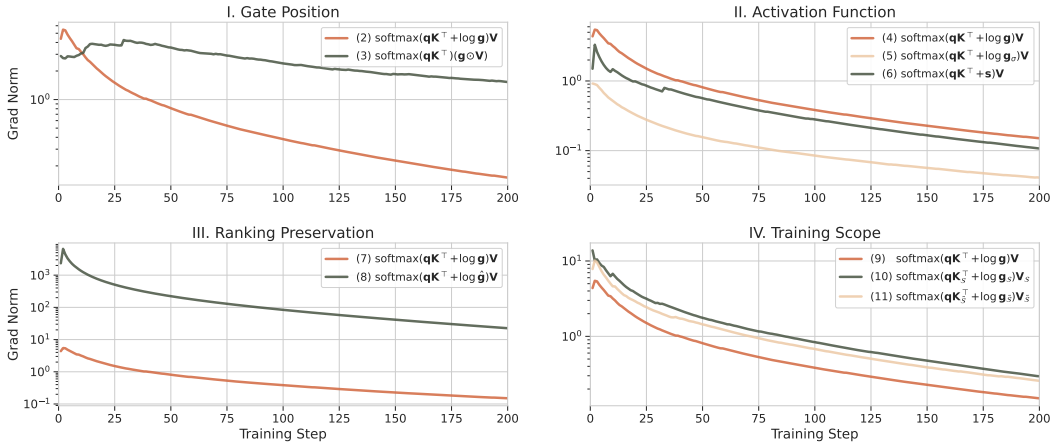
D Training Dynamics

We provide the loss and gradient norm curves for the controlled ablation studies in Section 4.2 in Figure 8. The training dynamics reveal distinct optimization behaviors across designs. The inner-softmax gate consistently achieves lower training loss than the outer-gate formulation. Independent sigmoid gates and raw-logit injection achieve even lower losses, but fail to improve sparsification, as both effectively approach ungated attention and learn little discriminative block ranking. Similarly, although the STE formulation yields lower training loss, its final sparsification performance remains inferior, indicating that training loss alone is insufficient to assess ranking quality. Hard gating further reduces loss while substantially increasing the gradient norm, suggesting more aggressive optimization. Sparse-scope training starts with higher loss but gradually approaches full-scope training, indicating that the two scopes can eventually reach comparable solutions despite different

gradient dynamics. Overall, effective selector training requires not only loss minimization but also preservation of a meaningful and discriminative block ranking signal.



(a)



(b)

Figure 8: Training dynamics for different gated sparsification designs. Inner softmax gate yields lower training loss than outer gate. Independent sigmoid gates and raw-logit injection achieve lower loss than normalized softmax gates. Hard gating reduces the training loss, but substantially increases the gradient norm. Sparse scope training starts with a higher loss but gradually approaches full scope training, while introducing noise yields little difference.

E Limitation

We further explore the scalability of the learned selector to RULER [62]. Specifically, we sample 0.5B tokens from the `allenai/dolma3_longmino_mix-100B-1125` dataset [3] and train the selector on 64K sequences using a packing strategy. During evaluation, we use the non-thinking mode and extend the context window to 128K with YaRN. As shown in Table 8, while SAS consistently improves over SeerAttention-R under shorter context lengths, its performance still degrades substantially as the context length increases, leaving a considerable gap from full attention. We believe this limitation primarily stems from the use of pooling based block summaries, which may struggle to preserve fine grained, localized information, such as needle-like signals, within each block, especially as the context length increases. Designing more expressive yet efficient selectors that can capture fine-grained information under context block compression is therefore an important direction for future work.

[3] `https://huggingface.co/datasets/allenai/dolma3_longmino_mix-100B-1125`

Table 8: Evaluation results on the RULER benchmark.

Budget	Method	Qwen3-4B						Qwen3-8B						Qwen3-14B					
		4K	8K	16K	32K	64K	128K	4K	8K	16K	32K	64K	128K	4K	8K	16K	32K	64K	128K
Full	Full Attn	92.95	91.27	86.61	79.25	73.17	63.81	93.10	90.13	89.59	87.04	77.25	73.87	95.28	92.71	92.91	92.15	92.19	82.23
2048	SeerAttn-R	92.08	79.28	56.01	30.29	18.12	13.09	92.15	80.16	63.76	42.16	24.49	13.99	94.40	84.50	62.80	44.73	30.73	14.95
	SAS	92.82	78.22	63.94	39.54	25.65	17.92	92.45	81.39	69.34	46.94	30.31	19.38	94.43	85.10	69.01	54.76	34.12	23.80
4096	SeerAttn-R	93.49	87.90	72.93	47.41	27.96	16.29	92.96	90.13	77.39	57.92	37.43	18.75	95.28	91.62	76.77	64.96	42.34	22.36
	SAS	93.49	89.09	75.72	51.90	35.53	21.87	92.96	87.96	79.88	63.50	39.18	23.46	95.28	91.52	83.35	66.82	45.92	29.95

F Final Performance of Sparse vs. Full Training Scope

The gradient analysis in Appendix G.3 shows that sparse scope updates unselected blocks only indirectly, which slows early optimization but does not change the final ranking quality. Table 9 reports the final accuracy of full scope and sparse scope across Qwen3-4B/8B/14B and token budgets of 1024, 2048, and 4096. Consistent with the training dynamics in Table 1, sparse scope matches full scope at convergence across all model scales and budgets, while incurring lower training cost.

Table 9: Final accuracy (%) of full training scope versus sparse training scope across model scales and token budgets. Standard deviations are shown in parentheses.

Budget	Train scope	MATH500			GPQA-Diamond			AIME24			AIME25		
		4B	8B	14B	4B	8B	14B	4B	8B	14B	4B	8B	14B
1024	full	91.33 (1.1)	91.71 (1.1)	93.08 (1.0)	51.42 (2.9)	55.62 (2.8)	61.77 (2.9)	-	-	-	-	-	-
	sparse	90.65 (1.1)	91.27 (1.1)	92.93 (1.0)	50.41 (2.8)	53.17 (2.9)	61.14 (2.9)	-	-	-	-	-	-
2048	full	93.00 (1.0)	93.09 (1.1)	93.56 (1.0)	54.40 (2.9)	58.43 (2.9)	64.14 (2.8)	68.13 (6.7)	70.73 (6.6)	76.67 (6.7)	55.39 (7.3)	59.27 (7.3)	64.04 (7.2)
	sparse	93.47 (1.0)	93.17 (1.0)	93.54 (1.0)	54.86 (2.8)	58.74 (2.8)	65.09 (2.8)	68.85 (6.6)	70.73 (6.7)	77.76 (6.6)	56.38 (7.3)	58.41 (7.2)	64.19 (7.1)
4096	full	93.10 (1.0)	93.40 (1.0)	94.12 (1.0)	54.80 (2.8)	60.32 (2.8)	64.39 (2.9)	71.17 (6.6)	73.52 (6.7)	78.49 (6.6)	61.30 (7.3)	64.24 (7.1)	67.58 (7.0)
	sparse	93.67 (1.0)	94.23 (1.0)	95.23 (0.8)	55.11 (2.9)	60.42 (2.8)	65.03 (2.8)	71.72 (6.8)	73.49 (6.7)	78.28 (6.4)	59.97 (7.3)	63.41 (7.2)	67.29 (6.9)

G Gradient Derivation

G.1 Gradient Derivation for Gate Position

We derive the gate gradients for outer and inner gate injection. First, we let

$$z_i = \mathbf{q}\mathbf{k}_i^\top, \qquad p_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \qquad b(i) = m \ \text{ if } i \in B_m.$$

Outer gate. For outer gating, the attention probability is computed before applying the gate:

$$\mathbf{o}_{outer} = \sum_i p_i g_{b(i)} \mathbf{v}_i.$$

Since p_i does not depend on g_m, the outer-gate gradient is

$$\begin{aligned} dg_m^{outer} &= \frac{\partial \mathcal{L}}{\partial g_m} = d\mathbf{o}^\top \frac{\partial \mathbf{o}_{outer}}{\partial g_m} \\ &= d\mathbf{o}^\top \sum_i p_i \frac{\partial g_{b(i)}}{\partial g_m} \mathbf{v}_i \\ &= \sum_{i \in B_m} p_i \, d\mathbf{o}^\top \mathbf{v}_i. \end{aligned}$$

Inner gate. For inner gating, the gate is injected into the attention logits:

$$\mathbf{o}_{inner} = \sum_i \tilde{p}_i \mathbf{v}_i, \qquad \tilde{p}_i = \frac{g_{b(i)} \exp(z_i)}{\sum_j g_{b(j)} \exp(z_j)}.$$

For a block gate g_m, its derivative is

$$\begin{aligned}
\frac{\partial \tilde{p}_i}{\partial g_m} &= \frac{\mathbb{I}[b(i)=m]\exp(z_i)\sum_j g_{b(j)}\exp(z_j) - g_{b(i)}\exp(z_i)\sum_{j\in B_m}\exp(z_j)}{\left(\sum_j g_{b(j)}\exp(z_j)\right)^2} \\
&= \frac{\mathbb{I}[b(i)=m]\exp(z_i)}{\sum_j g_{b(j)}\exp(z_j)} - \frac{g_{b(i)}\exp(z_i)}{\sum_j g_{b(j)}\exp(z_j)} \cdot \frac{\sum_{j\in B_m}\exp(z_j)}{\sum_j g_{b(j)}\exp(z_j)} \\
&= \frac{\mathbb{I}[b(i)=m]}{g_m}\frac{g_{b(i)}\exp(z_i)}{\sum_j g_{b(j)}\exp(z_j)} - \frac{g_{b(i)}\exp(z_i)}{\sum_j g_{b(j)}\exp(z_j)} \cdot \sum_{j\in B_m}\frac{\exp(z_j)}{\sum_r g_{b(r)}\exp(z_r)} \\
&= \frac{\mathbb{I}[b(i)=m]}{g_m}\tilde{p}_i - \tilde{p}_i \sum_{j\in B_m}\frac{1}{g_m}\frac{g_m\exp(z_j)}{\sum_r g_{b(r)}\exp(z_r)} \\
&= \frac{\mathbb{I}[b(i)=m]}{g_m}\tilde{p}_i - \tilde{p}_i \sum_{j\in B_m}\frac{\tilde{p}_j}{g_m} \\
&= \tilde{p}_i\left(\frac{\mathbb{I}[b(i)=m]}{g_m} - \sum_{j\in B_m}\frac{\tilde{p}_j}{g_m}\right).
\end{aligned}$$

Therefore,

$$\begin{aligned}
dg_m^{inner} &= \frac{\partial \mathcal{L}}{\partial g_m} = d\mathbf{o}^\top \frac{\partial \mathbf{o}_{inner}}{\partial g_m} \\
&= d\mathbf{o}^\top \sum_i \frac{\partial \tilde{p}_i}{\partial g_m}\mathbf{v}_i \\
&= d\mathbf{o}^\top \sum_i \tilde{p}_i\left(\frac{\mathbb{I}[b(i)=m]}{g_m} - \sum_{j\in B_m}\frac{\tilde{p}_j}{g_m}\right)\mathbf{v}_i \\
&= d\mathbf{o}^\top \left[\frac{1}{g_m}\sum_{i\in B_m}\tilde{p}_i\mathbf{v}_i - \frac{1}{g_m}\left(\sum_{j\in B_m}\tilde{p}_j\right)\sum_i \tilde{p}_i\mathbf{v}_i\right] \\
&= \frac{1}{g_m} d\mathbf{o}^\top \sum_{i\in B_m}\tilde{p}_i\left(\mathbf{v}_i - \mathbf{o}_{inner}\right) \\
&= \sum_{i\in B_m}\frac{\tilde{p}_i}{g_m} d\mathbf{o}^\top\left(\mathbf{v}_i - \mathbf{o}_{inner}\right).
\end{aligned}$$

G.2 Gradient Derivation for Ranking Preservation

Both variants use inner gate injection, so both inherit the inner-gate gradient from Appendix G.1, with $z_i = \mathbf{q}\mathbf{k}_i^\top$. They differ only in the attention weight that the forward normalizer assigns to each token.

Soft gating. All C blocks stay active, so the normalizer runs over the full context and block B_m appears in its own normalizer:

$$\tilde{p}_i^{soft} = \frac{g_{b(i)}\exp(z_i)}{\sum_j g_{b(j)}\exp(z_j)} \le 1, \qquad dg_m = \sum_{i\in B_m}\frac{\tilde{p}_i^{soft}}{g_m} d\mathbf{o}^\top\left(\mathbf{v}_i - \mathbf{o}\right).$$

Hence $\tilde{p}_i^{soft} \le 1$ for every token, and the gate gradient dg_m stays bounded.

Hard gating with STE. The unselected blocks satisfy $\log g_m^{hard} = -\infty$, so the forward normalizer sums only over the selected set $\mathcal{S} = \bigcup_{m\in\mathcal{I}} B_m$. STE still propagates a gate gradient to a non-selected block $m \notin \mathcal{I}$, whose ranking signal evaluates the attention mass $\tilde{p}_i^{hard}$ its tokens would receive.

Because B_m is excluded from $\mathrm{LSE}_{\mathcal{S}}$, this weight has no upper bound:

$$\tilde{p}_i^{hard} = \frac{\exp(z_i)}{\sum_{j\in\mathcal{S}}\exp(z_j)} = \exp(z_i - \mathrm{LSE}_{\mathcal{S}}) \le \exp\left(z_i - \max_{j\in\mathcal{S}} z_j\right), \qquad \mathrm{LSE}_{\mathcal{S}} = \log\sum_{j\in\mathcal{S}}\exp(z_j),$$

which grows exponentially once a dropped block outscores the selected set, $z_i > \max_{j\in\mathcal{S}} z_j$. The unbounded $\tilde{p}_i^{hard}$ enters dg_m and, accumulated over all dropped blocks, tokens, and heads, inflates the gate gradient.

G.3 Gradient Derivation for Training Scope

We derive the score gradients under sparse and full scopes. Let the activation function $\mathrm{softmax}$.

For any gate gradient dg_ℓ, we first give the $\mathrm{softmax}$ backward formulation

$$\begin{aligned} ds_m &= \frac{\partial \mathcal{L}}{\partial s_m} = \sum_{\ell=1}^{C} \frac{\partial \mathcal{L}}{\partial g_\ell}\frac{\partial g_\ell}{\partial s_m} \\ &= \sum_{\ell=1}^{C} dg_\ell\, g_\ell\, (\mathbb{I}[\ell = m] - g_m) \\ &= g_m dg_m - g_m \sum_{\ell=1}^{C} g_\ell dg_\ell \\ &= g_m \left(dg_m - \sum_{\ell=1}^{C} g_\ell dg_\ell\right). \end{aligned}$$

Sparse routing scope. Let $\mathcal{I} = \text{Top-}K(\mathbf{s}, K)$ denote the selected block indices. Under sparse routing scope, only selected blocks participate in the gated attention computation. For an unselected block $m \notin \mathcal{I}$, the output does not depend on g_m, so $dg_m^{sparse} = 0$.

Moreover, only selected blocks have nonzero gate gradients:

$$dg_\ell^{sparse} = 0, \qquad \ell \notin \mathcal{I}.$$

Therefore, for $m \notin \mathcal{I}$,

$$\begin{aligned} ds_m^{sparse} &= g_m \left(dg_m^{sparse} - \sum_{\ell=1}^{C} g_\ell dg_\ell^{sparse}\right) \\ &= -g_m \sum_{\ell=1}^{C} g_\ell dg_\ell^{sparse} \\ &= -g_m \sum_{\ell\in\mathcal{I}} g_\ell dg_\ell^{sparse}. \end{aligned}$$

Full routing scope. Under full routing scope, all blocks participate in the gated attention computation during training. Therefore, an unselected block $m \notin \mathcal{I}$ still has its own gate gradient dg_m^{full}. Using the same softmax backward identity,

$$\begin{aligned} ds_m^{full} &= g_m \left(dg_m^{full} - \sum_{\ell=1}^{C} g_\ell dg_\ell^{full}\right) \\ &= -g_m \left(-dg_m^{full} + \sum_{\ell=1}^{C} g_\ell dg_\ell^{full}\right). \end{aligned}$$

H Kernel Implementation

We provide a custom Triton implementation for the sparse scope soft gated attention used during SAS training. The block gate is organized as a table $\log \mathbf{G}$, where each query position and attention

head holds one log gate value for every previous context block. Block selection is encoded by a per query threshold τ: a block is *selected* when its log gate reaches the threshold ($\log g_m \geq \tau$), which realizes the Top-K set $\mathcal{I}$ without an explicit sort. Non selected blocks are masked out of the softmax so that only the selected set $\mathcal{S}$ participates.

For simplicity of presentation, the pseudocode omits the grouped query attention, and variable length packing that the full implementation supports, and focuses on the gate injection and the sparse scope.

Forward pass. The kernel traverses causal key value blocks in a tiled manner. The current (self) block always uses a unit gate and a causal mask, so local causal attention is always preserved. For each earlier block, the kernel maps key positions to their block index, loads the corresponding log gate, and adds it to the $\mathbf{q}\mathbf{K}_{\mathcal{S}}^{\top}$ scores before the online softmax update; blocks below the threshold are masked to $-\infty$ and an entire block is skipped when no query in the tile selects it.

Backward pass. We follow the standard recomputation based attention backward procedure. The kernel first computes the row-wise correction term $\mathbf{\Delta} = \sum_d \mathbf{O}_d \cdot d\mathbf{O}_d$. It then recomputes the gated attention probabilities from the saved log-sum-exp values and uses them to obtain gradients for $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. In addition, the gradient of each block log gate is accumulated by summing the score gradients over all tokens inside the corresponding selected block, producing a compact block-level gradient tensor that is propagated back to the selector. As a result, the implementation supports sparse scope gated attention training without storing the dense attention matrix.

Algorithm 3 Forward Pass of Fused SAS Kernel

```
def gated_scores(q, k, logG, tau, q_tile, kv_tile, B, sm_scale, self_block):
    S = sm_scale * matmul(q, k.T)
    b = block_id(kv_tile, B)
    if self_block:
        # current block: unit gate (log 1 = 0), causal mask
        return apply_causal_mask(S, q_tile, kv_tile)
    log_g = load(logG, q_tile, b) # [q_rows] per-query block gate
    active = log_g >= load(tau, q_tile) # selected blocks only (Top-K set)
    S = S + broadcast(log_g, kv_tile) # inject gate inside softmax
    return where(active, S, -inf) # mask out non-selected blocks

def soft_gate_fwd(Q, K, V, logG, tau, B, sm_scale):
    O, LSE = empty_output(Q, V), empty_lse(Q)

    for q_tile in tiles(Q):
        q = load(Q, q_tile)
        m, l = full(q.rows, -inf), zeros(q.rows)
        acc = zeros(q.rows, V.dim)

        # Phase 1: self block (causal) ; Phase 2: earlier blocks (sparse)
        for kv_tile in causal_tiles(K, q_tile):
            self_block = in_self_block(kv_tile, q_tile, B)
            if (not self_block) and no_query_selects(logG, tau, q_tile, kv_tile, B):
                continue # skip fully non-selected block
            k, v = load(K, kv_tile), load(V, kv_tile)
            S = gated_scores(q, k, logG, tau, q_tile, kv_tile, B, sm_scale,
                self_block)

            m_new = maximum(m, rowmax(S))
            P = exp(S - m_new[:, None])
            acc = acc * exp(m - m_new)[:, None] + matmul(P, v)
            l = l * exp(m - m_new) + rowsum(P)
            m = m_new

        O[q_tile] = acc / l[:, None]
        LSE[q_tile] = m + log(l)

    return O, LSE
```

Algorithm 4 Backward Pass of Fused SAS Kernel

```
def soft_gate_bwd(Q, K, V, logG, tau, O, dO, LSE, B, sm_scale):
    dQ, dK, dV = zeros_like(Q), zeros_like(K), zeros_like(V)
    dLogG, Delta = zeros_like(logG), rowsum(O * dO)

    # ---- Pass 1: dQ and dLogG (loop q outer, kv inner) ----
    for q_tile in tiles(Q):
        q, do = load(Q, q_tile), load(dO, q_tile)
        lse, delta = load(LSE, q_tile), load(Delta, q_tile)
        dq = zeros_like(q)

        for kv_tile in causal_tiles(K, q_tile):
            self_block = in_self_block(kv_tile, q_tile, B)
            if (not self_block) and no_query_selects(logG, tau, q_tile, kv_tile, B):
                continue
            k, v = load(K, kv_tile), load(V, kv_tile)
            S = gated_scores(q, k, logG, tau, q_tile, kv_tile, B, sm_scale,
                self_block)

            P = exp(S - lse[:, None])
            dS = P * (matmul(do, v.T) - delta[:, None])
            dq += sm_scale * matmul(dS, k)
            if not self_block:
                for b in blocks(kv_tile, B):
                    # gate gradient: sum dS over kv tokens in the selected block
                    dLogG[q_tile, b] += rowsum(dS[:, tokens(kv_tile, b)])

        dQ[q_tile] = dq

    # ---- Pass 2: dK and dV (loop kv outer, q inner) ----
    for kv_tile in tiles(K):
        k, v = load(K, kv_tile), load(V, kv_tile)
        dk, dv = zeros_like(k), zeros_like(v)

        for q_tile in attending_q_tiles(Q, kv_tile):
            self_block = in_self_block(kv_tile, q_tile, B)
            if (not self_block) and no_query_selects(logG, tau, q_tile, kv_tile, B):
                continue
            q, do = load(Q, q_tile), load(dO, q_tile)
            lse, delta = load(LSE, q_tile), load(Delta, q_tile)
            S = gated_scores(q, k, logG, tau, q_tile, kv_tile, B, sm_scale,
                self_block)

            P = exp(S - lse[:, None])
            dS = P * (matmul(do, v.T) - delta[:, None])
            dk += sm_scale * matmul(dS.T, q)
            dv += matmul(P.T, do)

        dK[kv_tile], dV[kv_tile] = dk, dv

    return dQ, dK, dV, dLogG
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe the paper's main contribution: training sparse block selection directly from the language modeling loss via differentiable context ranking.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of the work in Section E.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: This paper contains the full configuration of the experiments, and the experimental results can be reproduced.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The data and benchmarks used in this work are publicly available and cited. We release the code in `https://github.com/Tencent-Hunyuan/Simple-Attention-Sparsification`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We give the training and test details in Section 5.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: In the experiment section, we provide the standard deviation for each result.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the detail information of the computer resources like GPU, memory, time of execution.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impace in Section **??**.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cite the existing models, datasets, benchmarks, and software packages used in this work.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We have not decide to release any new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not include crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve human-subject research, crowdsourcing, user studies, surveys, or collection of personal data.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper uses open sourced LLMs as backbone models for sparse attention experiments, and their usage is described in the method and experimental setup.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.