SABER: STATE-AWARE BUDGET ESTIMATION AND ROUTING FOR SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention accelerates long-context Transformer inference by selecting a subset of key–value (KV) entries for each query. Existing adaptive sparse-attention methods typically rely on posterior pruning, leaving the upstream pre-selection budget as a manual design choice and introducing irregular computation patterns. We propose State-Aware Budget Estimation and Routing (**SABER**), an internal-probe-guided framework that unifies pre-allocation and online control of sparse-attention budgets. SABER predicts, for each query or generation state, the sufficiency of candidate budgets using lightweight probes over intermediate hidden states of the language model, and allocates the smallest budget that preserves task performance. This mechanism applies consistently across generation: it selects sufficient budgets from prompt states and, for long reasoning, continuously refines them as the generation trajectory evolves, enabling both one-shot allocation and trajectory-aware dynamic adjustment in a single framework. By performing input- and state-dependent allocation before and during decoding, SABER avoids over-provisioned attention and reduces unnecessary computation. Across long-context and reasoning benchmarks, SABER preserves accuracy relative to strong fixed-budget, top-p, and trainable sparse-attention baselines while reducing average selector budget by up to 67.6% and achieving up to 1.35$\times$ end-to-end speedup over state-of-the-art fixed-budget sparse attention methods.

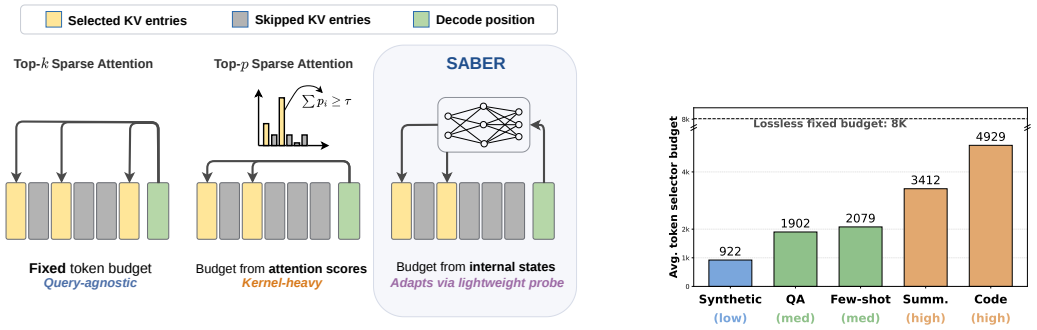
1 INTRODUCTION

Transformers deliver strong performance across language, vision, and multimodal tasks, but their dense self-attention mechanism scales quadratically with sequence length (Vaswani et al., 2017). This cost becomes especially restrictive in long-context inference and reasoning settings (Pope et al., 2023; Oviedo et al., 2026; Shi et al., 2024), where the model may need to process many tokens while only a subset of them are truly useful for each query. Sparse attention addresses this challenge by restricting each query to a smaller set of keys, reducing memory and compute while attempting to preserve the benefits of full attention (Sun et al., 2025; Beltagy et al., 2020; Tang et al., 2024b). However, deciding how to allocate attention budget across queries and over the course of generation remains a challenge.

A large body of prior sparse attention methods determine retained tokens using structured heuristics or attention scores. Fixed-budget methods such as top-k pruning keep the same number of keys for every query, which leads to regular computation patterns and straightforward implementations (Gupta et al., 2021; Tang et al., 2024b). Adaptive sparse attention variants such as top-p pruning instead allow the number of preserved keys to vary with the concentration of the attention distribution (Holtzman et al., 2020; Lin et al., 2025; Ni et al., 2026). These methods show that adaptive sparsity can yield better efficiency-accuracy tradeoffs than uniform sparsity. Yet, they still rely on attention scores alone, which capture per-token local relevance but fail to leverage higher-level signals about query difficulty and context requirements.

This limitation suggests that adaptive sparse attention needs a decision signal beyond attention scores. Figure 1 illustrates this perspective: rather than adapting only the retained tokens based on fixed heuristics or attention scores, SABER uses lightweight probes over intermediate hidden states to decide the upstream token budget before sparse attention is executed. Recent probing and self-knowledge studies show that language-model hidden states encode information that can be recovered by lightweight probes, including signals about answerability, correctness, and reasoning

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



(a) Sparse attention paradigms. **(b)** SABER's budget varies by task.

Figure 1: SABER reframes adaptive sparse attention as state-aware budget allocation rather than posterior token pruning. **(a)** top-k sparse attention applies a fixed token budget to all queries, while top-p sparse attention adapts the retained set according to attention distributions through kernel-level probability pruning. SABER allocates token budget based on lightweight probes over intermediate hidden states. **(b)** The average minimum sufficient token budget estimated by SABER varies substantially across task categories, while remaining far below the lossless fixed budget of 8K.

reliability before these properties are fully reflected in the generated output (Kadavath et al., 2022; Burns et al., 2023; Azaria and Mitchell, 2023; Liu et al., 2023; Zhang et al., 2025; Ghasemabadi and Niu, 2025). This suggests a state-aware view of sparse attention: hidden states can be used to estimate budget sufficiency, i.e., whether a candidate token budget is likely to preserve task performance. The key challenge is to turn this internal signal into a practical mechanism for allocating sparse-attention computation before decoding begins and as generation evolves.

In this paper, we propose State-Aware Budget Estimation and Routing (SABER), a probe-guided framework that treats sparse attention as a state-conditioned resource allocation problem. Instead of relying solely on attention scores or fixed heuristics, SABER leverages lightweight probes over intermediate hidden states to estimate, for each query or generation state, how much contextual support is required. Based on these predictions, SABER allocates the smallest sufficient attention budget and refines it as the model's internal state evolves, enabling both pre-allocation before decoding and trajectory-aware adjustment during generation. This design transforms sparse attention from scalar-score-based pruning into a unified, state-aware budget allocation mechanism that adapts to varying input difficulty and reasoning dynamics. Our main contributions are summarized as follows:

- **State-aware budget allocation via internal-signal-based sufficiency estimation.** We introduce a formulation of sparse attention as a state-conditioned resource allocation problem, where the goal is to select the smallest sufficient attention budget for each query or generation state. We show that intermediate hidden states during LLM generation encode predictive signals of budget sufficiency, and can be used to estimate whether a candidate budget preserves task performance, enabling upstream budget decisions beyond attention-score-based heuristics.
- **A unified framework for upstream allocation and online budget control.** We design a two-stage mechanism that maps sufficiency estimates to actionable budget decisions, combining prefill-time allocation from prompt states with a lightweight controller for trajectory-aware adjustment during generation. This enables consistent and efficient budget control without over-provisioned candidate sets or posterior pruning.
- **Substantial efficiency gains with minimal overhead across generation regimes.** On LongBench, SABER reduces the average attention token budget by 67.6% with only 0.5% additional latency using an 18M-parameter prefill-time estimator. Across long-reasoning benchmarks, SABER reduces the average token budget by 51.3% while incurring only 0.32% latency overhead via a lightweight 14.3M-parameter online router, preserving or improving accuracy relative to strong sparse-attention baselines.

2 Related Work

2.1 Sparse Attention

Early methods use fixed sparse structures, such as sparse factorizations and sliding-window attention with global tokens (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Recent inference-time methods instead select relevant tokens, blocks, or KV-cache pages using query-dependent scores, learned gates, or optimized sparse kernels (Correia et al., 2019; Gupta et al., 2021; Tang et al., 2024b; Jiang et al., 2024; Gao et al., 2025). Top-p-style pruning further adapts the retained set according to cumulative attention mass after an upstream candidate set has been selected (Lin et al., 2025; Ni et al., 2026; Zhang et al., 2026). Tactic (Zhu et al., 2026) further performs training-free upstream budget adaptation using attention-mass statistics and distribution fitting. Existing adaptive methods remain driven by attention statistics. In contrast, SABER decouples budget allocation from attention scores by predicting budget sufficiency from intermediate model states before engaging the sparse-attention backend.

Native Sparse Attention trains sparse patterns end-to-end with the model, combining coarse token compression and fine-grained token selection (Yuan et al., 2025; Liu et al., 2025; DeepSeek-AI et al., 2026). Such methods can be effective when sparsity is integrated during pretraining, but require modifying and training the backbone architecture. SABER is orthogonal: it acts as a lightweight inference-time budget allocator for existing LLMs and sparse-attention backends.

2.2 Head-Wise Budget Allocation and KV Cache Compression

Recent KV-cache compression methods allocate budgets non-uniformly across attention heads. Ada-KV adapts eviction budgets across heads, while RazorAttention and DuoAttention identify retrieval heads that require larger or full KV caches and compress the remaining heads more aggressively (Feng et al., 2024; Tang et al., 2024a; Xiao et al., 2024). These methods optimize head-wise cache storage or eviction, whereas SABER allocates the upstream token budget from the model state before sparse attention is executed and routes it online during long reasoning.

2.3 Internal Signals for Verification and Uncertainty

A growing body of work shows that language-model hidden states encode information about knowledge, answerability, correctness, and reliability. Linear probes recover linearly decodable information from intermediate representations (Alain and Bengio, 2016; Belrose et al., 2023), and recent internal-signal methods use hidden states to estimate whether a model knows an answer, extract latent knowledge, analyze truthfulness-related representations, or predict reasoning failures (Kadavath et al., 2022; Burns et al., 2023; Azaria and Mitchell, 2023; Liu et al., 2023; 2024; Zhang et al., 2025; Ghasemabadi and Niu, 2025). Output-based uncertainty measures such as semantic entropy provide a complementary view from generated responses (Kuhn et al., 2023; Farquhar et al., 2024). SABER uses this internal-signal perspective for computation control: instead of using probes only for diagnosis or verification, it estimates whether a candidate sparse-attention budget is sufficient and uses that signal to allocate budget in long-context inference and long reasoning.

2.4 Other Inference Optimizations

Beyond sparse attention, LLM inference can be accelerated along several orthogonal dimensions. Quantization reduces numerical precision and memory bandwidth requirements (Frantar et al., 2022; Lin et al., 2024; Xiao et al., 2023), while speculative decoding and multi-token prediction reduce serial decoding steps by proposing or predicting future tokens and verifying them in parallel (Leviathan et al., 2023; Cai et al., 2024). System-level kernels and serving runtimes improve hardware utilization and KV-cache management, including IO-aware exact attention, long-context decode kernels, and paged KV-cache management (Dao et al., 2022; Kwon et al., 2023). Linear-attention variants reformulate attention to scale linearly, rather than quadratically, with sequence length (Choromanski et al., 2020; Katharopoulos et al., 2020). These methods reduce the cost of executing model computation, whereas SABER controls how much sparse-attention computation is allocated at inference time; the two directions are complementary.

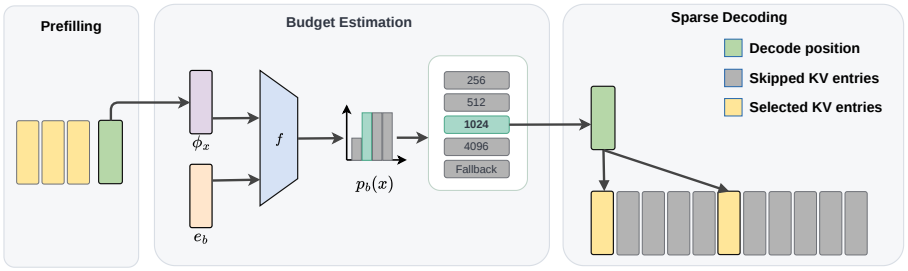


Figure 2: **Prefill-time budget estimation.** Given a long-context prompt, SABER first performs prompt prefill and extracts a prompt-state feature ϕ_x from the final prompt token. The feature ϕ_x is combined with a learned budget embedding e_b and fed into a budget-sufficiency probe f, which estimates the sufficiency score $p_b(x)$ for each candidate selector budget. Based on these scores, a pre-selection budget K_{pre} is selected before decoding begins. The selected budget is then passed to sparse attention, where the current query attends only to the top-K_{pre} prompt tokens.

3 METHOD

We formulate sparse decoding as a *selector-budget allocation* problem. Let $\mathcal{B} = \{B_1 < \cdots < B_M\}$ denote an ordered ladder of pre-selection budgets. We learn to assign an upstream token selector budget from $\mathcal{B}$ based on the current prompt or reasoning state before sparse attention is executed.

Our framework has three components. Section 3.1 describes how we collect budget-annotated trajectories and construct supervision signals for budget allocation. Section 3.2 instantiates the framework for long-context generation, where a single budget is selected after prompt prefill. Section 3.3 extends the framework to long reasoning, where the required budget may change as intermediate reasoning tokens are generated, requiring online budget adjustment. Detailed architectures and training procedures are provided in Appendix A.1.

3.1 DATA COLLECTION

We construct supervision for two regimes: long-context generation and long reasoning. Both settings require budget-conditioned labels, but differ in how correctness is obtained and how the input state is defined.

Long-context data. For long-context tasks, we run the sparse-attention model under different selector budgets $b \in \mathcal{B} \setminus \{B_M\}$, where B_M denotes the largest fallback budget. Then we obtain a label indicating whether budget b is sufficient to preserve the task outcome. This gives training examples of the form $(\text{question}, \text{context}, b, y^b)$, where $y^b \in \{0, 1\}$ denotes the budget-sufficiency label.

Long-reasoning data. We run the model under each budget $b \in \mathcal{B}$ and obtain decode trajectory c and correctness labels y^b by comparing the final answer with the ground-truth solution. This produces budget-conditioned rollout examples (c, b, y^b).

To train the online reasoning-state estimator, we further split each rollout according to the same inspection interval used at inference time. Given a generated reasoning trajectory divided into chunks $(c_1, c_2, \ldots, c_n)$, we construct prefix states

$$(c_1, b, y^b), \quad (c_1 + c_2, b, y^b), \quad \ldots, \quad (c_1 + \cdots + c_n, b, y^b).$$

These prefix-level examples are used to train the predictor and controller for online budget allocation.

3.2 PREFILL-TIME BUDGET ESTIMATION FOR LONG CONTEXT

We instantiate selector-budget allocation for long-context generation. Given a prompt x, the goal is to choose an upstream budget before sparse decoding begins:

$$K_{\text{pre}}(x) \in \mathcal{B}. \tag{1}$$

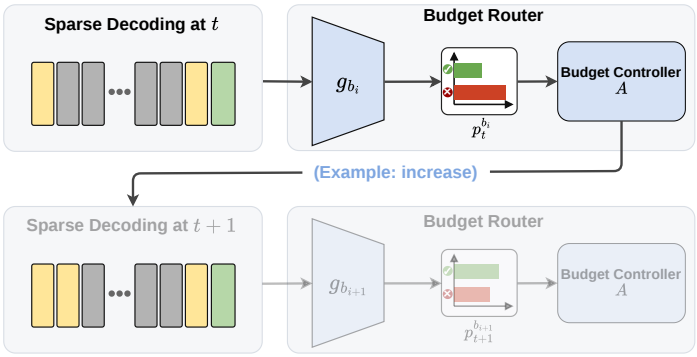


Figure 3: **Online budget routing for long reasoning.** Given the current sparse decoding state at interval t, the trajectory predictor g_{b_i} estimates budget sufficiency $p_t^{b_i}$, and the budget controller routes the token budget for the next decoding interval $t+1$. The faded bottom row illustrates the resulting state at $t+1$; the shown example increases the budget as the reasoning trajectory evolves.

The objective is to select the smallest budget that is likely to preserve enough long-context evidence for the current prompt, rather than using a fixed global budget for all inputs.

Figure 2 illustrates our prefill-time allocation pipeline. After prompt prefill, we extract a prompt-state feature ϕ_x from the final prompt token and use it to predict budget sufficiency before sparse decoding is executed. For each candidate budget b, a budget-aware probe combines ϕ_x with a learned budget embedding e_b and estimates

$$p_b(x) = f(\phi_x, e_b). \tag{2}$$

where $p_b(x)$ is the predicted probability that budget b is sufficient for prompt x. The probe is trained with binary sufficiency labels indicating whether each budget preserves the task outcome.

At inference time, we select the budget K_{pre} based on:

$$K_{\text{pre}}(x) = \min\Big(\{b \in \mathcal{B} : p_b(x) \geq \tau\} \cup \{B_M\}\Big). \tag{3}$$

The selected budget is passed directly to the sparse-attention selector, and decoding continues from the same prompt prefill state. This yields a one-shot prefill-time budget estimation mechanism: the budget is determined before sparse decoding, avoiding both a fixed global budget and posterior pruning over an over-provisioned candidate set.

3.3 Online Sparse-Attention Budget Routing for Long Reasoning

Prefill-time allocation assigns a single token budget before generation starts, which is insufficient for long reasoning. As intermediate reasoning tokens are generated, the prefix itself becomes new context, so budget sufficiency becomes trajectory-dependent. We therefore formulate long-reasoning budget allocation as online control over an ordered budget ladder $\mathcal{B} = \{B_1 < \cdots < B_M\}$. At inspection interval t, given prefix P_t and current budget $b_t \in \mathcal{B}$, the next budget is selected from a local neighborhood:

$$\mathcal{N}(b_t) = \{\text{down}(b_t),\ b_t,\ \text{up}(b_t)\} \cap \mathcal{B}. \tag{4}$$

Figure 3 illustrates this state-aware budget routing pipeline: the trajectory predictor estimates budget sufficiency from the current reasoning state, and the controller updates the upstream token selector budget along the ordered budget ladder as generation evolves.

Trajectory predictor. We use the hidden sequence H_t^b generated under budget b as the observable reasoning state of the current prefix. Since different sparse-attention budgets may induce different hidden-state distributions, we use a budget-specific predictor g_b for each budget $b \in \mathcal{B}$. Given H_t^b, the predictor outputs a sufficiency logit:

$$z_t^b = g_b(H_t^b), \qquad p_t^b = \sigma(z_t^b), \tag{5}$$

where p_t^b estimates the probability that the trajectory under budget b will eventually produce the correct final answer.

Budget controller. We convert predicted budget sufficiency into a cost-aware utility. For each budget b, we define the cost-aware utility U and advantage Δ as

$$U_t(b) = p_t^b - \lambda C(b), \qquad \Delta_t^{\text{down}} = U_t(\text{down}(b_t)) - U_t(b_t), \qquad \Delta_t^{\text{up}} = U_t(\text{up}(b_t)) - U_t(b_t), \quad (6)$$

where $C(b)$ is the normalized sparse-attention cost and λ controls the accuracy–cost trade-off.

At each inspection interval, we construct a controller state

$$x_t = \left[F_t^{b_t},\, \mathcal{H}_t(z),\, \psi_t\right], \quad (7)$$

where $F_t^{b_t}$ is the predictor feature, $\mathcal{H}_t(z)$ summarizes the recent logit history, and ψ_t contains lightweight decoding and budget metadata. The controller then predicts local utility advantages:

$$(\widehat{\Delta}_t^{\text{down}}, \widehat{\Delta}_t^{\text{up}}) = A_\theta(x_t). \quad (8)$$

At inference time, only the current-budget trajectory is observed, so the utilities of neighboring budgets cannot be directly evaluated without replaying the prefix. We first anchor the current utility using the observed predictor output:

$$\widehat{U}_t(b_t) = p_t^{b_t} - \lambda C(b_t). \quad (9)$$

The neighboring utilities are then estimated by adding the controller-predicted advantages:

$$\widehat{U}_t(\text{down}(b_t)) = \widehat{U}_t(b_t) + \widehat{\Delta}_t^{\text{down}}, \qquad \widehat{U}_t(\text{up}(b_t)) = \widehat{U}_t(b_t) + \widehat{\Delta}_t^{\text{up}}. \quad (10)$$

The next budget is selected as

$$b_{t+1} = \arg\max_{b \in \mathcal{N}(b_t)} \widehat{U}_t(b). \quad (11)$$

This allows the controller to increase the sparse-attention budget when the current reasoning trajectory becomes unreliable, and to return to lower-cost budgets when the trajectory remains stable.

4 Experiments

4.1 Experiment Settings

Training Data. For long-context tasks, we sample question-answering (QA) examples from single-document QA tasks in SCROLLS (Shaham et al., 2022). We restrict the sampled examples to contexts between 5K and 40K tokens. For long reasoning, we sample 7k math problems from DAPO-Math-17K (Yu et al., 2025). For each problem, we run the model under each budget in the reasoning budget ladder and label the rollout by comparing the extracted final answer with the ground-truth solution. We then split each trajectory using the same 512-token inspection interval as inference, producing prefix-level training examples for the budget-matched trajectory predictor and the online budget controller.

Benchmark and models. We evaluate SABER on two types of tasks: long-context tasks and reasoning tasks. For long-context tasks, we evaluate on the 13-task LongBench suite (Bai et al., 2024): including QA, summarization, few-shot learning, synthetic and coding. We evaluate long-context tasks on the representative open-source long-context LLM: LLaMA3.1-8B (Grattafiori et al., 2024), with context length up to 128K tokens. For reasoning tasks, we evaluate on AIME24, AIME25 (Mathematical Association of America) and MATH-500 (Lightman et al., 2023; Hendrycks et al., 2021) and select Qwen3-4B as the reasoning model with a 32K context window. For AIME24 and AIME25, we report mean single-sample accuracy over 16 independent generations to reduce variance on these 30-problem benchmarks. For MATH-500, we use one generation per question due to its larger problem count.

Baselines. We compare SABER with representative top-k, top-p, and trainable block-sparse attention methods. Quest (Tang et al., 2024b) is a query-aware sparse-attention method that selects top-k critical KV-cache pages. Twilight (Lin et al., 2025) applies top-p pruning after top-k pre-selection to remove low-importance tokens. These methods provide strong baselines for long-context sparse attention; however, their reported evaluations primarily focus on long-context tasks and do not

Table 1: Quantitative evaluation results on the 13-task LongBench suite (Bai et al., 2024), aggregated by task type. All metrics are task scores where higher is better. Category averages are computed from the full per-task results by grouping tasks into single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code. Average budget per category for SABER is provided in parentheses. In the final **Avg.** column, the best sparse-attention result is shown in **bold** and the second-best result is shown in *italics*.

Method	Budget Setting	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synthetic	Code	Avg.
Full	Dense	43.04	42.13	29.21	91.18	99.50	59.97	50.29
Fixed-budget Sparse Attention								
Quest	256	28.96	22.22	24.74	65.34	95.00	47.48	37.16
Quest	1024	38.53	35.50	27.99	81.94	99.50	56.87	46.25
Quest	4096	42.55	39.49	28.48	90.96	99.50	58.76	49.20
Quest	8192	42.43	41.85	28.77	91.61	100.00	58.11	*49.77*
Adaptive-budget Sparse Attention								
Twilight	8192 + top-p	40.83	40.92	28.87	91.25	100.00	60.80	49.59
SABER	2654	41.73 (2.6K)	42.18 (1.2K)	29.27 (3.4K)	91.09 (2.1K)	99.50 (0.96K)	58.70 (4.9K)	**49.81**
SABER+Twi.	2025†	40.97 (2.0K)	40.57 (1.0K)	28.90 (2.5K)	89.71 (1.6K)	99.50 (0.90K)	61.40 (3.6K)	49.49

† For SABER+Twi., the reported value is an attention-cost-equivalent budget.

explicitly study sparse attention for long reasoning. We also adapt SeerAttention-R (Gao et al., 2025), a trainable sparse-attention method that leverages a gating network to identify critical KV blocks for long reasoning. For the top-p variant of SeerAttention-R, we use the efficient top-p kernel from Twilight and set the pruning threshold to $p_{\text{prune}} = 0.95$ for a conservative accuracy-preserving configuration. Unless otherwise specified, all baselines follow their default configurations.

Implementation details. For long-context tasks, SABER performs prefill-time budget allocation over $\mathcal{B} = \{256, 512, 1024, 4096, 8192\}$, where 8192 is the fallback budget. We use a fixed sufficiency threshold $\tau = 0.65$ for all LongBench tasks. For reasoning tasks, SABER uses SeerAttention-R as the sparse-attention backend and performs online budget routing over $\{256, 512, 1024, 2048, 4096\}$. The online router starts from budget 256, inspects the generated prefix every 512 tokens, and uses cost coefficient $\lambda = 0.2$ in the utility function. Unless otherwise specified, all reasoning methods use the same default decoding configuration of the underlying model/evaluation pipeline, including the same sampling hyperparameters and a maximum generation length of 12K tokens.

4.2 Long-context Evaluation

Table 1 reports the grouped LongBench results. SABER achieves the best overall average among all sparse-attention baselines, while using an average pre-selection budget of only 2654 tokens, corresponding to a 67.6% reduction from the 8192 lossless budget setting. Its accuracy is also nearly lossless compared with full attention. Beyond the aggregate score, the category breakdown shows that SABER does not simply overfit to the training distribution. Although the budget estimator is trained using **single-document QA-style data**, SABER exhibits strong **cross-domain generalization**, remaining competitive across all LongBench categories and achieving particularly strong results on Multi-Doc QA and Summarization, despite the substantial shift in task format and information aggregation requirements. This suggests that the learned prefill-time budget signal captures a transferable notion of budget sufficiency rather than a task-specific shortcut.

We also include SABER+Twi. to test whether upstream budget estimation can be combined with posterior top-p pruning. Motivated by the observation that top-p pruning is most reliable when the upstream selector budget is sufficiently conservative, SABER+Twi. applies top-p only to high-budget branches ($K_{\text{pre}} \in \{4096, 8192\}$), where the candidate set is more conservative, while keeping pure top-k selection for low-budget branches ($K_{\text{pre}} \in \{256, 512, 1024\}$). This hybrid variant further reduces the attention-cost-equivalent budget while maintaining near-lossless performance, achieving a higher average score than Twilight-4k with lower computation.

Figure 1(b) shows the average budgets selected by SABER across LongBench task families. SABER assigns clearly different budgets to different task types: Synthetic tasks receive the smallest budgets, QA and few-shot tasks receive moderate budgets, while summarization and code receive substantially

larger budgets. This ordering is consistent with the budget demand of different LongBench task families. The Synthetic category is represented by PassageRetrieval-en, which is closer to localized retrieval: the model needs to identify the paragraph corresponding to a given summary, so the relevant evidence is often concentrated. QA and few-shot tasks receive moderate budgets, since the question or in-context examples constrain the relevant evidence regions. In contrast, summarization requires aggregating distributed information across long documents, and code completion is sensitive to long-range dependencies such as symbols, definitions, and cross-file context. Together, the accuracy results and allocation behavior show that SABER learns to allocate sparse-attention budget in a task-adaptive and transferable manner.

4.3 Long-reasoning Evaluation

Table 2 reports the long-reasoning results on AIME24/25 and MATH-500. SABER achieves the highest accuracy on AIME24 and MATH-500, outperforming the strongest fixed-budget SeerAttention baseline with up to a 51.3% lower average budget, while remaining close to the best fixed-budget result on AIME25. Meanwhile, top-p's weaker performance on AIME24/25 suggests that posterior top-p pruning alone is not sufficient for robust long-reasoning adaptation. By contrast, SABER adapts the selector budget according to the evolving reasoning trajectory, yielding a stronger accuracy–efficiency trade-off than either fixed-budget sparse attention or posterior top-p pruning. Unlike the long-context setting, we do not include a SABER+Twilight hybrid for long-reasoning, since the top-p variants do not show consistent gains over fixed-budget SeerAttention on the harder AIME benchmarks.

Table 2: Quantitative evaluation results on AIME24/25 and MATH-500, reported as accuracy (%). The best and second-best sparse-attention result in each benchmark column is shown in **bold** and *italics*. For SABER, Budget denotes the macro-average across the three benchmarks.

Method	Budget Setting	**AIME24**	**AIME25**	**MATH-500**
Full Attention	Dense	54.58	43.13	85.2
Fixed-budget Sparse Attention				
SeerAttention	1024	51.25	38.54	76.0
SeerAttention	2048	53.33	43.75	81.0
SeerAttention	4096	*56.67*	**45.21**	82.0
SeerAttention	8192	55.00	44.58	82.0
Adaptive-budget Sparse Attention				
Twilight	4096 + top-p	40.83	39.38	81.4
Twilight	8192 + top-p	44.58	41.88	*82.4*
SABER	1995	**58.54**	*45.00*	**85.2**

The budgets selected by SABER further show its adaptive behavior across long-reasoning workloads. SABER uses larger average budgets on the more challenging competition-level AIME24 and AIME25 benchmarks, with average token budgets of 2445 and 2497, respectively, while using an average budget of only 1042 on the relatively easier MATH-500 benchmark. This confirms that SABER does not behave like a fixed-budget method, but allocates sparse-attention budget according to the estimated reasoning demand of each workload.

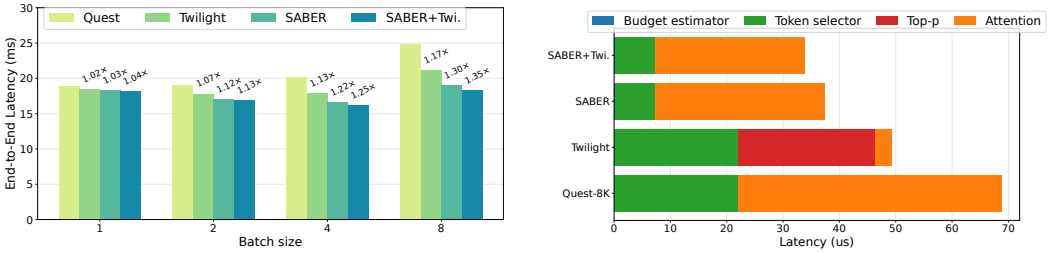
4.4 Efficiency

For adaptive methods, reported budgets and latencies are averaged over the budgets actually selected during inference; SABER executes only one selected budget at each decision point. The exact definition is detailed in Appendix A.2.

Table 3: End-to-end runtime on 12k-token long-reasoning generation.

Method	**Budget**	**Speedup**	**Runtime**
Full Attention	Dense	1.00×	215.31s
Top-k Sparse Attention	4096	3.08×	69.92s
SABER + router	1995	3.81×	56.27s (+0.18s router)

Figure 4 analyzes the efficiency of SABER in the long-context setting. Figure 4(a) shows the estimated end-to-end decoding latency at 16K context length under different batch sizes, where speedups are normalized to the fixed Quest-8K setting. SABER consistently reduces latency over other baseline methods, and the gain becomes larger at higher batch sizes where attention-side computation occupies a larger fraction of end-to-end decoding. Figure 4(b) further decomposes the self-attention-side latency. Compared with Quest-8K and Twilight, SABER reduces the cost by moving the budget decision upstream and selecting a smaller token budget before sparse attention is



(a) (b)

Figure 4: (a) End-to-end decoding latency at 16k context length across batch sizes, with annotations showing speedup over Quest with 8k token budget. (b) Self-attention time breakdown for long-context at 16k context length; the budget-estimator overhead of SABER is only $0.205\,\mu s$ and is therefore barely visible.

Table 4: Ablation of τ on LongBench.

τ	Score	Budget
0.50	49.48	1819.35
0.65	49.81	2654.18
0.80	50.30	4794.18

Table 5: Ablation of checkpoint interval.

Interval	AIME24	AIME25	MATH	Budget
256	55.00	44.58	86.2	2210
512	58.54	45.00	85.2	1995
1024	56.67	43.33	84.4	1547

executed. The additional budget-estimator overhead is only $0.2\,\mu s$ and is barely visible, indicating that the adaptive selector itself introduces negligible runtime overhead.

Table 3 reports the end-to-end runtime for 12K-token long-reasoning generation. SABER lowers the decoding time from 69.92s to 56.27s, corresponding to a $1.24\times$ speedup over the fixed-budget long-reasoning setting. The online budget router adds only 0.18s over the entire generation, contributing about 0.32% of the total SABER runtime. Thus, the cost of trajectory-aware budget routing is negligible compared with decoding. Additional runtime details are provided in Appendix A.2.

4.5 Ablation

We ablate two key design choices in SABER: the sufficiency threshold for prefill-time budget estimation and the checkpoint interval for online budget routing.

Table 4 studies the sufficiency threshold τ for prefill-time budget estimation. Increasing τ improves the score by selecting more conservative budgets, but also substantially increases the average budget. We use $\tau = 0.65$ as the default because it provides a strong accuracy–budget trade-off.

Table 5 studies the checkpoint interval for online budget routing. A shorter interval allows more frequent budget adjustment but tends to use a higher average budget, while a longer interval reduces budget but reacts less frequently to changes in the reasoning trajectory. We use 512 tokens as the default routing interval as it achieves the best overall balance.

5 Conclusion

In this paper, we introduced SABER, a state-aware framework for adaptive sparse-attention budget allocation. Instead of relying only on attention scores or fixed Top-k budgets, SABER uses lightweight probes over internal representations to estimate how much sparse-attention budget is needed before the budget is spent. For long-context inference, SABER performs one-shot prefill-time budget estimation from the final prompt state; for long reasoning, SABER uses an online budget router to adjust the selector budget as the generated reasoning trajectory evolves. Across long-context and reasoning benchmarks, SABER preserves strong task performance while substantially reducing computation, including up to a 67.6% reduction in average token budget and up to $1.35\times$ end-to-end latency speedup. Our work highlights the value of moving adaptive sparse attention beyond score-only pruning: internal model states provide useful signals for deciding not only which tokens to attend to, but also how much attention budget should be allocated at each stage of inference.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing author-designed methods. We have not used generative AI tools to develop conceptual frameworks, formulate hypotheses, design research methodology or experiments, formulate or prove mathematical claims, or interpret experimental results, and the remaining required-disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools for literature search and survey, software development and debugging, and drafting and editing portions of the manuscript. We have reviewed all AI-assisted work. AI-assisted code was tested for correctness, and AI-assisted literature and manuscript content was verified by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016. doi: 10.48550/arXiv.1610.01644. URL https://arxiv.org/abs/1610.01644.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 967–976, 2023. doi: 10.18653/v1/2023.findings-emnlp.68. URL https://aclanthology.org/2023.findings-emnlp.68/.

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

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

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

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ETKGuby0hcs.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. doi: 10.48550/arXiv.1904.10509. URL https://arxiv.org/abs/1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, 2019. doi: 10.18653/v1/D19-1223. URL https://aclanthology.org/D19-1223/.

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.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, Hayden Kwok-Hay So, Yu Hua, Ting Cao, Fan Yang, and Mao Yang. Seerattention-r: Sparse attention adaptation for long reasoning. *arXiv preprint arXiv:2506.08889*, 2025. doi: 10.48550/arXiv.2506.08889. URL https://arxiv.org/abs/2506.08889.

Amirhosein Ghasemabadi and Di Niu. Can LLMs predict their own failures? self-awareness via internal circuits. *arXiv preprint arXiv:2512.20578*, 2025. doi: 10.48550/arXiv.2512.20578. URL https://arxiv.org/abs/2512.20578.

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

Ankit Gupta, Guy Dar, Shaya Goodman, David Ciprut, and Jonathan Berant. Memory-efficient transformers via top-k attention. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, pages 39–52, 2021. doi: 10.18653/v1/2021.sustainlp-1.5. URL https://aclanthology.org/2021.sustainlp-1.5/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Joao P Hespanha, Daniel Liberzon, and A Stephen Morse. Hysteresis-based switching algorithms for supervisory control of uncertain systems. *Automatica*, 39(2):263–272, 2003.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2020. URL https://arxiv.org/abs/1904.09751.

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.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022. doi: 10.48550/arXiv.2207.05221. URL https://arxiv.org/abs/2207.05221.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*, 2023.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Jiang, Shauna Karavani, Pranav Pande, Mohammad Khan, et al. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

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. *arXiv preprint arXiv:2502.02770*, 2025. doi: 10.48550/arXiv.2502.02770. URL https://arxiv.org/abs/2502.02770.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

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.

Junteng Liu, Shiqi Chen, Yu Cheng, and Junxian He. On the universal truthfulness hyperplane inside LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18199–18224, 2024. doi: 10.18653/v1/2024.emnlp-main.1012. URL https://aclanthology.org/2024.emnlp-main.1012/.

Kevin Liu, Stephen Casper, Dylan Hadfield-Menell, and Jacob Andreas. Cognitive dissonance: Why do language model outputs disagree with internal representations of truthfulness? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4791–4797, 2023. doi: 10.18653/v1/2023.emnlp-main.291. URL https://aclanthology.org/2023.emnlp-main.291/.

Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. *arXiv preprint arXiv:2310.06824*, 2023.

Mathematical Association of America. American invitational mathematics examination (aime). https://maa.org/math-competitions/american-invitational-mathematics-examination-aime. Accessed: 2026-04-28.

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

Felipe Oviedo, Fiodar Kazhamiaka, Esha Choukse, Allen Kim, Amy Luers, Melanie Nakagawa, Ricardo Bianchini, and Juan M Lavista Ferres. Energy use of ai inference, efficiency pathways, and test-time scaling. *Joule*, 2026.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, et al. Scrolls: Standardized comparison over long language sequences. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 12007–12021, 2022.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

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.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024a.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. *arXiv preprint arXiv:2406.10774*, 2024b. doi: 10.48550/arXiv.2406.10774. URL https://arxiv.org/abs/2406.10774.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR, 2023.

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. *arXiv preprint arXiv:2410.10819*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

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.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025.

Jintao Zhang, Kai Jiang, Chendong Xiang, Weiqi Feng, Yuezhou Hu, Haocheng Xi, Jianfei Chen, and Jun Zhu. Spargeattention2: Trainable sparse attention via hybrid top-k+ top-p masking and distillation fine-tuning. *arXiv preprint arXiv:2602.13515*, 2026.

Kan Zhu, Tian Tang, Qinyu Xu, Zhan Jin, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, et al. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms. In *International Conference on Learning Representations*, volume 2026, pages 30626–30644, 2026.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Appendix

A.1 Implementation Details

Long-context budget-sufficiency predictor. For long-context generation, we implement the prefill-time estimator as a budget-aware binary predictor on top of a frozen Llama-3.1-8B-Instruct model with Quest sparse attention enabled. The candidate budgets are $\{256, 512, 1024, 4096\}$, with 8192 used as the fallback budget. The Quest chunk size is set to 16.

The predictor uses only the final prompt-token hidden state. Specifically, after prompt prefill, we extract hidden states from decoder **24 and 28** of Llama-3.1-8B-Instruct, following the conclusion in (Marks and Tegmark, 2023; Zou et al., 2023). Each hidden state has dimension 4096, and the two states are concatenated to form an 8192-dimensional prompt feature:

$$\phi_x = [h_{24}(x); h_{28}(x)] \in \mathbb{R}^{8192}.$$

We parameterize the predictor as a Multi-Layer Perceptron (MLP) that jointly processes the input feature and a candidate budget. Specifically, the 8192-dimensional input feature ϕ_x is first projected to a 2048-dimensional representation through a linear layer, followed by LayerNorm, a GELU activation, and a 10% dropout. Concurrently, the candidate budget is mapped into a 128-dimensional continuous space via a learned embedding layer. The projected feature and the budget embedding are then concatenated to form a 2176-dimensional joint vector. Finally, this vector is fed into a two-layer MLP with hidden sizes of 512 and 128 (each equipped with LayerNorm, GELU, and 10% dropout) to output the final 2-dimensional score. The two output logits correspond to fail and solve labels. The solve probability is used as the budget-sufficiency score. This predictor contains 17.98M trainable parameters when the input feature dimension is 8192.

Long-context predictor training recipe. The long-context predictor is trained from fixed-budget Quest generations. For each prompt and candidate budget b, the binary target indicates whether generation under budget b solves the example. We use Qwen3-30B-A3B (Yang et al., 2025) to generate the binary labels. The training cache contains 300 examples first solved at each of the four candidate budgets, 300 additional unsolved examples, and 300 additional monotonicity-violation examples. After expanding each example over the four candidate budgets, the final split contains 7200 training instances and 2000 evaluation instances.

We train the predictor with cross entropy for 20 epochs using batch size 128, learning rate 10^{-4}, weight decay 0, dropout 0.10, and shuffled training examples. We use a cosine learning-rate schedule with warmup ratio 0.1 and minimum learning-rate ratio 0.1. The best checkpoint is selected by evaluation accuracy. At inference time, we use threshold $\tau = 0.65$: the smallest candidate budget whose solve probability exceeds the threshold is selected, and 8192 is used if no candidate passes.

Long-reasoning completion-correctness predictors. For long-reasoning experiments, we train a separate completion-correctness predictor for each budget in

$$\{256, 512, 1024, 2048, 4096\}.$$

All predictors share the same architecture but are trained on hidden states generated under their corresponding budgets. Each predictor receives a variable-length hidden-state sequence

$$H_t^b \in \mathbb{R}^{S_t \times H},$$

where S_t is the prefix length at inspection step t. Since prefix lengths vary across examples, the implementation supports ragged batches by forwarding examples independently when necessary.

We adopt the hidden circuit encoder architecture in (Ghasemabadi and Niu, 2025) as the backbone of our predictor: The hidden sequence H_t^b is first processed via Layer Normalization and linearly projected to 192 dimensions. To ensure a consistent length, the sequence is then downsampled to a fixed size of 192 tokens using Adaptive Average Pooling. Next, we apply a multi-branch convolutional module comprising three parallel depthwise 1D convolutions with a kernel size of 5 and dilations of 1, 2, and 4, respectively. A learned softmax gate dynamically mixes the outputs of these branches, which is further refined by a Squeeze-and-Excitation (SE) style 1D gate and a learned positional embedding.

Following the convolutional feature extraction, the sequence is processed by three Set Attention Blocks (SABs) with four attention heads. To aggregate the sequence into a fixed-size representation, we apply Pooling by Multihead Attention (PMA) using three learned seed vectors. The resulting output is flattened and passed through a Multi-Layer Perceptron (MLP) with a hidden dimension of 384, utilizing a GELU activation and 10% dropout, yielding a 256-dimensional predictor feature F_t^b. Finally, F_t^b is fed into a classification head—a standard MLP with hidden dimensions of 384 and 128, equipped with LayerNorm, GELU activations, and 10% dropout—to generate the final 2-dimensional output. We use the solve-fail logit margin

$$z_t^b = \ell_{\text{solve}} - \ell_{\text{fail}}$$

as the scalar predictor score, and the 256-dimensional penultimate feature F_t^b is passed to the controller. This budget-specific predictor has 2.863M trainable parameters (14.31M in total for 5 different budgets).

Long-reasoning predictor training recipe. For each budget b, the predictor is trained using the correctness label of the rollout generated under that budget. Specifically, $y^b = 1$ if the corresponding rollout produces the correct final answer, and $y^b = 0$ otherwise. Each budget-specific predictor is trained with standard cross-entropy loss over the two output logits:

$$\mathcal{L}_{\text{pred}}^b = \text{CE}\left(\ell^b, y^b\right),$$

where $\ell^b = \{\ell_{\text{solve}}, \ell_{\text{fail}}\} \in \mathbb{R}^2$ denotes the fail/solve logits predicted by the budget-specific predictor g_b, and $y^b \in \{0, 1\}$ is the binary label.

We train each predictor for 10 epochs with batch size 2, learning rate 10^{-4}, weight decay 0, and gradient clipping at 1.0. We use a cosine learning-rate schedule with warmup ratio 0.1 and minimum learning-rate ratio 0.1. During training, we use accuracy to select the best checkpoint.

Controller input construction. The controller is trained offline from prefix-replay features. Generated completions are split into prefix checkpoints using a fixed stride of 512 tokens. For each prefix checkpoint and each budget, we replay the prefix through the corresponding sparse-attention model, run the budget-specific predictor, and store the predictor feature F_t^b, logit margin z_t^b, and teacher correctness probability. This produces a teacher probability vector over all budgets for the same prefix:

$$p_t^{\text{teacher}} = [p_t^{256}, p_t^{512}, p_t^{1024}, p_t^{2048}, p_t^{4096}].$$

We also store the first and second finite differences of the logit margin over prefix checkpoints, denoted dz_t and d^2z_t. The scalar features z_t, dz_t, and d^2z_t are normalized per budget, while prefix length is normalized globally.

Controller Architecture. The controller is designed as a lightweight Multi-Layer Perceptron (MLP) featuring a shared trunk and two task-specific output heads. At step t with the current budget b_t, the input vector x_t is constructed by concatenating multiple signals: the predictor feature $F_t^{b_t}$, the dynamic state variable $z_t^{b_t}$ along with its first and second derivatives ($dz_t^{b_t}$ and $d^2z_t^{b_t}$), the generation progress, the prefix length, the budget constraint $C(b_t)$, and a 16-dimensional learned budget embedding e_{b_t}.This composite input x_t is first normalized via Layer Normalization and then processed by a two-layer MLP trunk. Both linear layers within the trunk project the features to a hidden dimension of 256, and each is followed by a Gaussian Error Linear Unit (GELU) activation and 10% dropout. The resulting 256-dimensional representation is subsequently routed into two independent linear heads. The *panel head* projects the representation into 5 logits, corresponding to each candidate budget. Concurrently, the *advantage head* maps the representation to 2 outputs, which estimate the utility differences associated with the lower and upper neighboring budgets.

Controller training recipe. The controller is trained with panel distillation, advantage regression, and a lightweight pairwise ranking regularizer. The panel head is trained to match the teacher probability vector with binary cross entropy:

$$\mathcal{L}_{\text{panel}} = \text{BCEWithLogits}(r_t, p_t^{\text{teacher}}).$$

We apply neighbor-aware weights to the panel loss, with weights 0.5, 1.0, and 0.5 for the current, neighboring, and remote budget heads, respectively.

For the advantage head, we define the teacher utility

$$U_t(b) = p_t^b - \lambda C(b),$$

where $C(b)$ is the normalized budget cost and $\lambda = 0.2$. For each valid neighboring action $a \in \{\text{down}, \text{up}\}$, the teacher advantage is

$$\Delta_t^{a,*} = U_t(a(b_t)) - U_t(b_t),$$

where $a(b_t)$ denotes the neighboring budget reached by action a. The controller predicts the corresponding local advantage $\widehat{\Delta}_t^a$, and we train this head with SmoothL1 regression:

$$\mathcal{L}_{\text{adv}} = \frac{1}{|\mathcal{A}_t|} \sum_{a \in \mathcal{A}_t} \text{SmoothL1}\left(\widehat{\Delta}_t^a, \Delta_t^{a,*}\right),$$

where $\mathcal{A}_t$ denotes the set of valid neighboring actions.

We further add a pairwise ranking regularizer over each current-neighbor budget pair. Here, a pair refers to the current budget b_t and one of its valid neighboring budgets $a(b_t)$, where $a \in \{\text{down}, \text{up}\}$. This term does not compare all budgets globally; it only checks whether the controller predicts the correct local preference between the current budget and each neighboring budget. Specifically, if the teacher utility indicates that a neighboring budget has higher utility than the current budget, the corresponding predicted advantage should be positive; otherwise, it should be negative. The regularizer is applied only when the teacher utility difference is larger than a margin of 0.02, and follows standard pairwise learning-to-rank objectives (Burges et al., 2005). We use a pairwise temperature of 0.1.

The final controller objective is

$$\mathcal{L}_{\text{ctrl}} = \mathcal{L}_{\text{panel}} + \mathcal{L}_{\text{adv}} + 0.25\mathcal{L}_{\text{pair}}.$$

The controller is trained for 3 epochs with batch size 256, learning rate 3×10^{-4}, weight decay 0.01, hidden dimension 256, dropout 0.10, and gradient clipping at 1.0. The best controller checkpoint is selected by lowest development-set local regret.

A.2 Efficiency Details

This section clarifies how we compute the reported average budgets, since different sparse-attention backends expose different budget-like quantities.

LongBench Average Token Budget. For long-context LongBench experiments, we report a task-macro average routed budget. For each LongBench task $\mathcal{D}_j$, we compute

$$\bar{K}_{\mathcal{D}_j} = \frac{\sum_b b \cdot \text{count}_j(b)}{|\mathcal{D}_j|},$$

where $\text{count}_j(b)$ is the number of examples in task $\mathcal{D}_j$ routed to budget b. The final LongBench average budget is then

$$\bar{K}_{\text{LongBench}} = \frac{1}{N} \sum_{j=1}^{N} \bar{K}_{\mathcal{D}_j},$$

where $N = 13$ is the number of tasks in LongBench. Thus, each LongBench task contributes equally, matching the task-macro score aggregation.

In the SABER+Twilight framework, the main-table budget is defined as an attention-cost-equivalent budget rather than the raw count of tokens retained after Top-p pruning. For each sample, the effective budget K_i^{eff} is determined based on the selected routing path: if a sample utilizes the top-k branch, the effective budget remains equal to the routed upstream budget K_i; however, if the sample is processed via the Twilight Top-p branch, the effective budget is calculated as K_i scaled down by a factor of 1.4. This adjustment reflects the relative computational efficiency of the Twilight path, normalizing the attention overhead across different processing branches. This factor follows the attention speedup of Twilight and is used only as an accounting conversion. We then aggregate K_i^{eff} in the same way as Quest: first averaging within each LongBench task and then taking the macro-average across the 13 tasks. Therefore, the reported SABER+Twilight budget should be interpreted as an effective attention-cost budget, rather than the raw final kept-token count after Top-p pruning.

Table 6: Full LongBench summary results across fixed-budget Quest baselines, Twilight-style Top-p pruning, and our adaptive selector-budget allocation method. NQA and RB-P denote NarrativeQA and Repobench-P, respectively.

Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot	Synthetic	Code		Avg.
	Qasper	*MF-en*	*NQA*	*HotpotQA*	*2WikiMQA*	*Musique*	*GovReport*	*QMSum*	*MultiNews*	*TriviaQA*	*PR-en*	*LCC*	*RB-P*	*13-task*
Full	46.17	53.33	29.61	55.36	43.95	27.08	35.01	25.24	27.37	91.18	99.50	62.17	57.76	50.29
Quest-256	24.65	37.50	24.74	30.12	23.60	12.93	27.53	20.11	26.59	65.34	95.00	49.70	45.27	37.16
Quest-1024	38.47	49.32	27.81	47.43	38.48	20.59	33.71	23.67	26.60	81.94	99.50	60.78	52.96	46.25
Quest-4096	43.97	53.64	**30.05**	51.94	42.54	24.00	34.34	24.36	26.75	90.96	99.50	62.03	55.49	49.20
Quest-8192	**44.34**	53.25	29.71	**54.72**	**44.84**	25.98	34.62	24.98	26.70	**91.61**	**100.00**	62.02	54.20	49.77
Twilight	43.44	53.20	25.84	53.77	43.56	25.42	34.39	25.23	26.99	91.25	**100.00**	63.55	**58.06**	49.59
SABER	42.60	**54.48**	28.11	53.79	44.75	**28.01**	**35.02**	**25.51**	**27.29**	91.09	99.50	62.89	54.50	**49.81**
SABER + Twi.	42.87	53.13	26.91	52.55	43.00	26.15	34.17	25.49	27.03	89.71	99.50	**63.93**	**58.87**	49.49

Long-reasoning Average Token Budget. For long-reasoning experiments, we report the average attention budget during generation. Each decoded sample is divided into budget segments, where each segment has a budget $b_{i,t}$ and length $n_{i,t}$. For each sample i, we compute the token-weighted average budget:

$$\bar{B}_i = \frac{\sum_t b_{i,t} n_{i,t}}{\sum_t n_{i,t}}.$$

The benchmark-level reported budget is the arithmetic mean of $\bar{B}_i$ over samples:

$$\bar{B}_{\text{bench}} = \frac{1}{N} \sum_{i=1}^{N} \bar{B}_i,$$

where N is the number of evaluated generation samples in the benchmark. Intuitively, this formulation first derives a length-weighted average budget for each individual sample and then computes the arithmetic mean across the entire benchmark. For fixed-budget SeerAttention runs, the whole sample uses a single budget, so the reported value is simply the nominal fixed budget.

End-to-end Details We report end-to-end latency using the Quest sparse-attention kernel as the executable measurement backend. This choice is mainly due to implementation availability and compatibility: in our evaluation stack, we did not have an official SeerAttention-R end-to-end runtime path that could be directly integrated with our long-reasoning generation pipeline, adaptive routing logic, and benchmark scripts. Therefore, instead of mixing measured results from different kernels or reporting partially simulated runtime, we use the Quest kernel to measure the end-to-end decoding cost under controlled budget settings.

This measurement is intended to isolate the system-level effect of budget allocation rather than to compare low-level kernel engineering across sparse-attention implementations. For fixed-budget baselines, we run the same executable sparse-attention path with the corresponding budget. For SABER, the selected budget changes over generation, and the measured runtime follows the actual routed budget sequence. The predictor and controller overheads are included in the reported runtime and are amortized over the decoding intervals served by each routing decision.

Because the same kernel backend is used for both fixed-budget and adaptive-budget runs, the resulting latency comparison reflects the benefit of SABER's budget routing under a consistent execution environment. In particular, the reported end-to-end speedup should be interpreted as the speedup obtained from reducing the executed sparse-attention budget along the generation trajectory, rather than as a claim about a specialized SeerAttention kernel implementation.

A.3 Full LongBench Results

Table 6 reports the full LongBench summary table used in our comparison of fixed-budget baselines, Top-p pruning, and SABER and its top-p variant. These per-task results complement the grouped results in the main paper and show that SABER's improvement is not driven by a single task category. SABER achieves the best 13-task average among sparse-attention methods while remaining competitive across all categories, and SABER+Twi. further improves code performance, suggesting that upstream budget allocation can be combined with posterior pruning when the selected candidate set is sufficiently conservative.

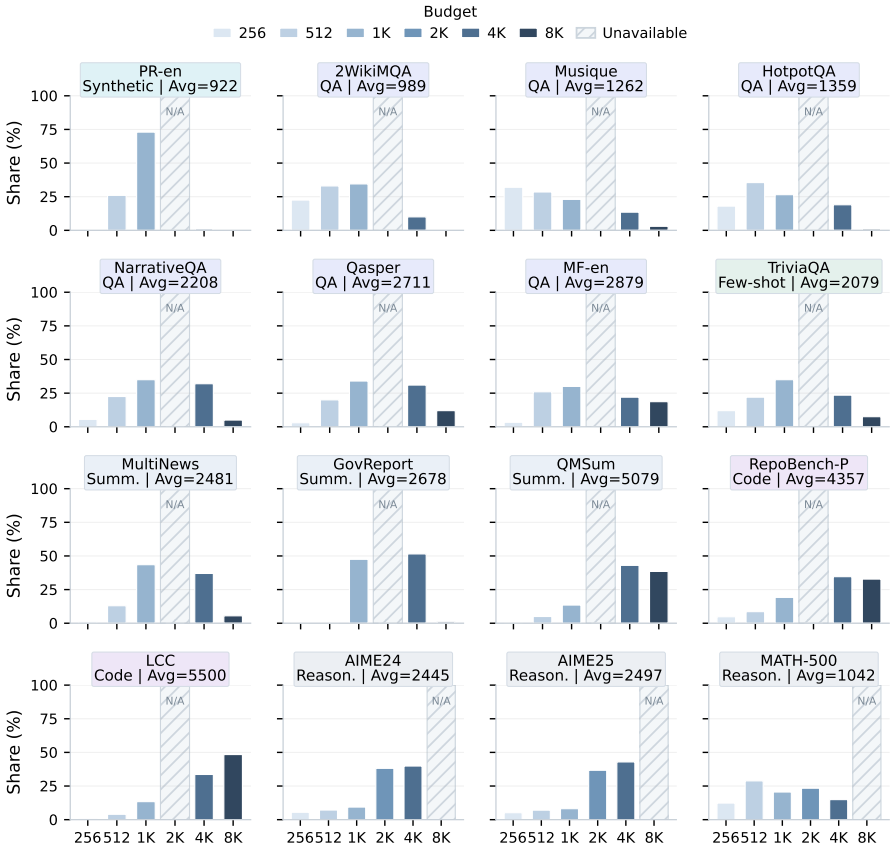


Figure 5: **Task-level budget distributions.** We visualize the budget distribution selected by SABER across LongBench and long-reasoning tasks. Each panel shows the percentage of routed samples or decoded tokens assigned to each budget bucket, with a shared color scale across all tasks. LongBench panels use routed-budget sample distributions, while reasoning panels use token-weighted generation-budget distributions. Hatched regions indicate unavailable budget buckets: 2K is not used in the LongBench budget set, and 8K is not used in the long-reasoning budget set. LongBench tasks are arranged by task family from lower-budget to higher-budget regimes, while reasoning tasks are separated in the last row.

A.4 Task-level Budget Distribution

Figure 5 visualizes the budget distributions selected by SABER across long-context and long-reasoning tasks. For long-context, the distributions correspond to prompt-level routed budgets, while for long reasoning they correspond to token-weighted online budgets across decoded tokens. Unavailable budget buckets are shown with hatching to keep the x-axis consistent across the two settings.

The distributions show that SABER learns task-dependent budget allocation rather than applying a uniformly small budget. Retrieval-style and localized QA tasks concentrate on lower budgets, while summarization and code tasks shift toward larger budgets because they require aggregating more distributed evidence or maintaining long-range dependencies. For reasoning, AIME24 and AIME25 spend more tokens at high budgets, whereas MATH-500 uses more low- and medium-budget intervals, matching their relative difficulty. These results provide distribution-level evidence that SABER adapts sparse-attention computation to task and trajectory demands.

Table 7: Bootstrap confidence intervals on LongBench. $\Delta = \text{SABER} - \text{baseline}$.

Baseline	Δ	95% CI
Quest-256	+12.65	[11.95, 13.35]
Quest-1024	+3.56	[3.04, 4.09]
Quest-4096	+0.61	[0.17, 1.07]
Quest-8192	+0.04	[-0.41, 0.49]
Full attention	-0.48	[-1.76, 0.80]

Table 8: Bootstrap confidence intervals on AIME24 and AIME25. $\Delta = \text{SABER} - \text{baseline}$.

Benchmark	Baseline	Δ	95% CI
AIME24	SeerAttention-1K	+7.29	[3.12, 12.08]
	SeerAttention-2K	+5.21	[0.42, 11.25]
	SeerAttention-4K	+1.88	[-1.67, 5.83]
	SeerAttention-8K	+3.54	[-1.25, 8.75]
	Full attention	+3.96	[1.25, 6.88]
AIME25	SeerAttention-1K	+6.46	[2.71, 11.04]
	SeerAttention-2K	+1.25	[-2.92, 5.83]
	SeerAttention-4K	-0.21	[-4.38, 4.38]
	SeerAttention-8K	+0.42	[-2.92, 4.17]
	Full attention	+1.88	[-2.08, 5.63]

A.5 Additional Evaluation and Ablation Studies

We provide additional analyses of statistical uncertainty, ablations of the trajectory predictor and budget controller, robustness to the judge model, and the data collection and training cost of SABER.

A.5.1 Statistical Uncertainty in Evaluation

We quantify the statistical uncertainty of LongBench and AIME evaluations using 20,000 bootstrap resamples. For LongBench, Table 7 reports the 95% confidence intervals (CIs) for the macro-score difference Δ. This measures whether SABER provides a consistent improvement over each baseline across the 13 LongBench tasks. The results show that SABER significantly improves over Quest-256, Quest-1024, and Quest-4096, while remaining statistically comparable to Quest-8192 and full attention at a substantially lower average budget.

For AIME, Table 8 reports the 95% CIs for the accuracy difference between SABER and each baseline. On AIME24, SABER achieves higher average accuracy than all baselines, with statistically significant improvements over SeerAttention-1K, SeerAttention-2K, and full attention. On AIME25, SABER improves over the 1K and 2K baselines and achieves accuracy comparable to the 4K, 8K, and full-attention settings; the improvement over SeerAttention-1K is statistically significant.

A.5.2 Ablation of the Predictor and Controller

For our online sparse attention budget router, the trajectory predictor provides the essential state-dependent routing signal, while the learned controller further improves the overall accuracy–budget trade-off. We isolate these contributions by comparing fixed SeerAttention-4K with deterministic upward-only and hysteresis policies driven by the same trajectory-predictor signals, and then with the SABER learned controller.

Using the same trajectory-predictor signals, upward-only routing increases the budget when predicted sufficiency falls below the decision threshold and never decreases it. Hysteresis adds conservative downgrades to suppress frequent budget switching (Hespanha et al., 2003): downgrading requires two consecutive qualifying checkpoints, and each budget change triggers a two-checkpoint cooldown. Table 9 compares these two heuristic routing policies with SABER's budget controller driven by the same trajectory predictor.

Table 9: Comparison of fixed-budget SeerAttention, predictor-driven heuristic routing, and the learned SABER controller on AIME24, AIME25, and MATH-500. Each entry reports accuracy (%) / average token budget.

Method	AIME24	AIME25	MATH-500
Fixed SeerAttention-4K	56.67 / 4096	45.21 / 4096	82.00 / 4096
Upward-only routing	57.08 / 2662	45.00 / 2759	85.80 / 1183
Hysteresis routing	50.83 / 2113	43.54 / 2233	84.60 / 831
SABER learned controller	58.54 / 2445	45.00 / 2497	85.20 / 1042

Table 10: LongBench performance of the long-context estimator trained with budget-sufficiency labels from different judge models.

Judge model	LongBench score	Avg. routed budget
Qwen3-30B-A3B	49.81	2654.2
GPT-5.5-Codex	49.39	1929.1

Predictor contribution. Upward-only routing uses the trajectory predictor without a learned controller and substantially reduces the budget relative to fixed SeerAttention-4K while preserving or improving accuracy. This demonstrates that the predictor itself provides an effective state-dependent signal for adaptive budget allocation.

Controller contribution. Holding the trajectory-predictor signals fixed, the learned controller achieves the most consistently balanced accuracy–budget trade-off across all three benchmarks. It uses less budget than upward-only routing while avoiding the accuracy degradation caused by hysteresis, showing that learned trajectory-aware control adds value beyond the trajectory-predictor signal alone.

A.5.3 Robustness to the Judge Model

We examine whether long-context budget estimation is sensitive to the judge model used to construct budget-sufficiency labels. We relabel all 7,200 budget outcomes using GPT-5.5-Codex and retrain the estimator. The two judges agree on 84.17% of individual budget labels. At the sample level, the assigned minimal budget remains unchanged for 63.3% of examples, while 25.0% shift to a lower budget and 11.7% to a higher budget. Because the alternative judge induces a different calibration of the budget-sufficiency labels, we recalibrate the inference threshold on the held-out calibration split, changing it from 0.65 to 0.75.

Table 10 shows that the estimator remains effective when trained with labels from a different judge model. With GPT-5.5-Codex labels, SABER achieves a LongBench score of 49.39 compared with 49.81 using Qwen3-30B-A3B labels, while reducing the average routed budget by 27.3%. Both judge models lead to competitive LongBench performance with substantially reduced routed budgets. These results suggest that SABER's effectiveness is robust to the choice of judge model.

A.5.4 Data Collection and Training Cost

We report the GPU-hour cost of constructing budget-conditioned training data and optimizing the learned components of SABER in both the long-context and long-reasoning settings.

For long-context training, we evaluate 5,000 examples under four candidate budgets, producing 20,000 budget-conditioned completions. From this pool, we select 1,800 examples, yielding 7,200 budget-conditioned training labels. For long reasoning, we generate rollouts for 7,000 DAPO problems under five candidate budgets. The trajectory predictor is trained on the full rollout pool, while the controller uses cross-budget prefix-replay features constructed from a 100-question subset, yielding 48,445 cached prefix-level samples.

Table 11 shows that the long-context pipeline requires 20.47 GPU-hours in total, while the long-reasoning pipeline requires 204.12 GPU-hours. In both settings, parameter optimization accounts for only a small fraction of the total cost; most computation is spent on generating budget-conditioned

Table 11: Data collection and training cost for SABER in the long-context and long-reasoning settings.

Stage	Scale	GPU-hours
Long-context		
Multi-budget completion generation	5,000 examples, 20,000 completions	6.92
Qwen judge labeling	20,000 completions, batched inference	13.30
Predictor training	1,800 examples, 7,200 labels	0.254
Total		**20.47**
Long reasoning		
DAPO trajectory generation	7,000 problems $\times$ 5 budgets	125.92
Predictor training	5 budget-specific predictors	2.10
Controller feature preparation	100 questions $\times$ 5 budgets, 512-token prefix replay	76.09
Controller training	48,445 cached prefix samples	0.011
Total		**204.12**

trajectories, judge labeling, and prefix replay. These data-collection and training costs are incurred once, after which the trained predictor and controller are reused across subsequent inference requests.

A.6 Additional Related Work

Tactic. Tactic (Zhu et al., 2026) is a recent training-free adaptive sparse-attention method that determines upstream budgets from QK-derived attention statistics using clustering and distribution fitting. SABER instead predicts the sufficiency of candidate budgets from intermediate model states, separating budget estimation from the sparse-attention backend.

Tactic reports results on six LongBench tasks using the same base model, but we could not identify a publicly available implementation. We therefore compare the reported results on these shared tasks, normalizing performance to each paper's full-attention baseline. Relative to Tactic-90%, SABER uses 10.4% fewer selected tokens on average while maintaining comparable performance retention, only 0.88 percentage points lower. Beyond long-context adaptation, SABER further formulates long reasoning as an online budget-control problem, allowing the attention budget to increase, remain unchanged, or decrease as the reasoning state evolves.