The Sparse Frontier: Sparse Attention Trade-offs in Transformer LLMs

Anonymous ACL submission

Abstract

Sparse attention offers a promising strategy to extend long-context capabilities in Transformer LLMs, yet its efficiency–accuracy trade-offs remain unclear due to the lack of comprehensive evaluation. We address this gap with the largest-scale empirical analysis to date of training-free sparse attention, evaluating six methods across multiple model families and sizes, sequences up to 128K tokens, and sparsity levels up to 0.95 (i.e., $1/20$ attention budget) on nine diverse tasks. We first organise the rapidly evolving landscape of sparse attention methods into a taxonomy along four design axes. Our analysis then yields actionable insights: 1) sparse attention is effective—larger sparse models outperform smaller dense ones at equivalent cost, improving the Pareto frontier; 2) due to computational constraints, token-to-page importance estimation is unfeasible during prefilling, where the choice of an alternative solution (global-to-token or block-to-block) depends on the task, but is possible during decoding, enabling better generalisation and tolerance to higher sparsity; 3) longer sequences tolerate higher sparsity, suggesting that fixed-budget methods in production are suboptimal. Together, these findings provide practical guidance for deploying sparse attention and methodological recommendations for future evaluations.

1 Introduction

The ability to model long sequences in large language models (LLMs) lies at the heart of long-context processing (Liu et al., 2025a) and inference-time scaling (Snell et al., 2024; Muennighoff et al., 2025). The fundamental bottleneck for this ability is the self-attention mechanism (Bahdanau et al., 2015; Vaswani et al., 2017): during the prefilling stage, its computational complexity scales quadratically with sequence length—hence, ballooning time-to-first-token and deployment cost (Jiang et al., 2024). In the decoding phase, the key–value (KV) cache grows linearly with sequence length, and the need to load from memory this expanding cache for each generation step dominates the runtime (Nawrot et al., 2024).

Sparse attention mechanisms aim to address these challenges by approximating dense attention outputs with only a subset of query–key interactions (Fu, 2024). These span both training-based variants—such as DMS (Łańcucki et al., 2025), DeepSeek's NSA (Yuan et al., 2025), and SWA used in OpenAI's gpt-oss and Google's Gemma 3—and training-free methods that operate directly on pretrained models, such as Vertical-Slash (Jiang et al., 2024) deployed in Qwen 2.5-1M (Yang et al., 2025) and integrated into vLLM. Sparse attention has not only seen widespread adoption in industry, but also in the research community: over 150 papers with "sparse attention" in the title were submitted to arXiv between January 2025 and January 2026. Despite this popularity, the viability and robustness of sparse attention remain unclear due to a lack of comprehensive large-scale evaluation.

In this work, we carry out the largest-scale empirical analysis to date of training-free sparse attention methods, covering three model families (Qwen 2.5, Llama 3.1, Gemma 3) with sizes between 4B and 72B parameters, sequences between 16K to 128K tokens, and sparsity levels up to 0.95 (i.e., $1/20$ attention budget). To enable a controlled analysis, we first survey existing approaches, addressing the challenge of comparing rapidly evolving methods whose implementation details often obscure their core design principles. We distil these approaches into four key axes: units of sparsification (blocks/pages or verticals and slashes), importance estimation (fixed or context-aware), budget allocation across layers (uniform or adaptive), and KV cache management (eviction or full cache). Based on this taxonomy, we select six representative methods spanning these design dimensions and harmonise their implementations, allowing us to rigorously evaluate their distinct effects.

We focus specifically on training-free sparse attention because training-based alternatives require prohibitive computational resources and possibly access to proprietary training data (Nawrot et al., 2024). While this is a limitation, we expect insights from our training-free analysis to transfer to training-based methods given their similarity.[1] While previously Li et al. (2025), Liu et al. (2025b), and Yuan et al. (2024) provided a preliminary exploration of training-free methods, they covered limited configurations, specific use cases, or did not control for sequence lengths, hindering a systematic analysis (see Appendix C for a detailed comparison).

For our evaluation, we curate a benchmark suite of 9 long-context tasks designed to systematically probe the influence of key factors on sparse attention performance. These factors include diverse *task types* (ranging from retrieval to multi-hop variable tracking and information aggregation), varying *naturalness* of sequences (synthetic or natural language), and precisely controlled *sequence lengths*. The importance of these dimensions is underscored by prior work indicating their significant impact on sparse attention effectiveness (Chen et al., 2024; Liu et al., 2024). Alongside established benchmarks (Rajpurkar et al., 2018; Pang et al., 2022; Tseng et al., 2016), we introduce novel, more challenging tasks based on natural language story templates. These complement synthetic benchmarks like RULER (Hsieh et al., 2024), whose results may fail to extrapolate to realistic data, by evaluating core skills in a controllable yet realistic natural language setting. All this provides us with a toolbox to address fundamental questions that currently remain unresolved:

RQ1: Is sparse attention effective? (Section 4.1) An isoCost analysis reveals that sparsification enables larger sparse models to outperform smaller dense ones at equivalent cost (i.e., FLOPs during prefilling and memory reads during decoding). For long sequences, only high-sparsity configurations lie on the Pareto frontier.

RQ2: Which sparse attention method should practitioners use? (Section 4.2) Prefill and decoding phases display different trends. During prefilling, computational constraints force a choice between fine-grained token selection (Vertical-Slash) and block-based selection (Block-Sparse)—neither of which generalises across all tasks. During decoding, token-to-page selection (e.g., Quest) is computationally feasible, enabling greater flexibility and higher compression tolerance than prefilling.

RQ3: How does the sequence length affect tolerance to sparse attention? (Section 4.3) Longer sequences permit higher sparsity while maintaining accuracy, consistently across model families. This suggests that fixed-budget methods deployed in production are suboptimal, and future designs should adapt sparsity to sequence length.

Overall, our findings provide practical guidance for deploying sparse attention and methodological recommendations for future evaluations in this rapidly evolving field.

2 Training-Free Sparse Attention

The self-attention mechanism computes query Q, key K, and value V representations from an input sequence $X \in \mathbb{R}^{n \times d}$. The output O_i for the i-th token is a weighted sum of values, $O_i = \sum_{j=1}^{n} A_{ij} V_j$, where the attention weights A_{ij} are derived from scaled dot-products between queries and keys: $A_i = \text{softmax}(Q_i K^\top / \sqrt{d})$. We omit multi-head details for brevity.

Transformer-based text generation involves two phases. **Prefilling** processes the entire input sequence, computing the lower-triangular part of the $n \times n$ attention matrix A, leading to $O(n^2)$ complexity. **Decoding** generates tokens autoregressively. While attention is $O(n)$ per step (single query), loading the expanding Key-Value (KV) cache from memory becomes the main bottleneck.

Sparse attention methods reduce these costs by computing only a subset of QK interactions, making A sparse. This lowers computational load during prefilling and memory transfers during decoding. We quantify the effectiveness of these methods using **sparsity**—the fraction of non-computed QK interactions. Equivalently, sparsity of $1 - 1/k$ corresponds to retaining only a $1/k$ fraction of the attention interactions. For instance, sparsity 0.9 (or equivalently, $1/10$ attention budget) means computing only 10% of the original QK interactions.

The speedup from sparse attention depends on how much of total cost is attention. Since attention scales quadratically while other components (MLP, embeddings) scale linearly with sequence length, attention dominates at longer contexts—yielding greater benefits from sparsification. Models with built-in architectural sparsity, such as

[1] For instance, Quest (Tang et al., 2024) and NSA (Yuan et al., 2025) both use page-based selection for sparse decoding.

sliding-window or linear attention layers, have lower baseline attention ratios and require longer sequences for additional sparsification to provide comparable gains (Appendix B).

We categorise training-free sparse attention methods along four axes: unit of sparsification, importance estimation, budget allocation, and KV cache management. We exclude token merging methods (Wang et al., 2024; Nawrot et al., 2024), which do not rely on sparsity.

2.1 Unit of Sparsification

Sparse attention methods differ primarily in the structural units of the attention matrix they prune or retain. Common units include *local windows* (contiguous regions around each query), *vertical columns* (tokens globally available to all queries), *slashes* (tokens at fixed offsets from each query), and *blocks* (fixed-size tiles of the attention matrix, such as 64×64 tokens). Larger structured units such as blocks or windows offer improved computational efficiency via better memory locality, whereas smaller units allow finer-grained, more precise selection of important information.

Block-based methods select blocks of units to approximate full attention. For prefilling, Star Attention approximates attention using local blocks and the first prefix block. MInference's Block-Sparse pattern (Jiang et al., 2024) additionally incorporates a set of dynamically selected blocks for each chunk of query tokens. For decoding, Quest (Tang et al., 2024) and InfLLM (Xiao et al., 2024a) divide the KV cache into contiguous pages and select a subset of them for each decoded token.

Vertical–slash patterns represent another essential class of units. Early sparse attention methods like LM-Infinite (Han et al., 2024) and StreamingLLM (Xiao et al., 2024b) utilised local sliding windows supplemented by prefix tokens shared globally, also known as attention sinks. Extending this approach, Tri-shape (Li et al., 2025) added full attention for suffix tokens, whereas SnapKV (Li et al., 2024b) introduced dynamically chosen vertical columns. MInference (Jiang et al., 2024) built on this by adding diagonal slashes at arbitrary offsets beyond the local window.[2] For a visual illustration of these attention patterns, see Figure 4 in the Appendix.

[2]Interestingly, to efficiently compute attention along these diagonals, MInference uses 64×64 blocks aligned with these diagonals rather than computing attention for individual query–key pairs.

2.2 Importance Estimation

To identify which specific units to retain, one can use fixed patterns—applied identically across all inputs—or dynamic patterns that adapt to the content being processed. Fixed patterns introduce no computational overhead but cannot adapt to varying input requirements, while dynamic patterns better preserve model quality but require additional computation to identify important connections.

Fixed patterns are identified with offline calibration to work well across all inputs. StreamingLLM (Xiao et al., 2024b), LM-Infinite (Han et al., 2024) and MoA (Fu et al., 2024) determine the number of initial tokens (attention sinks) and the width of a local sliding window.

Content-aware methods typically estimate the importance of QK units (tokens, blocks, or diagonals) to retain only the top-k most relevant ones, maximising attention score recall. They use lightweight heuristics such as approximated attention scores from highest-magnitude dimensions (SparQ; Ribar et al., 2024) or block-wise pooled token representations (Jiang et al., 2024). Some approaches subsample queries (SampleAttention; Zhu et al., 2024), recognising that recent query tokens often provide better indicators of KV unit importance, as in MInference's Vertical-Slash (Jiang et al., 2024) and SnapKV (Li et al., 2024b). During decoding, aggregated attention scores (H2O; Zhang et al., 2023) or the latest query (TOVA; Oren et al., 2024) guide the selection of KV units, again prioritising units likely to receive high attention weights. Some methods incorporate complementary heuristics alongside attention scores, such as norms of keys (Devoto et al., 2024) or values (Guo et al., 2024).

Critically, the cost of sparse attention includes both the sparse operation and importance estimation overhead. During prefilling, per-cell importance estimation would require quadratic cost, so methods must either select fine-grained units globally (e.g., vertical columns shared across all queries) or use coarser block-to-block selection. More flexible unit selection improves accuracy but increases estimation cost. During decoding, per-query selection is feasible since only one query is processed per step, enabling methods like Quest to perform finer token-to-block selection and tolerate higher compression (Section 4.2).

2.3 Budget Allocation

The third dimension in sparse attention design is budget allocation: distributing computational resources across model components (layers and heads) for a target sparsity. This involves a trade-off between uniform simplicity and adaptive expressivity.

Uniform allocation assigns an equal budget (tokens or blocks) to each head as in Block-Sparse (Jiang et al., 2024) and SnapKV (Li et al., 2024b). This is computationally simple but overlooks that layers and heads contribute differently to accuracy and have diverse attention sparsity (Zhang et al., 2024).

Adaptive methods vary budget allocation. PyramidKV (Cai et al., 2024) and PyramidInfer (Yang et al., 2024b) observe that attention score entropy decreases with layer depth, allocating larger budgets to early layers. Mixture of Sparse Attention (MoA; Fu et al., 2024) uses Taylor approximations to optimally distribute the global budget across layers. Within layers, Ada-KV (Feng et al., 2024) flexibly allocates by selecting top-$(k \times h)$ tokens (where h is head count), allowing critical heads to retain more keys while pruning others. *Threshold-based* allocation offers maximum flexibility by removing a fixed global budget. Methods like Twilight (Lin et al., 2025), FlexPrefill (Lai et al., 2025), Tactic (Zhu et al., 2025), and SampleAttention (Zhu et al., 2024) set coverage thresholds (e.g., 95% of attention mass). Each head dynamically selects units to meet these thresholds, allowing high-entropy attention heads to consume more budget and the overall budget to vary per sample.

2.4 KV Cache Management

The final dimension distinguishes methods based on KV cache management during decoding.

KV cache eviction methods (e.g., H2O (Zhang et al., 2023), SnapKV (Li et al., 2024b)) permanently discard selected tokens based on estimated importance, directly reducing memory footprint but sacrificing information fidelity as discarded tokens cannot be recovered.

Full KV cache retention methods (e.g., Quest (Tang et al., 2024), SparQ (Ribar et al., 2024)) maintain the entire cache but optimize computation by selectively loading only necessary KV pairs during attention calculation. While incurring small memory overhead for auxiliary data structures needed for importance estimation, they avoid information loss and can operate effectively at higher sparsity levels compared to eviction-based methods, though they do not reduce peak memory requirements.

3 Experimental Setup

3.1 Models

We perform experiments primarily on Qwen 2.5 (Yang et al., 2024a) (7B, 14B, 32B, 72B parameters), complemented by Llama 3.1 (Dubey et al., 2024) (8B, 70B) and Gemma 3 (Gemma Team, 2025) (4B, 12B, 27B). All three families use instruction-tuned variants to support chain-of-thought evaluation. Qwen 2.5 was selected as our primary family as it uniquely satisfies strict methodological requirements for controlled scaling experiments—see Appendix A.3 for rationale. For Qwen and Llama, we modify the attention mechanism across all layers. Gemma 3 employs hybrid attention where 5 out of 6 layers use sliding window attention (1024 tokens) by design; we apply sparse attention methods only to the remaining dense (global attention) layers. We preserve the original architectures and utilise the vLLM inference engine (Kwon et al., 2023) with full bf16 precision. Implementation details are in Appendix A.1.

3.2 Sparse Attention Methods

We evaluate six state-of-the-art sparse attention methods (Table 1), which we choose as a representative set spanning across the key dimensions described in Section 2. We focus exclusively on content-aware methods, as prior work has demonstrated that fixed patterns consistently underperform their content-aware counterparts (Li et al., 2025).

3.3 Tasks

We evaluate 9 diverse tasks selected to reflect different characteristics along 3 key dimensions known to influence sparse attention performance: task difficulty—defined by *Dispersion* (how hard it is to locate necessary information) and *Scope* (how much information must be processed) (Goldman et al., 2024)—and data *Naturalness* (natural language vs. synthetic data). This multi-dimensional approach is motivated by recent findings that attention patterns vary significantly across task types: retrieval tasks often exhibit localised attention, while reasoning tasks show more uniform distributions that are challenging for sparse methods (Liu et al.,

	Method	Unit	Budget	KV Cache Management
Prefill	**Vertical-Slash** (Jiang et al., 2024)	verticals and slashes	uniform	N/A
	FlexPrefill (Lai et al., 2025)	verticals and slashes	threshold-based	N/A
	Block-Sparse (Jiang et al., 2024)	blocks	uniform	N/A
Decode	**SnapKV** (Li et al., 2024b)	tokens	uniform	eviction
	Ada-SnapKV (Feng et al., 2024)	tokens	adaptive	eviction
	Quest (Tang et al., 2024)	pages	uniform	full cache

Table 1: Full list of content-aware sparse attention methods benchmarked in our experiments. These represent diverse strategies in terms of units, budget allocation, and KV cache management.

2025c; Chen et al., 2024; Li et al., 2025). The naturalness dimension is also crucial, as synthetic tasks yield different token representation distributions compared to natural language (Liu et al., 2024). Our task suite therefore incorporates four core tasks from the **RULER** benchmark (Hsieh et al., 2024)—Retrieval (NIAH), Multi-hop reasoning (VT), Aggregation (CWE), and QA (SQuAD)—to provide controlled environments (mostly synthetic) for specific capabilities. We complement these with natural texts from benchmarks with minimal contamination risk (Li et al., 2024a), such as QuALITY and TOEFL, though these represent low-dispersion, low-scope tasks. Thus, we additionally introduce three novel tasks (**Story** Retrieval, Multi-hop, Filtering) that translate RULER's challenging tasks (with high dispersion or scope) into naturalistic narratives, more representative of real-world use. We deliberately avoid open-ended tasks like summarisation due to unreliable evaluation metrics (Yen et al., 2024; Ye et al., 2024), focusing instead on structured-output tasks requiring factual answers, enabling precise evaluation via Exact Match Accuracy, Intersection-over-Union (IoU), and F1 score (all ranging from 0 to 1). These tasks are summarised in Table 4, with detailed descriptions in Appendix A.2 and examples in Appendix G.

3.4 Evaluation Settings

Our evaluation covers input lengths of 16k, 32k, and 64k tokens for all model families, with 128k evaluations limited to Qwen and Llama using Vertical-Slash and Quest only; Gemma exhibited near-zero performance at 128k. We use 100 samples per configuration for Qwen and 50 for Llama and Gemma. We evaluate all combinations of task, model size, sequence length, and sparse attention pattern at sparsity levels from 0 (dense) to 0.95 (i.e., $1/20$ attention budget), interpolating performance at intermediate points. We ensure input samples are within 95–100% of the target maximum token length, providing a consistent basis for evaluating the impact of sequence length on performance. Following Karpinska et al. (2024), we adopt a structured prompt format that encourages models to explicitly reason through chain-of-thought before providing answers in a consistent, parsable structure (see Appendix E). As metrics of computational cost, we report FLOPS for prefilling and memory access for decoding, as these reflect the respective computational bottlenecks of each phase (see Section 2). Appendix B provides more details, including indexing costs for sparse attention methods.

4 Results

4.1 isoCost Analysis

RQ1: Is sparse attention effective? Results in Figure 1 illustrate the average performance across tasks against computational cost for different model sizes and levels of sparsity.[3] As implementation-agnostic proxies for computational cost, we use FLOPs for prefilling and memory transfers for decoding, which correlate with wall-clock time under optimised implementations (see Appendix B for cost formulas and breakdowns). We visualise Pareto frontiers to identify configurations that offer the best performance-cost trade-offs, i.e., those not dominated by any other configuration in terms of both cost and performance.

Sparse attention improves the Pareto frontier. In Figure 1, the Pareto frontier reveals an efficiency crossover where sparsification enables larger sparse models to outperform smaller dense ones at equivalent computational cost. For Qwen at 128k tokens, only high-sparsity configurations lie on the Pareto frontier. During prefilling, models with sparsity

[3]We approach this question using Vertical-Slash for prefilling and Quest for decoding, as these are, on average, the best-performing patterns for their respective inference phases (see Section 4.2).

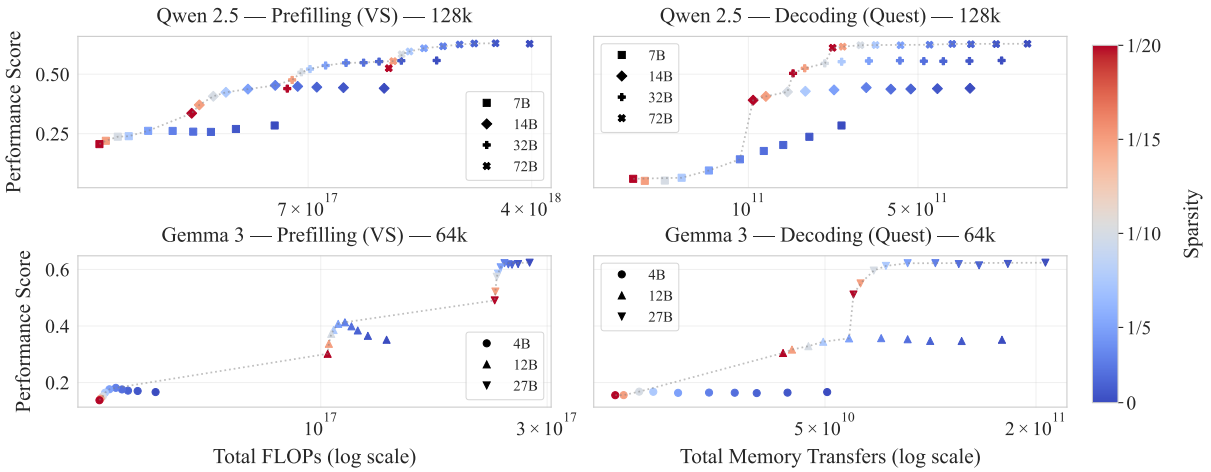

Figure 1: isoCost analysis for Qwen 2.5 (128k tokens) and Gemma 3 (64k tokens). Each point corresponds to a (model size, sparsity) configuration, with performance aggregated across 9 tasks. **Left column**: prefilling with Vertical-Slash (Jiang et al., 2024) (FLOPs). **Right column**: decoding with Quest (Tang et al., 2024) (memory transfers). Standard error is negligible (Appendix D.1) and omitted for visual clarity. Dotted lines show Pareto frontiers connecting configurations that are not dominated by any other configuration. Key findings: (1) sparsification enables larger sparse models to outperform smaller dense models at equivalent cost; (2) the impact of sparsity is less pronounced for Gemma due to its sliding-window architecture (Figure 14).

0.8–0.93 (i.e., $1/5$ to $1/15$ attention budget) remain optimal, while sparsity 0.95 ($1/20$ budget) falls below the optimal boundary. Decoding shows better resilience to high sparsity, with even 0.95 sparsity configurations being preferable to smaller dense models. For Gemma, we observe similar trends during decoding, but configuration overlap is absent for prefilling—this reflects Gemma's lower baseline attention ratio because of its sliding-window architecture (see Section 2 and Appendix B).

4.2 Per-Task Analysis

RQ2: Which sparse attention method should practitioners use? Figure 2 presents per-task performance across sparse attention methods, aggregated over three model families and sequence lengths up to 64k. The 9 tasks introduced in Section 3.3 are grouped by their information retrieval characteristics: Single QA (one query, localised answer), Multiple QA (multiple queries targeting distinct facts), High Scope/Low Dispersion (broad context, concentrated answers), and Low Scope/High Dispersion (narrow focus, scattered information). Three findings emerge from this analysis.

Prefill and decoding phases display different flexibility. As discussed in Section 2, the computational constraints of each inference phase fundamentally determine what patterns can be selected, which in turn affects generalisation across tasks. During prefilling, per-cell importance estimation would require quadratic cost, leaving only two strategies: global selection of fine-grained units (Vertical-Slash) or block-to-block selection (Block-Sparse). Neither strategy dominates—the optimal choice is task-dependent.

During prefilling, Vertical-Slash shows strong performance on retrieval tasks (Low Scope, Low Dispersion) by enabling fine-grained token selection for locating specific facts. Tasks demanding broader context access or multi-step reasoning (High Scope or Dispersion, e.g., Ruler VT, Story Filtering) benefit from Block-Sparse, which selects distinct key-token blocks for each query block, accommodating the processing of multiple independent segments.

During decoding, token-to-page selection becomes computationally feasible since only one query is processed per step. This greater flexibility enables Quest to generalise better across tasks and tolerate higher compression than either prefilling approach, while retaining the full KV cache. Eviction-based decoding methods (SnapKV, AdaSnapKV) that permanently discard tokens illustrate the cost of sacrificing the full cache—irreversible compression is detrimental when discarded tokens become relevant later, though this comes with the benefit of reduced memory footprint. Nevertheless, Quest can degrade on synthetic tasks such as Ruler NIAH, where random symbol sequences yield less distinguishable key representations compared to

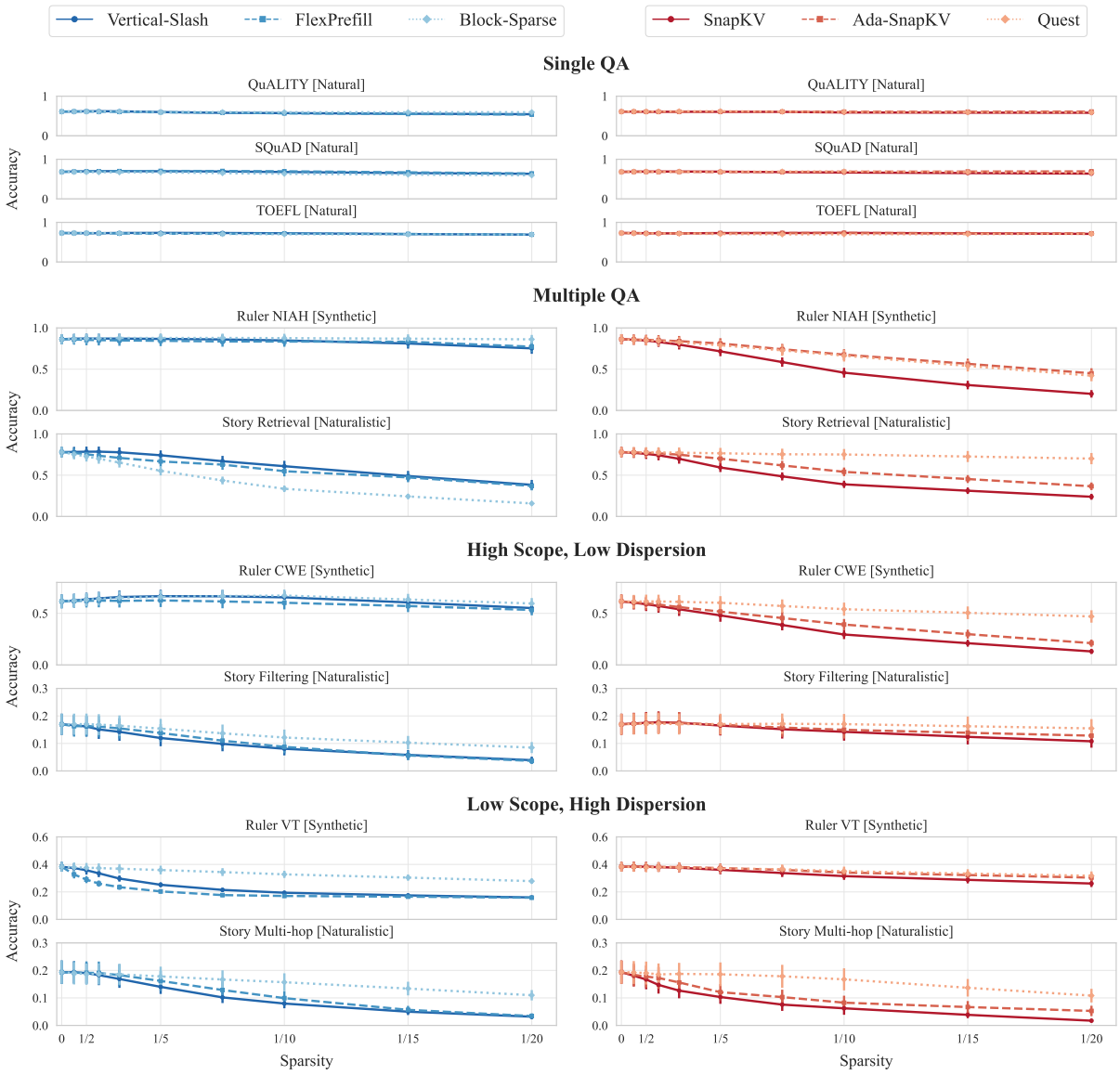

Figure 2: Per-task performance comparison of sparse attention methods, aggregated over Qwen 2.5, Llama 3.1, and Gemma 3 models at sequence lengths 16k, 32k, and 64k. Error bars indicate standard error. **Left column**: prefilling methods (Vertical-Slash, FlexPrefill, Block-Sparse). **Right column**: decoding methods (SnapKV, Ada-SnapKV, Quest). Tasks are grouped by information retrieval characteristics. Per-family breakdowns are provided in Appendix D.2.

natural language (Liu et al., 2024)—Quest's page-level granularity amplifies this effect, as coarser blocks struggle more than Ada-SnapKV's token-level selection to differentiate between unrelated token sets.

Dynamic budget allocation benefits are phase-dependent. Adaptive methods that allocate different budgets across layers or sequences yield inconsistent results. During prefilling, FlexPrefill matches or underperforms Vertical-Slash's uniform allocation, likely due to the "attention sink phenomenon" (Chen et al., 2024): threshold-based selection captures high-attention tokens while missing information in the distribution's long tail. During decoding, Ada-SnapKV consistently outperforms uniform SnapKV, particularly on multi-query tasks (Story Retrieval), though both eviction methods remain inferior to Quest's full-cache approach.

Sparsity tolerance varies dramatically across tasks. The gap between task groups reveals a deployment risk: methods achieving high sparsity on easy tasks may fail on harder ones. Single QA tasks (QuALITY, SQuAD, TOEFL) tolerate sparsity 0.95 ($1/20$ budget) with minimal degradation across all

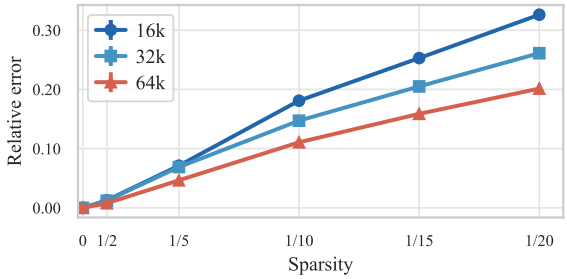

Figure 3: Sequence length effects on sparsity tolerance. Relative error is $(\bar{p}_{\text{dense}} - \bar{p}_{\text{sparse}})/\bar{p}_{\text{dense}}$, where $\bar{p}$ denotes mean performance. Results aggregated across all tasks, methods, and models (Qwen 2.5, Llama 3.1, Gemma 3). Absolute error and per-family breakdowns are provided in Appendix D.3.

methods. Multiple QA tasks (Ruler NIAH, Story Retrieval) show substantial degradation at sparsity 0.8–0.9 ($1/5$ to $1/10$ budget). Tasks with High Scope or High Dispersion degrade even at modest sparsity (0.5–0.67, i.e., $1/2$ to $1/3$ budget) for some methods. Evaluating sparse attention only on Single QA benchmarks—or averaging across task types—masks these vulnerabilities. Robust deployment requires testing across diverse task characteristics, as sparsity levels safe for retrieval tasks can cause failures on aggregation or multi-hop reasoning. Moreover, sequence naturalness affects methods asymmetrically—Quest outperforms Ada-SnapKV on natural-language retrieval (Story Retrieval) but underperforms on synthetic retrieval (Ruler NIAH)—underscoring the need for benchmarks spanning both natural and synthetic data.

4.3 Sequence Length Effects

RQ3: How does sequence length affect tolerance to sparse attention? Figure 3 shows that **for a fixed attention budget fraction, longer sequences incur smaller degradation**: for example, at a $1/20$ budget, the relative error decreases from ≈ 0.33 (16k) to ≈ 0.26 (32k) and ≈ 0.20 (64k). This indicates that the same sparsity ratio becomes less harmful as the sequence length grows. This pattern holds consistently across all model families. Nawrot et al. (2024) observe similar results for a training-aware KV compression method: their learned mechanism applies lower sparsity at the beginning of sequences and increases sparsity with sequence length. This behaviour may be explained by Herdan's law (Herdan, 1960), which posits that new information becomes rarer over time, facilitating higher sparsity with distance.

To relate this trend to **budget scaling**, we interpret the plot as approximate *iso-error* curves. For a target relative error of ≈ 0.2, the required budget fractions are roughly $1/10$ (16k), $1/15$ (32k), and $1/20$ (64k). In contrast, a **fixed token budget** would imply fractions $1/10 \to 1/20 \to 1/40$ as length grows, which already exceeds the ≈ 0.2 error target at 32k (the $1/20$ point is ≈ 0.26). For a stricter target such as ≈ 0.1, the scaling is less uniform: $1/5$ stays below 0.1 for both 16k and 32k, while 64k requires only a modest reduction in sparsity to stay near 0.1. These observations imply that the optimal token budget should grow *sublinearly* with sequence length: doubling the context does not require doubling the token budget, but keeping the budget constant would incur increasing degradation. While current dynamic methods lack robustness (Section 4.2), developing reliable sublinear budget allocation mechanisms remains a promising direction for future work.

5 Conclusions

This study provides the largest-scale empirical analysis of training-free sparse attention to date, covering three model families (Qwen 2.5, Llama 3.1, Gemma 3), model scales (4B–72B parameters), sequence lengths (16K–128K tokens), sparsity levels up to 0.95 (i.e., $1/20$ attention budget), and nine diverse long-sequence tasks. We organise the rapidly evolving landscape of sparse attention methods into a taxonomy along four design axes and introduce novel benchmarks consisting of natural texts that are fully controllable yet challenging. Our analysis yields three key insights.

Evidence of effectiveness. Sparse attention enables larger models to outperform smaller dense ones at equivalent computational cost, improving the Pareto frontier. Thus, sparsity becomes crucial for optimal LLM scaling.

Practical deployment guidance. Method selection should be task-aware: fine-grained token selection (e.g., Vertical-Slash) excels at retrieval, chunk-based methods (e.g., Block-Sparse) suit reasoning and aggregation, and Quest provides robust decoding across most scenarios.

Design recommendations. Longer sequences tolerate higher sparsity while maintaining accuracy. This suggests that fixed-budget methods deployed in production are suboptimal; future designs should adapt sparsity levels to sequence length, possibly growing the token budget sublinearly.

Limitations

First, we evaluate only training-free sparse attention methods. Training-based approaches could reduce train-inference mismatch, but require substantial computational resources and access to proprietary training data.

Second, our experimental coverage, while extensive, is bounded. We evaluate three model families (Qwen 2.5, Llama 3.1, Gemma 3) that met our methodological requirements for controlled scaling experiments with native long-context support; other families may exhibit different behaviour. We test only instruction-tuned models; reasoning models with extended chain-of-thought capabilities (e.g., o1, DeepSeek-R1) may have different attention patterns and sparsity tolerance. Our nine tasks, though selected to span diverse dispersion levels, processing scopes, and data naturalness, do not exhaustively cover all long-context scenarios—open-ended tasks like summarisation were excluded due to unreliable automated metrics. Additionally, experiments at 128k tokens are limited due to low baseline performance and lack of robustness across models; more conclusive evidence on how sequence length affects sparse attention scaling requires stronger long-context models.

Third, we report hardware-agnostic computational costs (FLOPs and memory access) rather than wall-clock timings. Actual speedups depend on hardware, batch size, and implementation quality, which vary across deployment environments.

Fourth, we do not investigate interactions between sparse attention and other model efficiency techniques such as quantisation, weight pruning, or mixture-of-experts sparsity. These methods are often combined in practice, and their joint effects on attention sparsity tolerance remain unexplored.

References

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv:2406.02069*.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. 2024. Magicpig: LSH sampling for efficient LLM generation. *arXiv:2410.16179*.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l2 norm-based strategy for kv cache compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18476–18499.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 herd of models. *arXiv:2407.21783*.

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

Tianyu Fu, Haofeng Huang, Xuefei Ning, Genghan Zhang, Boju Chen, Tianqi Wu, Hongyi Wang, Zixiao Huang, Shiyao Li, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. 2024. Moa: Mixture of sparse attention for automatic large language model compression. *arXiv:2406.14909*.

Yao Fu. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv:2405.08944*.

Gemma Team. 2025. Gemma 3 technical report. *arXiv:2503.19786*.

Omer Goldman, Alon Jacovi, Aviv Slobodkin, Aviya Maimon, Ido Dagan, and Reut Tsarfaty. 2024. Is it really long context if all you need is retrieval? Towards genuinely difficult long context NLP. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16576–16586.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008.

Gustav Herdan. 1960. *Type-Token Mathematics*. Mouton, The Hague.

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

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Marzena Karpinska, Katherine Thai, Kyle Lo, Tanya Goyal, and Mohit Iyyer. 2024. One thousand and one pairs: A "novel" challenge for long-context language models. *arXiv:2406.16264*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. 2025. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *The Thirteenth International Conference on Learning Representations*.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Maria Ponti. 2025. Inference-time hyper-scaling with KV cache compression. In *Advances in Neural Information Processing Systems*.

Xinze Li, Yixin Cao, Yubo Ma, and Aixin Sun. 2024a. Long context vs. RAG for LLMs: An evaluation and revisits. *arXiv:2501.01880*.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. 2025. SCBench: A kv cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. Snapkv: LLM knows what you are looking for before generation. *arXiv:2404.14469*.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. 2025. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. *arXiv:2502.02770*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, et al. 2025a. A comprehensive survey on long context language modeling. *arXiv:2503.17407*.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. 2025b. Can LLMs maintain fundamental abilities under kv cache compression? *arXiv:2502.01941*.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. 2025c. Can LLMs maintain fundamental abilities under kv cache compression? *arXiv:2502.01941*.

Xiaoran Liu, Ruixiao Li, Qipeng Guo, Zhigeng Liu, Yuerong Song, Kai Lv, Hang Yan, Linlin Li, Qun Liu, and Xipeng Qiu. 2024. Reattention: Training-free infinite context with finite attention scope. *arXiv:2407.15176*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv:2501.19393*.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. 2024. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. 2022. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 5336–5358.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 784–789.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. Sparq attention: Bandwidth-efficient LLM inference. In *International Conference on Machine Learning*, pages 42558–42583. PMLR.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv:2408.03314*.

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

Bo-Hsiang Tseng, Sheng-Syun Shen, Hung-Yi Lee, and Lin-Shan Lee. 2016. Towards machine comprehension of spoken content: Initial toefl listening comprehension test by machine. *arXiv:1608.06378*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for LLMs on long-context tasks. *arXiv:2407.08454*.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. *arXiv:2309.17453*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2.5 technical report. *arXiv:2412.15115*.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. 2025. Qwen2.5-1M technical report. *arXiv:2501.15383*.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024b. Pyramidinfer: Pyramid kv cache compression for high-throughput LLM inference. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3258–3270.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. 2024. Justice or prejudice? Quantifying biases in LLM-as-a-judge. In *Neurips Safe Generative AI Workshop 2024*.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. 2024. Helmet: How to evaluate long-context language models effectively and thoroughly. *arXiv:2410.02694*.

Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. 2024. Kv cache compression, but what must we give in return? A comprehensive benchmark of long context capable approaches. In *The 2024 Conference on Empirical Methods in Natural Language Processing*.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv:2502.11089*.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C.S. Lui, and Haibo Chen. 2024. Unifying kv cache compression for large language models with leankv. *arXiv:2412.03131*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. 2025. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context LLMs. *arXiv:2502.12216*.

Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, Guanyu Feng, Xin Lv, Huanqi Cao, Xiao Chuanfu, Xingcheng Zhang, Dahua Lin, and Chao Yang. 2024. Sampleattention: Near-lossless acceleration of long context LLM inference with adaptive structured sparse attention. *arXiv:2406.15486*.

A Experimental details

A.1 Implementation Details

This section provides supplementary details on the sparse attention patterns evaluated, focusing on hyperparameter tuning and specific configurations used to achieve target sparsity levels. We tuned hyperparameters for each pattern using ablation studies on the Qwen-7B model with a 16K sequence length across all tasks, varying sparsity from 0 to 0.9. Our main experiments evaluated performance at sparsity levels 0.33, 0.5, 0.6, 0.7, 0.8, 0.87, 0.9, 0.93, 0.95 (corresponding to attention budgets $1/1.5$, $1/2$, $1/2.5$, $1/3.33$, $1/5$, $1/7.5$, $1/10$, $1/15$, $1/20$), using linear interpolation for intermediate values where necessary. Table 3 summarizes the final parameters we used for each pattern, sequence length, and sparsity level. In total we evaluated 7065 configurations with 100 samples per configuration. We used approximately 4 compute nodes with 8 H100 GPUs each for 21 days.

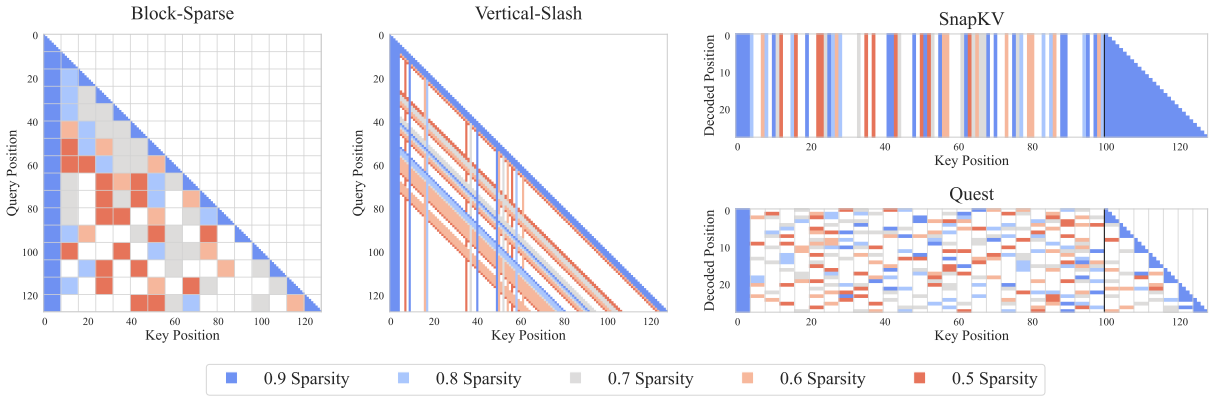

Figure 4: Visualization of sparse attention patterns. Block-Sparse and Vertical-Slash operate during prefilling (showing query-key attention matrix), while SnapKV and Quest operate during decoding (showing decoded tokens attending to KV cache positions). Colors indicate different sparsity levels from 0.5 (red) to 0.9 (blue). The black vertical lines in SnapKV and Quest mark the prefill/decode boundary.

A.1.1 Block-Sparse Attention

We implement block-sparse attention by dividing the attention matrix into fixed-size blocks. The original implementation is available under the MIT license. Based on our ablation studies (Figure 5), we selected a block size of 16x16, as smaller blocks consistently yielded better performance. To achieve a target sparsity level, we select the top-k key blocks for each query block, where k is determined via binary search. We always preserve attention sinks (the first key block) and local context (diagonal key blocks corresponding to the query block).

A.1.2 Vertical-Slash Pattern

We implement the Vertical-Slash pattern (Jiang et al., 2024), available under the MIT license, by allocating a uniform budget to global (vertical columns) and local (slash diagonals) attention components. We select the most important verticals and slashes by approximating attention scores using a limited window of recent query tokens. Our ablation studies (Figure 11) revealed task-dependent optimal approximation window sizes: 512 tokens for retrieval-heavy tasks (Ruler NIAH, Story Retrieval) and 256 tokens for other tasks. This observation correlates with the typical query lengths for these tasks (see Table 2). We consistently preserve the first 4 (prefix) and the most recent 64 (local) tokens. To achieve target sparsity levels, we compute the required number of verticals and slashes based on collected attention statistics for each sequence length.

A.1.3 FlexPrefill

We implement FlexPrefill (Lai et al., 2025), available under the Apache-2.0 license, which enhances Vertical-Slash by introducing dynamic budget allocation per layer and head, controlled by a coverage parameter α and a minimum budget (`min_budget`). We set $\tau = 0$ in our experiments, hence disabling

Table 2: Statistics of token lengths of question and instruction for each task across 100 samples, informing the choice of approximation window size for Vertical-Slash and FlexPrefill.

Task	Mean Tokens	Min Tokens	Max Tokens
QA QuALITY	243.63	196	423
QA SQuAD	217.08	210	235
QA ToeflQA	237.67	202	270
RULER CWE	227.00	227	227
RULER NIAH	337.74	330	350
RULER VT	230.00	230	230
Story Filtering	184.00	184	184
Story Multi-hop	192.97	192	195
Story Retrieval	457.54	452	462

Query-Aware attention. This choice stemmed from two key considerations: first, our preliminary tests indicated no significant performance gains from enabling it, aligning with the findings reported in the original work; second, this setting isolates the dynamic budget allocation mechanism, allowing us to specifically evaluate its impact compared to the fixed allocation used in the Vertical-Slash pattern. We employ the same task-dependent approximation windows (256 / 512 tokens) and critical token preservation strategy (first 4 prefix, most recent 64 local) as in our Vertical-Slash implementation. Our ablations (Figure 8) indicated that setting `min_budget` to 512 significantly improved performance, suggesting the importance of maintaining a minimum level of connectivity during prefilling. We achieved target compression ratios by selecting the appropriate α based on attention statistics while keeping `min_budget` fixed at 512. For high compression ratios where dynamic allocation proved less effective, we set $\alpha = 0$, effectively reverting to a uniform allocation of Vertical-Slash.

A.1.4 SnapKV

We implement SnapKV (Li et al., 2024b), available under the CC-BY 4.0 license, by compressing the Key-Value (KV) cache after the prefilling stage and applying a uniform token budget across all heads for the subsequent decoding phase. We predict token importance for decoding by computing attention scores using a window of recent query tokens (approximation window). Our ablations showed an optimal approximation window size of 256 tokens (Figure 9), with no significant task dependency observed, unlike Vertical-Slash and FlexPrefill. We smooth the calculated token importance scores using 1D average pooling with a kernel size of 21 (chosen based on Figure 10). We always preserve the first 4 and the most recent 128 tokens. We control sparsity by setting the ‘token_capacity‘ (token limit per head) to achieve the target sparsity level.

A.1.5 Ada-SnapKV

We implement Ada-SnapKV (Feng et al., 2024), available under the MIT license, which extends SnapKV by incorporating dynamic token budget allocation per head. One difference in our implementation between Ada-SnapKV and SnapKV is that we use max-aggregation (instead of averaging) across query positions and heads for score calculation; this empirically proved more effective for adaptive allocation but had no effect for uniform (SnapKV) allocation. We utilize the same smoothing kernel size (21) and critical token preservation strategy (first 4 prefix, most recent 128 local) as in our SnapKV implementation. Our ablations (Figure 7) indicated that providing each head with a minimum budget of 20% of its capacity was optimal. Performance was found to be less sensitive to minimum budget (performing well within the 10-50% range) compared to FlexPrefill’s sensitivity, but degraded sharply when approaching 100% (uniform allocation), underscoring the benefits of dynamic allocation during decoding. We control sparsity by setting ‘token_capacity‘, identically to SnapKV.

A.1.6 Quest

We implement Quest (Tang et al., 2024), available under the CC-BY 4.0 license, which applies dynamic sparse attention during the decoding phase at the page level. Based on our ablations (Figure 6), we used a page size of 16 tokens. We represent pages by their minimum and maximum key values to enable efficient similarity computation with queries. At each decoding step, we select the most relevant pages based on query-page similarity scores, always including the page containing the current token. We control sparsity by setting the 'token_budget' (number of tokens selected per step) to achieve the target sparsity level.

Table 3: Pattern parameters for different sequence lengths and sparsity levels. At 128k tokens, we only evaluate Vertical-Slash and Quest.

Pattern	Parameter	Sequence Length	Values for Different Sparsity Levels
Vertical & Slash	Verticals/Slashes	16384	164, 240, 315, 400, 448, 576, 768, 1024, 1536, 2304
		32768	290, 384, 448, 576, 704, 1024, 1536, 2304, 3584, 4608
		65536	400, 448, 544, 640, 960, 1280, 2304, 4096, 6144, 8192
		128000	480, 768, 1024, 1536, 2048, 3584, 5632, 10240, 13312, 18432
FlexPrefill	(α, min_budget)	16384	(0, 164), (0, 240), (0, 315), (0, 400), (0.55, 512), (0.71, 512), (0.88, 512)
		32768	(0, 290), (0, 384), (0.45, 512), (0.6, 512), (0.7, 512), (0.8, 512), (0.92, 512)
		65536	(0, 400), (0.45, 512), (0.55, 512), (0.7, 512), (0.77, 512), (0.85, 512), (0.94, 512)
Block Sparse	top_chunks	16384	26, 35, 53, 71, 108, 188, 300
		32768	52, 69, 105, 141, 216, 376, 600
		65536	104, 139, 210, 283, 432, 752, 1200
SnapKV/AdaSnapKV	token_capacity	16384	819, 1092, 1638, 2183, 3276, 4915, 6553, 8192, 9830, 11468
		32768	1638, 2185, 3276, 4367, 6553, 9830, 13107, 16384, 19660, 22937
		65536	3276, 4371, 6553, 8735, 13107, 19660, 26214, 32768, 39321, 45875
Quest	token_budget	16384	816, 1088, 1632, 2176, 3280, 4912, 6560, 8192, 9824, 11472
		32768	1632, 2192, 3280, 4368, 6560, 9824, 13104, 16384, 19664, 22944
		65536	3280, 4368, 6560, 8736, 13104, 19664, 26208, 32768, 39328, 45872
		128000	6400, 8544, 12800, 17056, 25600, 38400, 51200, 64000, 76800, 89600

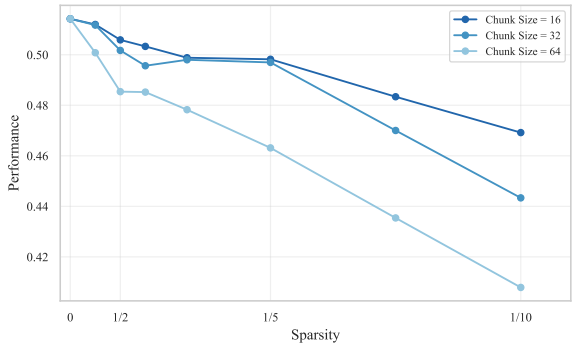

Figure 5: Block-Sparse block size.

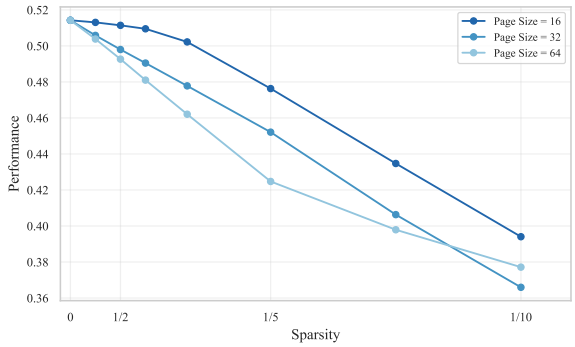

Figure 6: Quest page size.

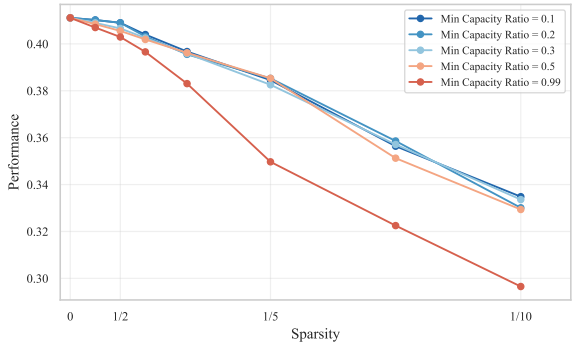

Figure 7: Ada-SnapKV min budget.

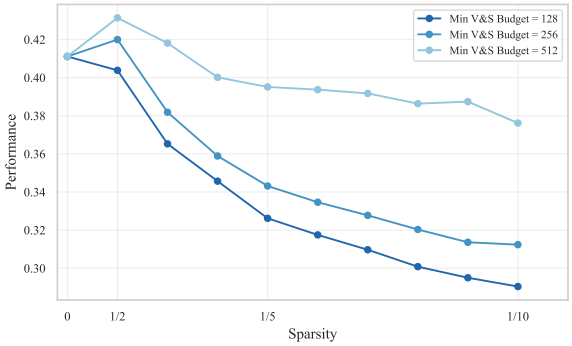

Figure 8: FlexPrefill min budget.

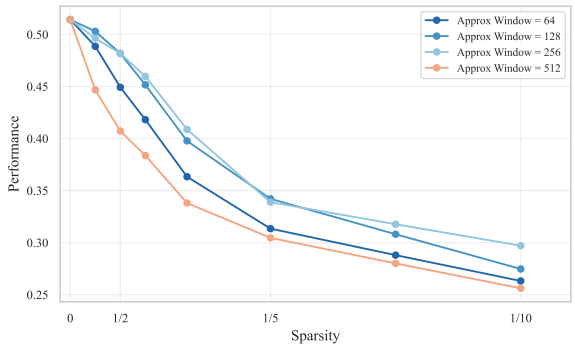

Figure 9: SnapKV/Ada-SnapKV approximation window.

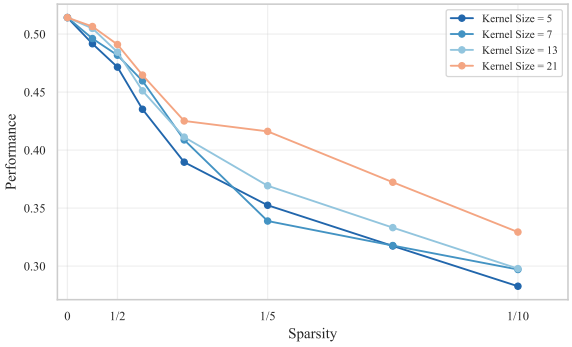

Figure 10: SnapKV/Ada-SnapKV kernel size.

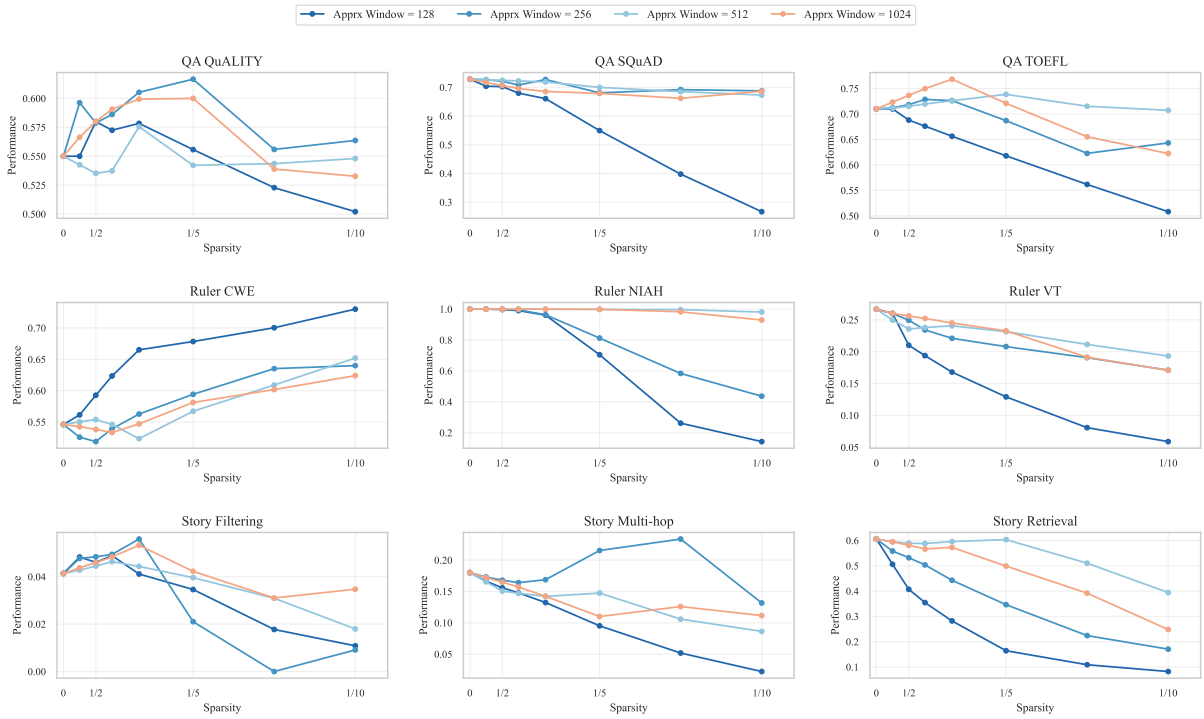

Figure 11: Vertical-Slash approximation window ablation per task.

A.2 Task Details

This section provides further details on the nine evaluation tasks used in our experiments. These tasks, summarized in Table 4 at the end of this subsection, are grouped into Question Answering, synthetic tasks from RULER (Hsieh et al., 2024), and our Story tasks. We specify key hyperparameters, evaluation metrics, and characterize each task along the axes of Scope (Low vs. High) and Dispersion (Low vs. High). Scope refers to the amount of information required, while Dispersion indicates how difficult it is to locate the relevant information within the context.

A.2.1 Question Answering (QA)

We use SQuAD (Rajpurkar et al., 2018) (CC BY-SA 4.0) from RULER (Apache-2.0) and two other QA datasets selected for minimal data contamination (Li et al., 2024a): QuALITY (Pang et al., 2022) (CC BY 4.0) and ToeflQA (Tseng et al., 2016)[4].

- **Setup:** Each example contains one answer-bearing document and distractor documents to reach the target sequence length. Documents are shuffled and numbered; the question refers to a specific document ID.
- **Preprocessing:** We remove duplicate question-context pairs and filter examples where the original context exceeds 8k tokens (ensuring space for at least one distractor at 16k sequence length).
- **Evaluation:** Exact Match Accuracy (QuALITY, ToeflQA multiple-choice), token-level F1 (SQuAD open-ended).
- **Characteristics:** Natural text. Requires identifying and processing a specific document, thus characterised by **Low Dispersion** and **Low Scope**. See Appendix G.1.

A.2.2 Synthetic – RULER Tasks

We use three synthetic tasks from the RULER benchmark (Hsieh et al., 2024) (Apache-2.0).

- **Needle-in-a-Haystack (NIAH):** Extract values for 4 target keys from a document containing relevant and distractor key-value pairs (random hyphenated strings). Evaluated using Exact Match Accuracy. Requires finding specific items, characteristic of **Low Dispersion** and **Low Scope**. See Appendix G.2.
- **Common Word Extraction (CWE):** Identify the 10 most frequent words (appearing 30 times each) among distractor words (appearing 3 times each), sampled from a vocabulary of ∼9,000 English words[5]. Evaluated using Intersection-over-Union (IoU). Requires processing the entire context to count frequencies, demanding **Low Dispersion** (words are presented directly as a list, not obscured within complex structures) but **High Scope** (all words must be processed to determine frequencies). See Appendix G.3.
- **Variable Tracking (VT):** Resolve variable assignments (direct or chained) to identify all variables matching a target value. Context includes repeated filler text ("*The grass is green...*"). Evaluated using IoU. Requires tracking dependencies across the context, demanding **High Dispersion** (information location depends on chains) and **Low Scope** (only specific chains matter). See Appendix G.4.

A.2.3 Semi-Synthetic – Story Tasks

These tasks use procedurally generated multi-chapter narratives that scale with sequence length. Each chapter follows a schema involving travel, dialogue, and item transactions. See Appendix F for an example narrative. We release them under the CC BY 4.0 license.

[4]Originally released at https://github.com/iamyuanchung/TOEFL-QA. License information is missing; a GitHub issue requesting clarification was opened on August 7, 2023, but has not received a response.

[5]https://github.com/mrmaxguns/wonderwordsmodule

- **Story Retrieval:** Answer 16 factoid questions (e.g., location visited, item acquired) about specific chapters, with chapter IDs provided in the questions. Evaluated using Exact Match Accuracy. Requires accessing specific chapters, characteristic of **Low Dispersion** and **Low Scope**. See Appendix G.5.

- **Story Filtering:** Identify the three specific chapters where no item purchases occurred. The prompt explicitly asks for these three chapter IDs, and the narrative is constructed such that exactly three chapters meet this condition. Evaluated using IoU. Requires checking all chapters, demanding **Low Dispersion** (information is chapter-based) but **High Scope** (all chapters must be checked). We found this task to be challenging even for the largest models evaluated. See Appendix G.6.

- **Story Multi-hop:** Given a target item, identify the item acquired immediately before it, requiring reasoning across the transaction history in multiple chapters. In our setup, an item is acquired in every chapter; this simplifies the task to locating the chapter where the target item was acquired and retrieving the item name from the immediately preceding chapter. We found this simplified version to be highly challenging, even for the largest models evaluated, thus we did not explore more complex variants (e.g., selective item acquisition requiring longer lookbacks). Evaluated using Exact Match Accuracy. Requires tracking history across the narrative, demanding **High Dispersion** (relevant transactions can be far apart) and **Low Scope** (only specific transaction pairs matter). See Appendix G.7.

Task Name	Description	Dispersion	Scope	Natural
QA (SQuAD)	Open-ended QA on a specified document among distractors	Low	Low	✓
QA (QuALITY, TOEFL)	Multiple-choice QA on a specified document among distractors	Low	Low	✓
Ruler NIAH	Extract 4 values for specified keys among many distractor key-value pairs	Low	Low	×
Ruler VT	Identify variables that resolve to a specific value via chained assignments	High	Low	×
Ruler CWE	Identify the 10 most frequent words from a list with distractors	Low	High	×
Story Retrieval	Answer 16 factoid-style questions about specific chapters in a long narrative	Low	Low	✓
Story Multi-hop	Identify the item acquired immediately before a target item across chapters	High	Low	✓
Story Filtering	Identify chapters where no item purchases occurred in a long narrative	Low	High	✓

Table 4: Summary of 9 evaluation tasks: QA tasks are based on existing datasets—SQuAD (Rajpurkar et al., 2018), QuALITY (Pang et al., 2022), TOEFL (Tseng et al., 2016)—while NIAH, VT, and CWE are taken from the RULER benchmark (Hsieh et al., 2024). The remaining three (Story Retrieval, Multi-hop, and Filtering) are our contribution: we automatically generate multi-chapter narratives to evaluate the same skills as RULER tasks but expressed in naturalistic text. For each task, we indicate whether it has High or Low *dispersion* (information is difficult to locate), High or Low *scope* (large amount of necessary information), and whether it is based on *natural* text or is synthetic.

A.3 Model Details

Our choice of Qwen 2.5 as the primary model family was driven by strict methodological requirements. We needed a model family satisfying three criteria simultaneously: (1) native 128k context support, since sparse attention benefits emerge primarily at very long sequences; (2) multiple model sizes maintaining reasonable (non-random) performance across all sequence lengths with consistent training procedures on identical data to enable rigorous size-based comparisons; and (3) instruction-tuned versions for chain-of-thought evaluation to mitigate short-output evaluation bias toward sparse decoding methods with dense prefill (Yuan et al., 2024).

After evaluating available open-source families, Qwen 2.5 was the only one meeting these requirements. Other families were excluded for the following reasons:

- **Command**: Different training data across sizes (8B vs 32B/104B).
- **Llama 3.1/3.2**: Smaller models (1B, 3B) failed at 16k–32k sequence length on most tasks; 405B exceeded our computational budget.
- **Mistral**: Multiple fine-tunes but fewer than three sizes with consistent training.
- **Phi-3**: The 14B model showed unexpectedly poor long-context performance, worse than 4B according to RULER evaluations.
- **Yi**: Limited to 32k sequence length.
- **GPT-OSS**: Only two model sizes; additionally requires custom attention implementation with variable attention head biases, which is not supported by training-free sparse attention methods.

To broaden our scope and provide additional evidence for our findings, we also evaluate on Llama 3.1 (8B, 70B) and Gemma 3 (4B, 12B, 27B). Gemma 3 employs hybrid attention where 5 out of 6 layers use sliding window attention with a window size of 1024 tokens, while every 6th layer uses global (dense) attention. This makes Gemma 3 particularly interesting for our study: it is already heavily sparsified by design, and we apply training-free sparse attention methods only to the dense layers. This allows us to analyze whether additional sparsification benefits models that already incorporate architectural sparsity.

Family	Size	Layers	Q Heads	KV Heads	Huggingface
Llama 3.1	8B	32	32	8	meta-llama/Llama-3.1-8B-Instruct
	70B	80	64	8	meta-llama/Llama-3.1-70B-Instruct
Qwen 2.5	7B	28	28	4	Qwen/Qwen2.5-7B-Instruct
	14B	48	40	8	Qwen/Qwen2.5-14B-Instruct
	32B	64	40	8	Qwen/Qwen2.5-32B-Instruct
	72B	80	64	8	Qwen/Qwen2.5-72B-Instruct
Gemma 3	4B	34	8	4	google/gemma-3-4b-it
	12B	48	16	8	google/gemma-3-12b-it
	27B	62	32	16	google/gemma-3-27b-it

Table 5: Overview of models used in the evaluation. Qwen 2.5 and Llama 3.1 officially support context lengths up to 128k tokens; however, we evaluate 128k only for Vertical-Slash and Quest. Gemma 3 supports up to 128k tokens but exhibited near-zero performance at this length across most configurations, so we evaluate Gemma 3 up to 64k only. We use 100 samples per configuration for Qwen and 50 for Llama and Gemma.

B Computational Cost Analysis

We analyze computational cost using implementation-agnostic metrics that correlate with wall-clock time under optimized implementations: FLOPs for prefilling (compute-bound) and memory transfers for decoding (memory-bound). This approach avoids confounds from implementation-specific inefficiencies while capturing the fundamental cost structure.

B.1 Cost Formulas

For prefilling, which is compute-bound, we compute total FLOPs as:

$$\text{FLOPS}_{\text{prefill}} = B \cdot (\text{FLOPS}_{\text{embedding}} + \text{FLOPS}_{\text{attention}} + \text{FLOPS}_{\text{mlp}} + \text{FLOPS}_{\text{logits}}) \tag{1}$$

$$\text{FLOPS}_{\text{embedding}} = 2 \cdot L \cdot d \tag{2}$$

$$\text{FLOPS}_{\text{attention}} = N \cdot (2Ld \cdot (d + 2d_h n_{kv} + d) + \rho \cdot (2hL^2 d_h + 3hL^2 + 2hL^2 d_h)) \tag{3}$$

$$\text{FLOPS}_{\text{mlp}} = N \cdot (6 \cdot L \cdot d \cdot d_{\text{mlp}} + 2 \cdot L \cdot d_{\text{mlp}}) \tag{4}$$

$$\text{FLOPS}_{\text{logits}} = 2 \cdot L \cdot d \cdot |V| \tag{5}$$

For decoding, which is memory-bound, we measure memory accesses:

$$\text{Memory}_{\text{decode}} = \text{Memory}_{\text{weights}} + B \cdot \text{Memory}_{\text{kv\_cache}} \tag{6}$$

$$\text{Memory}_{\text{kv\_cache}} = N \cdot 2 \cdot L \cdot d_h \cdot n_{kv} \cdot \rho \tag{7}$$

$$\text{Memory}_{\text{weights}} = N \cdot (4d^2 + 3d \cdot d_{\text{mlp}}) + d \cdot |V| + d \tag{8}$$

where L is sequence length, d is hidden dimension, h is number of query heads, d_h is head dimension, n_{kv} is number of key-value heads, N is number of layers, d_{mlp} is MLP intermediate dimension, $|V|$ is vocabulary size, ρ represents attention density ($1 -$ sparsity), and B is batch size.

For sparse methods, we include importance estimation overhead. Vertical-Slash indexing:

$$\text{FLOPS}_{\text{VS indexing}} = B \cdot N \cdot h \cdot [2dLq + 3Lq + 2Lq + 2L \log_2(L) + \frac{L}{64}(k_v + k_s)] \tag{9}$$

Quest indexing requires loading page representations:

$$\text{Memory}_{\text{Quest indexing}} = B \cdot N \cdot n_{kv} \cdot 2d \cdot \frac{L}{p} \tag{10}$$

where q is the number of queries for importance estimation, k_v/k_s are selected vertical/slash patterns, and p is page size (16).

B.2 Prefilling: Sequence Length Drives Sparsity Impact

Attention cost scales quadratically with sequence length ($O(L^2)$) during prefilling, while non-attention costs (MLP, embeddings, logits) scale linearly ($O(L)$). This creates two regimes: at shorter sequences, non-attention components dominate; at longer sequences, attention becomes the primary cost.

Figure 12 illustrates the practical consequence. At 16K tokens, attention represents 40% of prefilling FLOPs (averaged across Qwen 7B–72B), so 5× sparsity yields only 1.5× speedup. At 64K tokens, attention rises to 68%, yielding 2.2× speedup. At 128K tokens, attention dominates at 80%, enabling 2.8× speedup. Notably, the standard deviation across model sizes is small (±4–6%), indicating this relationship holds regardless of model scale.

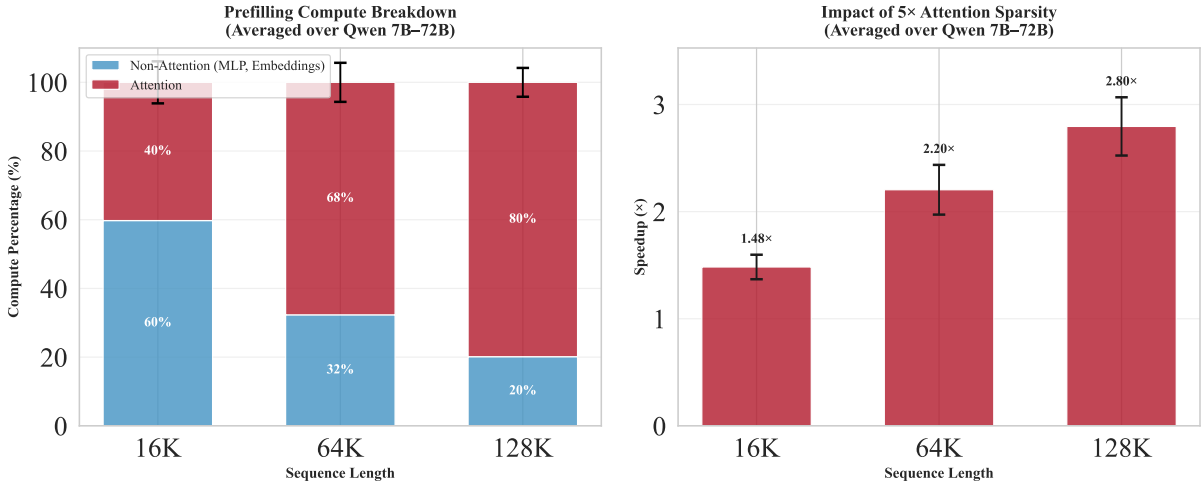

Figure 12: Prefilling compute breakdown and sparsity benefits, averaged over Qwen 7B–72B with error bars showing standard deviation across model sizes. **Left**: As sequence length increases from 16K to 128K, attention grows from 40% to 80% of total FLOPs. **Right**: Consequently, 5× attention sparsity yields progressively greater speedups—from 1.5× at 16K to 2.8× at 128K.

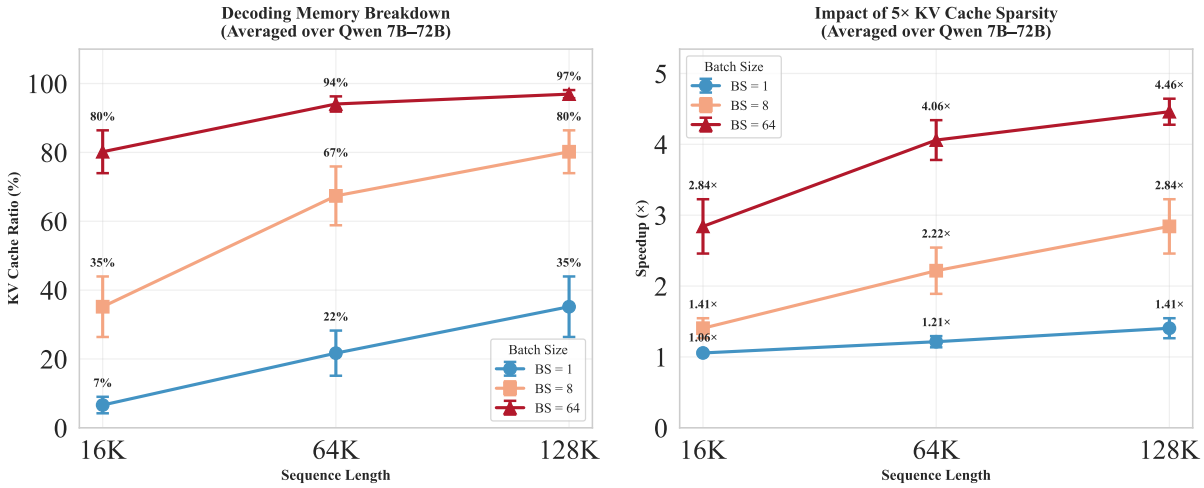

Figure 13: Decoding cost breakdown and sparsity benefits, averaged over Qwen 7B–72B with error bars showing standard deviation across model sizes. **Left**: KV cache ratio increases with both sequence length and batch size. **Right**: Corresponding speedup from 5× KV cache sparsity. At batch size 1, sparse attention provides minimal benefit. At batch size 64, speedups reach 2.8–4.7×.

B.3 Decoding: Sequence Length and Batch Size Both Matter

Unlike prefilling, decoding cost depends on both sequence length and batch size. KV cache access scales linearly with context length ($O(L)$) and batch size ($O(B)$), while model weight loading is constant. This creates an important distinction: for prefilling, all cost components scale linearly with batch size, so the attention-to-total ratio remains constant. Decoding behaves differently—model weights are loaded once per forward pass regardless of batch size, while KV cache access scales with batch size.

Figure 13 shows this clearly (averaged across Qwen 7B–72B). At batch size 1, weight loading dominates: KV cache represents only 7% at 16K tokens, rising to 35% at 128K. At batch size 8, the picture shifts: KV cache reaches 35–80%. At batch size 64, KV cache dominates at 80–97%, and sparse attention becomes highly effective with 2.8–4.7× speedups. The standard deviation across model sizes remains modest (±1–9%), confirming these trends hold across model scales.

B.4 Sliding-Window Architectures Need Longer Sequences

Figure 14: Attention cost ratio comparison between Qwen 14B (dense attention) and Gemma 12B (sliding-window attention) at batch size 8. Gemma's architectural sparsity results in substantially lower attention ratios, requiring longer sequences for additional sparse attention to provide meaningful cost reduction.

Models with built-in architectural sparsity, such as Gemma 3's sliding-window attention (5 out of 6 layers use 1024-token windows), have lower baseline attention ratios. Figure 14 compares similar-sized models: at 64K tokens with batch size 8, Qwen 14B has 76% attention ratio for prefilling versus Gemma 12B's 42%. For decoding, Qwen reaches 79% versus Gemma's 61%. At 128K tokens, Gemma's attention ratio rises to 54% for prefilling and 75% for decoding.

This difference has practical implications for isoCost comparisons. Because Gemma has a lower attention-to-total ratio at 64K, sparse prefilling reduces a smaller fraction of total FLOPs than for Qwen, so the dense–sparse frontiers overlap later (i.e., at higher costs / longer sequences). In principle, one would expect stronger overlap for Gemma at longer contexts as the attention ratio rises, but in our experiments most model–task configurations at 128K exhibit near-zero performance, preventing a meaningful prefilling comparison at that length. Decoding is less constrained: at sufficiently high batch size (e.g., $B = 64$), KV-cache transfers dominate even for sliding-window models, so sparse decoding still yields practical benefits and can exhibit dense–sparse overlap in isoCost space (see Section 4).

C Comparison to Prior Work

Several concurrent works have explored evaluation of sparse attention methods. We summarise the key differences below.

SCBench (Li et al., 2025) does not control for sequence length and evaluates at most two models from a single family, making it difficult to analyse the effects of sequence length and model size on sparse

attention performance. Our work systematically varies sequence length (16K–128K tokens) and evaluates multiple model sizes within each of three model families.

Liu et al. (2025b) only considers models up to 10B parameters and does not address sparse attention in the prefilling phase. In contrast, we evaluate models up to 72B parameters and analyse both prefilling and decoding phases separately, revealing phase-specific behaviours.

Yuan et al. (2024) tests sequence lengths only up to 32K tokens and includes models up to 10B parameters. Our evaluation extends to 128K tokens and 72B parameters, capturing the regime where sparse attention benefits are most pronounced.

In summary, our work is the first to systematically conduct an isoCost analysis for sparse attention, providing new insights into efficiency–accuracy trade-offs and generalisation across model size, sequence length, and sparsity.

D Extra Results

D.1 Statistical Error Bounds

We report standard error in Figure 2 (main text, RQ2) for our per-task and per-method results. We omit standard error bars in isocost figures (Figure 1) for visual clarity, as the standard error is negligible. Here we derive the upper bound for the standard error across all configurations.

Since performance metrics lie in the $[0, 1]$ range, the maximum standard deviation is $\sigma_{\max} = 0.5$ (achieved when the metric has a Bernoulli distribution with $p = 0.5$). For configurations where we aggregate results over N samples, the standard error is:

$$SE = \frac{\sigma}{\sqrt{N}} \leq \frac{\sigma_{\max}}{\sqrt{N}} = \frac{0.5}{\sqrt{N}} \tag{11}$$

In Figure 1, we aggregate performance across 9 tasks with 100 samples each (for Qwen), yielding $N = 900$ samples total:

$$SE_{\max} = \frac{0.5}{\sqrt{900}} = \frac{0.5}{30} \approx 0.0167 \tag{12}$$

This upper bound of approximately 0.017 is substantially smaller than the performance differences we observe between configurations (typically > 0.05), justifying our decision to omit error bars for visual clarity in the isocost analysis.

D.2 Per-Task Results by Model Family

This section provides per-task performance breakdowns for each model family, complementing the aggregated analysis in Section 4.2. Figures 15 to 17 show results for Qwen 2.5, Llama 3.1, and Gemma 3 respectively.

D.3 Sequence Length Effects

Figure 18 presents the absolute error perspective on sequence length effects, complementing the relative error analysis in Section 4.3. The absolute error is $\bar{p}_{\text{dense}} - \bar{p}_{\text{sparse}}$, where $\bar{p}$ denotes mean performance. The pattern mirrors the relative error findings: longer sequences tolerate higher sparsity with smaller absolute performance degradation.

Figure 19 provides per-family breakdowns of the sequence length analysis. The trend of improved sparsity tolerance at longer sequences holds consistently across all three model families, with minor variations in magnitude.

D.4 Model Size Analysis

We analyse how sparsity tolerance varies with model scale. Figure 20 shows model size effects aggregated across all tasks, methods, and sequence lengths for each model family. On average, model size shows no clear correlation with sparsity tolerance—the lines for different model sizes largely overlap, indicating that larger models do not systematically tolerate more or less sparsity than smaller ones.

However, this aggregate finding masks important task-dependent patterns revealed in Figure 21.

Model size effects depend on task difficulty. Figure 21 presents contrasting perspectives on model size effects. On tasks where all model sizes achieve near-perfect dense accuracy (left column: Story Retrieval, Ruler NIAH), larger models tolerate more sparsity—at sparsity 0.95 ($1/20$ budget), 72B shows 0.20 absolute error compared to 0.50 for 7B. Conversely, on challenging tasks where dense accuracy scales with model size (right column: Ruler VT, Story Filtering), larger models exhibit *larger* absolute errors at equivalent sparsity—72B shows 0.21 absolute error while 7B shows only 0.03 at sparsity 0.95. The relative error perspective (bottom row) shows consistent patterns: larger models have lower relative error on easy tasks but higher relative error on hard tasks.

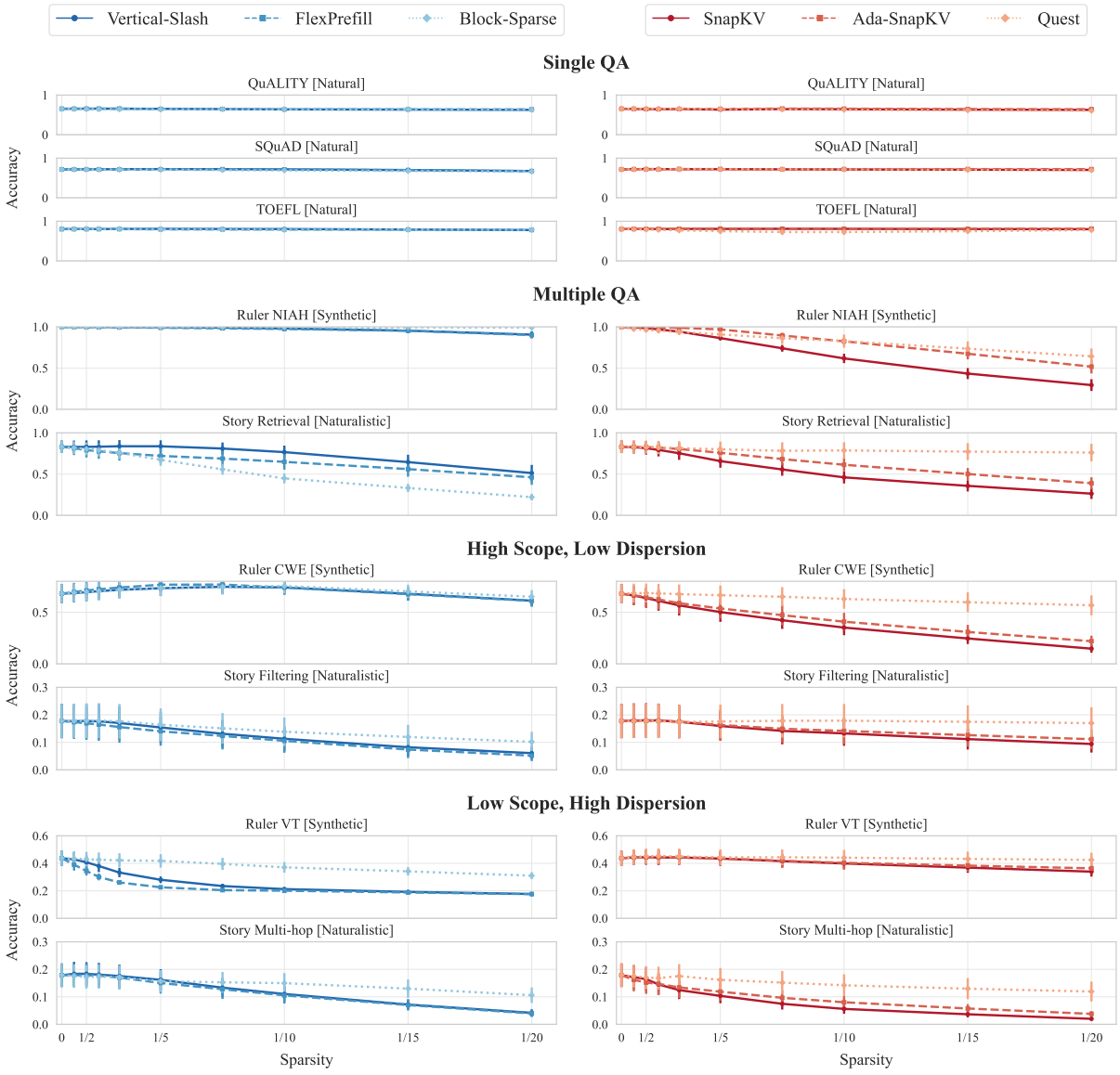

Figure 15: Per-task performance for **Qwen 2.5** models (7B, 14B, 32B, 72B) at sequence lengths 16k, 32k, and 64k. **Left**: prefilling methods. **Right**: decoding methods.

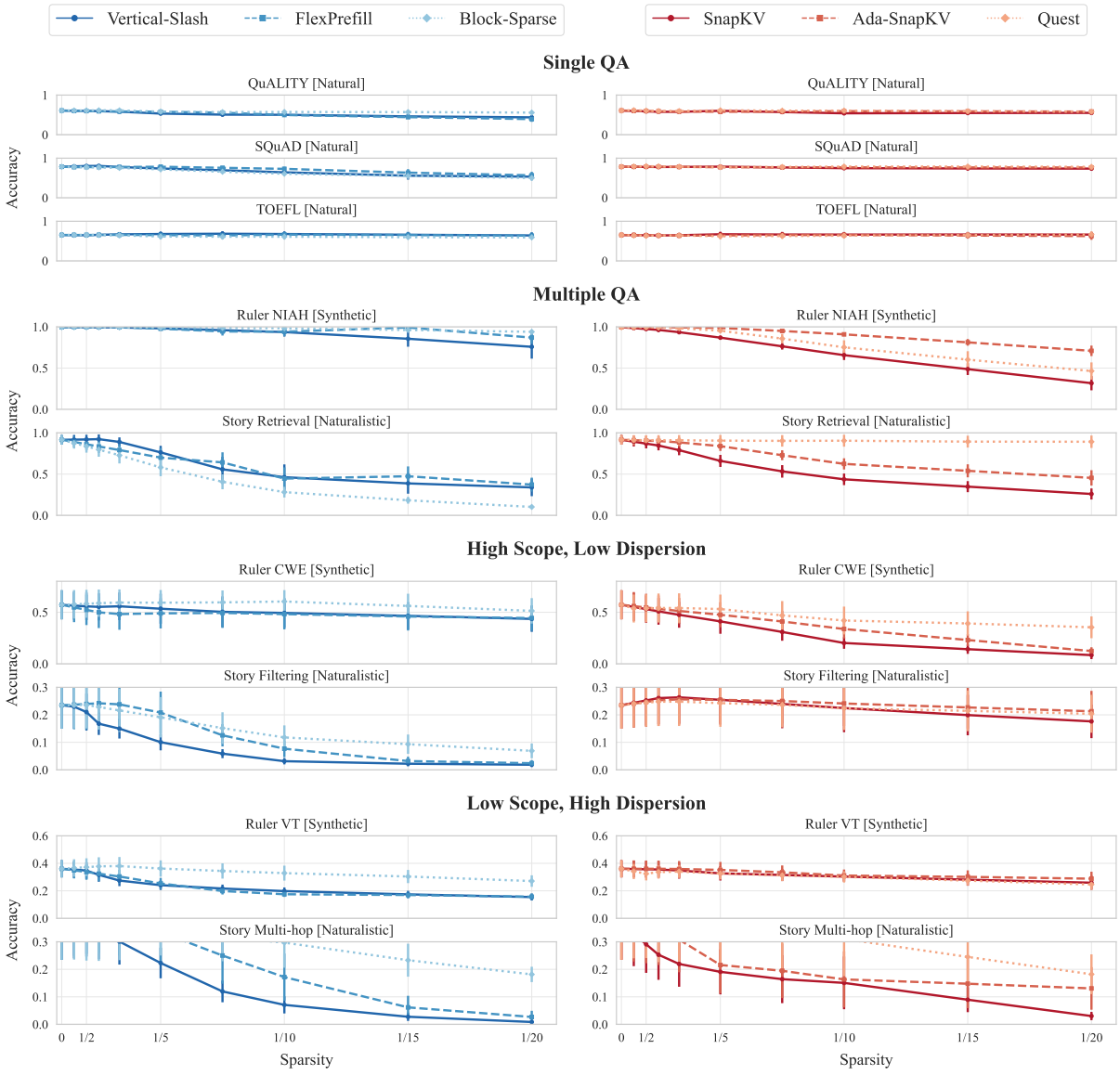

Figure 16: Per-task performance for **Llama 3.1** models (8B, 70B) at sequence lengths 16k, 32k, and 64k. **Left**: prefilling methods. **Right**: decoding methods.

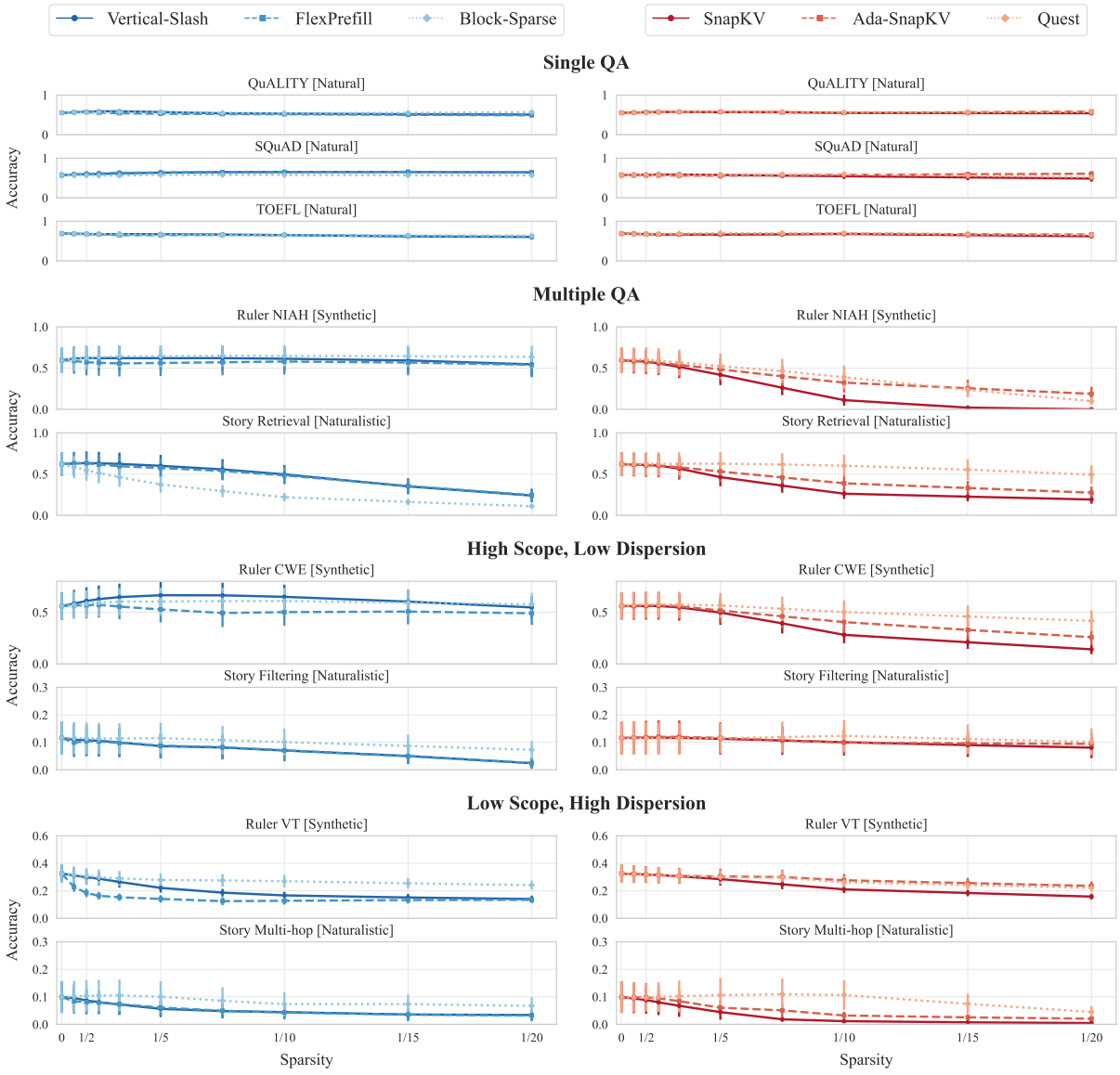

Figure 17: Per-task performance for **Gemma 3** models (4B, 12B, 27B) at sequence lengths 16k, 32k, and 64k. **Left**: prefilling methods. **Right**: decoding methods.

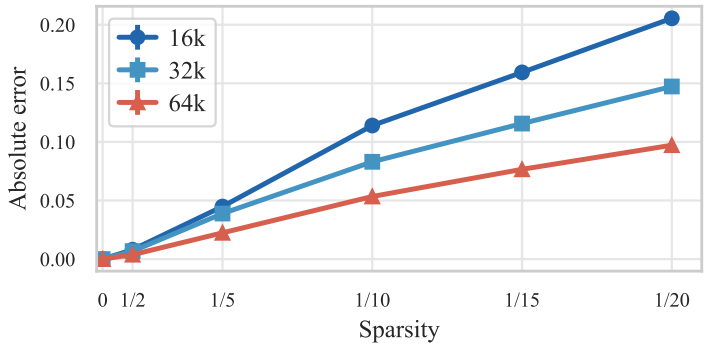

Figure 18: Absolute error vs. sparsity across sequence lengths. Results aggregated across all tasks, methods, and models (Qwen 2.5, Llama 3.1, Gemma 3).

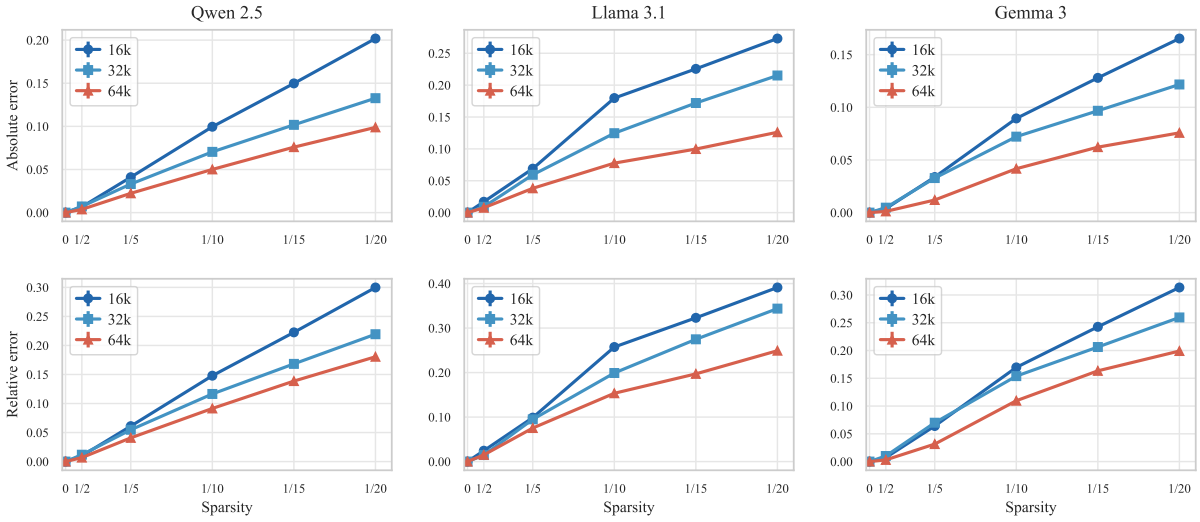

Figure 19: Sequence length effects on sparsity tolerance by model family. **Top row**: absolute error vs. sparsity. **Bottom row**: relative error vs. sparsity. Results aggregated across all tasks and methods within each family.

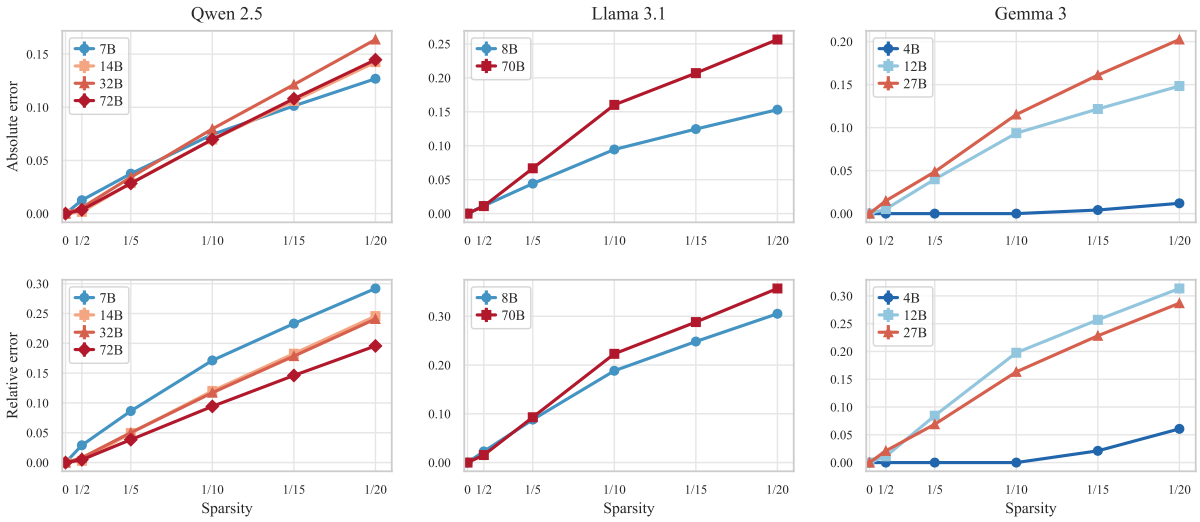

Figure 20: Model size effects on sparsity tolerance aggregated across all tasks. **Top row**: absolute error vs. sparsity. **Bottom row**: relative error vs. sparsity. Results aggregated across all tasks, methods, and sequence lengths 16–64k for each model family.

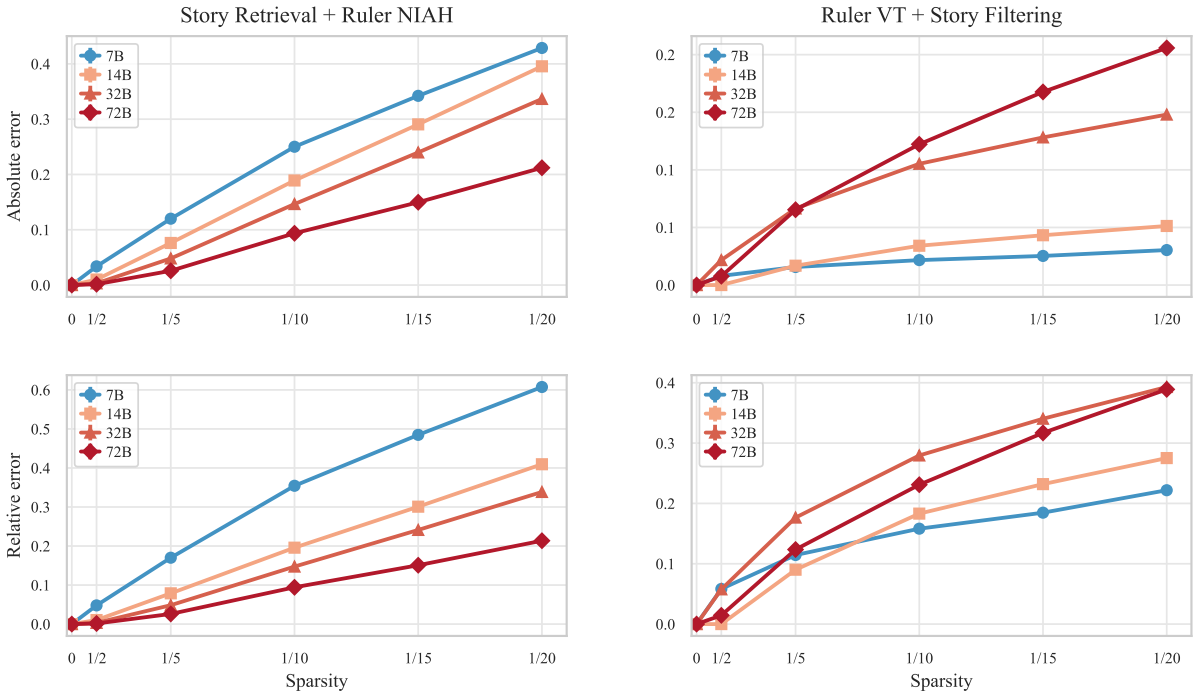

Figure 21: Model size effects on sparsity tolerance for Qwen 2.5 (7B–72B). Absolute error is $\bar{p}_{\text{dense}} - \bar{p}_{\text{sparse}}$; relative error is $(\bar{p}_{\text{dense}} - \bar{p}_{\text{sparse}})/\bar{p}_{\text{dense}}$, where $\bar{p}$ denotes mean performance. **Top row**: absolute error vs. sparsity. **Bottom row**: relative error vs. sparsity. **Left column**: easy tasks (Story Retrieval, Ruler NIAH). **Right column**: hard tasks (Ruler VT, Story Filtering). Results aggregated across methods and sequence lengths 16–64k.

These divergent patterns arise from how sparsity interacts with model capacity. Sparse attention reduces effective model capacity by limiting information flow. When a model operates far above a task's difficulty threshold, this capacity reduction has minimal impact on outputs. When model capacity approximately matches task difficulty, even modest sparsity degrades performance. Larger models achieve higher dense accuracy on difficult tasks, operating closer to their capacity limits on these tasks—making them more vulnerable to capacity reductions from sparsity. Evaluations on tasks where models achieve perfect or near-perfect accuracy—common in benchmarks like Needle-in-a-Haystack—cannot reveal these vulnerabilities.

E Prompt Template

Input format:

```
You are provided with a task introduction, context, and a question.

{task_intro}

Below is your question. I will state it both before and after the context.

<question>
{question}
</question>

<context>
{context}
</context>

<question_repeated>
{question}
</question_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
{answer_format}
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
{extra_instructions}
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

F Example Story Narrative

Chapter 1:
Beneath gentle breezes, Arion ventured into Athens, curious about its secrets. Long journeys had led Arion to Athens, a step closer to understanding. Soon enough, a tense negotiation seized everyone's attention. Cleo appeared as if expecting Arion, engaging them without delay. Carefully, they navigated the topic of old feuds, wary of awakening dormant animosities that still simmered. In a calm moment, they compared notes on the traders who passed through Athens, each leaving their subtle mark. In hushed tones, they spoke of local customs and distant rumors, sharing hints of hidden pathways. Following subtle bargaining with Cleo, Arion claimed ownership of lavish crystal lamp. With a light gesture, Arion acknowledged Cleo once more before departing. Nothing would be the same as Arion left Athens, thoughts turning inward. In quiet corners, ambitions simmered, waiting for a spark.

Chapter 2:
At dawn, Arion reached the gates of Hippo Regius, where merchants and travelers converged. This place might hold a clue Arion had long sought. Hardly had Arion arrived before a violent storm stirred uneasy whispers. Thanos approached Arion, eyes bright with opportunity. They lingered over tales of old alliances and forgotten disputes, weaving past into present. They debated the meaning of recent events, each seeking patterns in the chaos. Their reflections turned to the interplay of supply and demand, seeing how fortunes might turn in an instant. After reaching terms with Thanos, Arion took possession of ceremonial gold seal. Arion turned from Thanos, ready to move on. In parting, Arion acknowledged that the journey still had far to run. Hidden corners of the city promised knowledge or peril.

Chapter 3:
The threshold of Emerita Augusta welcomed Arion, who felt the weight of untold stories. Arion came here hoping to learn something new, or perhaps gain an advantage. Within hours, a violent storm disrupted the familiar routines. There, Arion encountered Niko, who seemed eager to exchange words or goods. Their words lingered on rumors of distant lands, where fortunes or ruin awaited bold seekers. They debated the meaning of recent events, each seeking patterns in the chaos. Their dialogue danced around subtle clues, each suggestion hinting at treasures undiscovered. The transaction concluded with Arion acquiring delicate porcelain sword from Niko. With a light gesture, Arion acknowledged Niko once more before departing. Eventually, Arion moved on, carrying new impressions forward. The distant hum of voices hinted at unseen deals.

Chapter 4:
Under fading daylight, Arion set foot in Berenice, eager to learn what it offered. A quiet determination brought Arion to Berenice, ever searching for meaning. a sudden market crash cast its shadow over Berenice, changing plans and minds. Roxana approached Arion, eyes bright with opportunity. Together, they reflected on the nature of trust and deceit, aware that fate often twists. They compared accounts of strange visitors bearing knowledge or confusion, each arrival a new riddle in Berenice. A short exchange revealed uncharted corners of Berenice, where knowledge or secrets might dwell. mystic bronze lamp changed hands as Arion completed the purchase from Roxana. Arion handed over lavish crystal lamp to Roxana as the deal closed. With a light gesture, Arion acknowledged Roxana once more before departing. As Arion prepared to depart, the path ahead remained uncertain but compelling. Somewhere, a whisper promised answers for those who dared.

Chapter 5:
Under fading daylight, Arion set foot in Syracuse, eager to learn what it offered. In pursuit of truth, Arion looked to Syracuse for subtle revelations. Not long after arriving, an opulent banquet shook the local order. Phaedra appeared as if expecting Arion, engaging them without delay. Their words traced over delicate negotiations that had once sealed lasting truces in Syracuse. Carefully, they navigated the topic of old feuds, wary of awakening dormant animosities that still simmered. They delved into the subtle art of earning trust in a place where trust was scarce and hard-won. With measured consideration, Arion purchased engraved emerald goblet from Phaedra, examining it closely. In quiet understanding, Arion left Phaedra, their paths diverging. In parting, Arion acknowledged that the journey still had far to run. A subtle tension lingered, as though fate held its breath.

G Example Task Inputs

G.1 Question Answering (QA)

Input format:

```
I will provide you with multiple documents and ask you a question about one specific document.

Below is your question. I will state it both before and after the context.

<question>
Question about document 39:
Who works to get workers higher compensation?
</question>

<context>
Document 1:
[...text omitted...]

Document 39:
Jobs with high demand and low supply pay more. Professional and labor organizations
can raise wages by limiting worker supply and using collective bargaining or political influence.

Document 47:
[...text omitted...]
</context>

<question_repeated>
Question about document 39:
Who works to get workers higher compensation?
</question_repeated>

Instructions:
1. Provide a brief explanation of your reasoning process.
2. Then, give your final answer in this format:
<answer>
Your answer here...
</answer>

Your response must follow this structure:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- Do not use complete sentences in the answer.
- For dates: Include ONLY the COMPLETE date if specifically asked.
- For locations: Use the shortest unambiguous form (e.g., 'New York' not 'New York City').
- For comparisons: State ONLY the answer that matches the criteria
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I found the relevant sentence in document 39, which states that professional and labor
organizations help increase wages using bargaining and political means.
</explanation>
<answer>
Professional and labor organizations
</answer>
```

G.2 RULER - Needle-in-a-Haystack (NIAH)

Input format:

```
I will provide you with a document containing multiple key-value pairs.
Your task is to extract specific values associated with given keys.

Below are your questions. I will state them both before and after the context.

<questions>
Extract the values for the following keys:
key-A, key-B, key-C, key-D
</questions>

<context>
The value for key-A is: value-A.
The value for key-X is: value-X.
The value for key-B is: value-B.
The value for key-Y is: value-Y.
The value for key-C is: value-C.
The value for key-Z is: value-Z.
The value for key-D is: value-D.
</context>

<questions_repeated>
Extract the values for the following keys:
key-A, key-B, key-C, key-D
</questions_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
1. The answer for <key1> is <value1>.
2. The answer for <key2> is <value2>.
etc.
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- Provide answers in the exact order of the requested keys
- Each answer must follow the format: "<number>. The answer for <key> is <value>."
- Ensure exact key matches - do not modify or paraphrase the keys
- Values must match exactly as they appear in the document
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I scanned the context for exact matches of the requested keys. For each key, I extracted
the value as stated directly after the pattern "The value for key-X is: ...".
</explanation>
<answer>
1. The answer for key-A is value-A.
2. The answer for key-B is value-B.
3. The answer for key-C is value-C.
4. The answer for key-D is value-D.
</answer>
```

G.3 RULER - Common Word Extraction (CWE)

Input format:

```
You will be given a numbered list of words. Your task is to identify the most
frequently occurring words. You should solve this task by carefully reading
and analyzing the word list. Do not attempt to write code or use programming
tools to count frequencies. This is a test of your ability to track word
frequencies directly.

Below is your question. I will state it both before and after the context.

<question>
The list contains exactly 10 words that appear 30 times each.
All other words appear 3 times each.
Your task is to identify the 10 words that appear 30 times each.
</question>

<context>
1. alpha
2. beta
3. gamma
4. delta
5. alpha
6. epsilon
...
[...list continues with randomized repeated words...]
...
N. gamma
</context>

<question_repeated>
The list contains exactly 10 words that appear 30 times each.
All other words appear 3 times each.
Your task is to identify the 10 words that appear 30 times each.
</question_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process.
   Explain how you identified the relevant information from the context
   and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
1. word_one
2. word_two
...
10. word_ten
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- List exactly 10 words, one per line, numbered from 1 to 10.
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I scanned the word list and tracked the frequency of each word.
The following 10 words appeared 30 times each, which I confirmed by careful counting.
</explanation>
<answer>
1. diligent
2. ash
3. pour
4. chateau
5. marble
6. laparoscope
7. grub
8. vinyl
9. mobility
10. kettledrum
</answer>
```

G.4 RULER - Variable Tracking (VT)

Input format:

```
I will provide you with a text containing variable assignments. The text contains two types of assignments:
1. Numeric assignments that set a variable to a number (e.g., "VAR ABC = 12345")
2. Copy assignments that set a variable equal to another variable (e.g., "VAR XYZ = VAR ABC")
Variables are sequences of uppercase letters. The assignments can appear in any order in the text.

Below is your question. I will state it both before and after the context.

<question>
Which variables resolve to the value 41015? A variable resolves to 41015 if it is either directly assigned
41015, or assigned to another variable that resolves to 41015.
</question>

<context>
VAR A = VAR B
VAR B = 41015
VAR C = VAR D
VAR D = VAR B
VAR E = 12345
VAR F = VAR G
VAR G = VAR H
VAR H = VAR B
</context>

<question_repeated>
Which variables resolve to the value 41015? A variable resolves to 41015 if it is either directly assigned
41015, or assigned to another variable that resolves to 41015.
</question_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
VARIABLE_ONE VARIABLE_TWO etc.
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- List ONLY the variable names that resolve to the target value.
- Variables can be listed in any order.
- Do not include "VAR" prefix in your answer. Do not include punctuation.
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I traced each variable assignment to see if it leads to the value 41015. B is directly assigned 41015.
A, D, and H point to B. C and G point to D and H, respectively. So A B C D G H resolve to 41015.
</explanation>
<answer>
A B C D G H
</answer>
```

G.5 Story Retrieval

Input format:

```
You are given a narrative composed of multiple chapters. Throughout these chapters, the
protagonist travels between different locations, meets various characters, and engages
in trading activities. All items mentioned in the narrative are unique, and their
ownership can change through trades. Your task is to carefully read the narrative and
answer the questions based on the provided information.

Below are your questions. I will state them both before and after the context.

<questions>
1. In Chapter 3, which character did the protagonist interact with?
2. In Chapter 5, which specific item was acquired by the protagonist?
3. In Chapter 7, which specific location did the protagonist visit?
</questions>

<context>
Chapter 1:
[...text omitted...]

Chapter 3:
Arion entered Babylon and met Thanos. After exchanging stories, Arion acquired a silver idol.

Chapter 5:
In Berenice Troglodytica, Arion encountered Xanthe and traded for a golden vase.

Chapter 7:
Delphi welcomed Arion with quiet mystery. A meeting with Vitalis ended with a jade idol.
</context>

<questions_repeated>
1. In Chapter 3, which character did the protagonist interact with?
2. In Chapter 5, which specific item was acquired by the protagonist?
3. In Chapter 7, which specific location did the protagonist visit?
</questions_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
1. ANSWER_ONE
2. ANSWER_TWO
etc.
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- For answers, use one line per answer with the number prefix
- Do not include articles like 'the' or 'a' in answers
- Answers should be specific names/items/locations mentioned in the text
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I located Chapter 3 in the context and identified Thanos as the mentioned character.
In Chapter 5, Arion acquired a golden vase from Xanthe.
Chapter 7 stated that Arion visited Delphi, so I used that as the answer.
</explanation>
<answer>
1. Thanos
2. Golden Vase
3. Delphi
</answer>
```

G.6 Story Filtering

Input format:

```
You are given a narrative composed of multiple chapters. Throughout these chapters, the
protagonist travels between different locations, meets various characters, and engages
in trading activities. All items mentioned in the narrative are unique, and their
ownership can change through trades. Your task is to carefully read the narrative and
answer the questions based on the provided information.

Below is your question. I will state it both before and after the context.

<question>
Identify all chapters where the protagonist did not buy any item.
Note: There are exactly 2 chapters without any purchases.
</question>

<context>
Chapter 1:
[... Arion visits Athens and purchases a crystal lamp ...]

Chapter 2:
[... Arion travels to Hippo Regius and buys a gold seal ...]

Chapter 3:
[... Arion enters Babylon and engages in an ongoing event but do not buy anything ...]

Chapter 4:
[... Arion arrives in Pergamon and has conversations, but no purchases are mentioned ...]

Chapter 5:
[... Arion goes to Delphi and buys a jade idol ...]
</context>

<question_repeated>
Identify all chapters where the protagonist did not buy any item.
Note: There are exactly 2 chapters without any purchases.
</question_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
chapter_id_1, chapter_id_2, ...
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- In the answer section, provide only the chapter IDs separated by commas.
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I scanned each chapter to check whether a purchase by the protagonist was explicitly
described. In Chapter 3 and 4, no item acquisition are mentioned. Other chapters include
phrases like "Arion purchased" or "Arion acquired", indicating a transaction.
</explanation>
<answer>
3, 4
</answer>
```

G.7 Story Multi-hop

Input format:

```
You are given a narrative composed of multiple chapters. Throughout these chapters,
the protagonist travels between different locations, meets various characters,
and engages in trading activities. All items mentioned in the narrative are unique,
and their ownership can change through trades. Your task is to carefully read the
narrative and answer the questions based on the provided information.

Below is your question. I will state it both before and after the context.

<question>
What was the last item that the protagonist acquired before acquiring timeworn amber sword?
</question>

<context>
Chapter 1:
[... narrative text omitted for brevity ...]

Chapter 17:
The transaction concluded with Arion acquiring pristine bronze seal from Damon.

Chapter 18:
After reaching terms with Marcus, Arion took possession of timeworn amber sword.
</context>

<question_repeated>
What was the last item that the protagonist acquired before acquiring timeworn amber sword?
</question_repeated>

Instructions:
1. First, provide a brief explanation of your reasoning process. Explain how you identified
   the relevant information from the context and how you determined your answer.
2. Then, provide your final answer following this exact format:
<answer>
ITEM_NAME
</answer>

Your response must follow this structure exactly:
<explanation>
Your explanation here...
</explanation>
<answer>
Your answer here...
</answer>

Important:
- Provide only the item name in the answer section.
- Do not include articles like 'the' or 'a' in your answer.
- The item name must be exactly as mentioned in the text.
- Keep your explanations clear, coherent, concise, and to the point.
- Do not include any additional text, explanations, or reasoning in the answer section.
```

Example answer:

```
<explanation>
I located the chapter where the protagonist acquired the timeworn amber sword.
Then, I scanned earlier chapters to find the most recent prior acquisition,
which occurred in Chapter 17 with the item pristine bronze seal.
</explanation>
<answer>
pristine bronze seal
</answer>
```

H Use of AI Assistants

We used Claude Opus 4.5 for grammar and style suggestions during the writing of this paper. All scientific content, analysis, and conclusions are the authors' own work.