DELTA: Dynamic Layer-Aware Token Attention for Efficient Long-Context Reasoning

Hossein Entezari Zarch, Lei Gao, Chaoyi Jiang, Murali Annavaram
University of Southern California
{entezari, leig, chaoyij, annavara}@usc.edu

Abstract

Large reasoning models (LRMs) achieve state-of-the-art performance on challenging benchmarks by generating long chains of intermediate steps, but their inference cost is dominated by decoding, where each new token must attend to the entire growing sequence. One approach to reduce this latency is to evict entries from the key-value (KV) cache, thereby reducing the active context used in attention computation. However, such sparse attention methods suffer from severe accuracy degradation on reasoning tasks due to cumulative selection errors and the evolving importance of tokens over long derivations. We present **DELTA**, a training-free sparse attention mechanism that improves computational efficiency without sacrificing model accuracy. DELTA partitions transformer layers into three groups: initial layers that use full attention, a small set of Δ-*layers* that identify salient tokens via aggregated head-level attention scores, and subsequent sparse-attention layers that attend only to the selected subset. This design preserves the full KV cache in GPU memory for accuracy, while avoiding expensive full-attention computation over many layers. On reasoning benchmarks such as AIME and GPQA-Diamond, DELTA matches or surpasses full attention in accuracy, while reducing the number of attended tokens by up to $4.25\times$ and delivering $1.54\times$ end-to-end speedup. Our results show that selective reuse of intermediate attention maps offers a robust path toward efficient long-context reasoning. The code is available at https://github.com/hoenza/DELTA.

1 Introduction

Recent progress in large language models (LLMs) has led to systems with impressive capabilities in reasoning and self-reflection. Large reasoning models (LRMs) such as DeepSeek-R1 (Guo et al., 2025), Gemini-2.5-pro (Google DeepMind, 2025), OpenAI-o3 (OpenAI, 2025b), Qwen3 (Yang et al., 2025), and GPT-OSS (OpenAI, 2025a) leverage test-time scaling by generating long chains of intermediate reasoning steps, significantly improving accuracy on challenging benchmarks (AIME, 2025; Rein et al., 2024; Hendrycks et al., 2021; Wei et al., 2022). However, serving such models efficiently remains difficult due to severe memory and compute bottlenecks in attention operation (Vaswani et al., 2017), especially under long-context generation settings (Dao, 2023; Ye et al., 2025).

LLM inference consists of two stages: *prefilling* and *decoding*. In the prefilling stage, the model processes the prompt, computes hidden representations, and materializes all key–value (KV) vectors as a KV cache in GPU high-bandwidth memory (HBM). During decoding, tokens are generated autoregressively: for each new token, the model computes its KV vectors, appends them to the cache in HBM, and attends over the entire history to produce the next output. Because the KV cache grows linearly with sequence length and batch size (Kwon et al., 2023), the amount of data that must be read from HBM increases rapidly. For instance, with a 32K-token context and a batch size of 128, the KV cache of Llama-3-8B in float16 already exceeds 500 GB.[1] Unlike the prefilling stage, which writes the KV cache once, the decoding stage must repeatedly stream all previously stored KV entries from HBM for every new token. This makes decoding inherently *memory-bandwidth bound*: throughput is limited by the cost of moving hundreds of gigabytes of KV data per step. As context length or batch size grows, this bandwidth pressure scales linearly, quickly overwhelming GPU memory systems and severely constraining long-context inference.

These bandwidth limitations are particularly acute for reasoning workloads. Unlike typical NLP tasks that involve long inputs but short outputs, reasoning problems often begin with concise prompts

[1] Computed as Layers × Sequence Length × Batch Size × KV Heads × Head Dim × 2 (for K&V) × 2 bytes = $32 \times 32\text{K} \times 128 \times 8 \times 128 \times 2 \approx 512$ GB.

Findings of the Association for Computational Linguistics: ACL 2026, pages 11502–11518
July 2-7, 2026

yet require lengthy derivations spanning tens of thousands of tokens. This decode-heavy profile magnifies the bandwidth bottleneck, as each step involves scanning ever-larger KV caches. As a result, the decoding stage dominates both latency and resource usage: for example, using full attention in HuggingFace, DeepSeek-R1-Distill-Llama-8B requires more than 15 minutes on a single NVIDIA A100 GPU to generate 32K tokens for one AIME problem (Yue et al., 2025). Optimizing the decoding stage is therefore essential for efficient LLM serving in reasoning applications.

Meanwhile, the unique structure of reasoning workloads opens new opportunities for efficiency. While prefilling benefits from full attention to capture global context, the much longer decoding phase is well-suited to sparsity. Sparse attention reduces computation and bandwidth requirements by restricting reads to a subset of salient tokens rather than scanning the entire KV cache. Prior work has explored two complementary directions: selection-based methods (Tang et al., 2024; Hao et al., 2025; Liu et al., 2024a; Gao et al., 2025; Yuan et al., 2025; Yang et al., 2024), which preserve the full KV cache but attend only to chosen tokens, and eviction-based methods (Hu et al., 2025; Li et al., 2024; Xiao et al., 2023b; Zhang et al., 2023; Adnan et al., 2024; Cai et al., 2025), which permanently discard unselected tokens to reduce storage cost of KV cache. Both rely on identifying important tokens using predefined criteria, and together they demonstrate the potential of sparse attention as a foundation for efficient long-decode inference.

However, applying sparse attention to long reasoning generations remains challenging. Unlike standard generation tasks, where some information loss can be tolerated, step-by-step reasoning demands that critical context be preserved throughout the entire derivation to maintain logical consistency (Hu et al., 2025). In practice, accuracy drops sharply when token selection errors accumulate over long sequences (Gao et al., 2025). Eviction-based methods such as RaaS (Hu et al., 2025) illustrate this issue: by permanently removing tokens judged less important, they risk discarding tokens that later become essential once the generation length grows beyond the KV cache capacity. The core difficulty is twofold: (1) attention patterns evolve over time, and (2) a token's importance can change, so tokens that seem irrelevant early may become highly influential later in the reasoning process.

At the same time, we make two key observations. First, attention maps across consecutive layers exhibit strong correlation: within a local block of layers, the first layer often predicts the important tokens for subsequent layers with high reliability. Second, attention distributions change gradually during decoding, which suggests that token importance can be predicted using intermediate layers without computing full attention everywhere. These insights highlight both the risk of aggressive eviction and the opportunity for accurate, low-cost selection.

To address the accuracy–efficiency tradeoff, we introduce **DELTA**, a training-free, selection-based sparse attention mechanism. DELTA preserves the full KV cache but restricts computation to a carefully chosen subset of tokens at each decoding step. It operates as a plug-and-play module that leverages the full attention maps of a small set of intermediate layers to predict the salient tokens for the upcoming layers. By selecting tokens using head-wise attention signals together with a stable recency window, DELTA significantly reduces the runtime cost of attention without incurring noticeable accuracy degradation. In summary, our contributions are:

- We provide a detailed token-level analysis of attention distributions in large reasoning models, revealing two properties: (1) strong correlation of attention patterns across consecutive layers, and (2) gradual but ongoing shifts in token importance during long generations.
- We propose **DELTA**, a training-free sparse attention mechanism that combines a head-aware token scoring rule with a stable recency window to retain the recent context most critical for reasoning.
- We demonstrate that DELTA achieves accuracy on par with, or better than, full attention on challenging reasoning benchmarks, while delivering up to $1.54\times$ end-to-end speedups. Compared with state-of-the-art sparse attention methods, DELTA reduces the number of attended tokens by up to $4.25\times$, all without sacrificing accuracy.

2 Background

LLM inference process. Decoder-only LLMs generate tokens auto-regressively in two stages: the prefilling stage and the decoding stage.

Prefilling stage. At layer i, the input hidden states are $X^i \in \mathbb{R}^{b\times s\times h}$, where b is the batch size, s is the prompt length, and h is the embedding dimension. Queries, keys, and values are projected as

$$Q^i = X^i W_Q^i; K^i = X^i W_K^i; V^i = X^i W_V^i \quad (1)$$

where

$$W_Q^i \in \mathbb{R}^{h\times h}, \quad W_K^i, W_V^i \in \mathbb{R}^{h\times(g\cdot d_{\text{head}})}.$$

Here m denotes the number of query heads, $g \le m$ the number of KV groups, and $d_{\text{head}} = h/m$ the per-head dimension.

In grouped-query attention (GQA), the queries are divided into m query heads,

$$Q^i = [Q_1^i, \ldots, Q_m^i], \quad Q_j^i \in \mathbb{R}^{b\times s\times d_{\text{head}}}, \quad (2)$$

while the keys and values are divided into only g groups,

$$K^i = [K_1^i, \ldots, K_g^i], \quad V^i = [V_1^i, \ldots, V_g^i], \quad K_\ell^i, V_\ell^i \in \mathbb{R}^{b\times s\times d_{\text{head}}}. \quad (3)$$

Each query head j is assigned to one KV group $\phi(j) \in \{1, \ldots, g\}$. GQA generalizes standard attention mechanisms: when $g = m$, it reduces to multi-head attention (MHA), and when $g = 1$, it reduces to multi-query attention (MQA).

The scaled dot-product attention for head j is

$$A_j^i = \frac{Q_j^i (K_{\phi(j)}^i)^\top}{\sqrt{d_{\text{head}}}}, \quad O_j^i = \text{softmax}(A_j^i) V_{\phi(j)}^i, \quad (4)$$

where A_j^i are the attention scores and $O_j^i \in \mathbb{R}^{b\times s\times d_{\text{head}}}$ is the head output.

The outputs of all query heads are concatenated and linearly projected:

$$O^i = [O_1^i, \ldots, O_m^i] W_O^i, \quad W_O^i \in \mathbb{R}^{h\times h}. \quad (5)$$

A feed-forward network (FFN) follows the GQA block:

$$X^{i+1} = \sigma(O^i W_1^i) W_2^i, \quad (6)$$

where $W_1^i \in \mathbb{R}^{h\times d_{\text{FFN}}}$, $W_2^i \in \mathbb{R}^{d_{\text{FFN}}\times h}$, and $\sigma(\cdot)$ is a non-linear activation.

Decoding stage. At step t, each layer receives a single token embedding $x^i \in \mathbb{R}^{b\times 1\times h}$. The new key and value are concatenated to the cached ones:

$$K^i \leftarrow [\, K^i;\ x^i W_K^i \,], V^i \leftarrow [\, V^i;\ x^i W_V^i \,]. \quad (7)$$

The subsequent GQA and FFN computations mirror the prefilling stage.

Decode cost and memory I/O. While prefilling writes the KV cache once, decoding must repeatedly read all past K/V entries for each new token, making long-context inference inherently memory-bandwidth bound. Prior work reports that decoding dominates end-to-end latency under long contexts and that KV memory movement constitutes a major fraction of decode time, underscoring the need to reduce KV reads without sacrificing accuracy (Kwon et al., 2023; Dao, 2023).

Sparse attention. Full attention requires each query to attend to all past tokens, which scales linearly with sequence length. For clarity, the sparse-selection formulation below is written for a single decoding query, and we therefore omit the batch dimension. Sparse attention reduces this cost by restricting computation to a subset of k tokens. For head j, let the exact attention weights be

$$\alpha_j^i = \text{softmax}(A_j^i) \in \mathbb{R}^s.$$

Instead of attending to all s tokens, we select an index set $\rho \subseteq \{1, \ldots, s\}$ with $|\rho| = k$. Since computing ρ from α_j^i directly is expensive, practical methods rely on an approximation function f that predicts which tokens are likely to have high attention:

$$\rho = \arg\max_{\rho' : |\rho'| = k} f(Q_j^i, K_{\phi(j)}^i, V_{\phi(j)}^i, \rho'). \quad (8)$$

The quality of the selection is measured by the *attention recall*, defined as the fraction of the ground-truth attention mass preserved in the selected subset:

$$R_j^i = \frac{\sum_{u\in\rho} \alpha_j^i(u)}{\sum_{u=1}^s \alpha_j^i(u)}. \quad (9)$$

Maximizing R_j^i under the budget constraint k is the central objective of sparse attention methods, ensuring efficiency while maintaining accuracy.

Sparsity and query dependence. Self-attention exhibits substantial sparsity beyond the earliest layers: a small subset of critical tokens typically accumulates most attention mass, enabling accurate computation on a reduced context. However, criticality is strongly *query dependent*: the tokens that matter vary with the current query vector Q, and may change rapidly across consecutive decode steps. Heuristics based only on past usage (eviction) risk losing later-salient tokens, whereas query-aware selection retains high recall under long reasoning traces (Tang et al., 2024; Zhang et al., 2023; Ge et al., 2023).

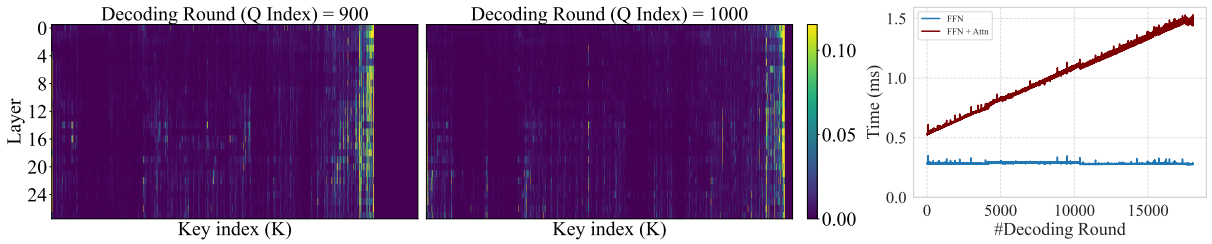

Figure 1: (Left) Attention maps from Qwen-7B at decoding steps 900 and 1000, where each row corresponds to a Transformer layer. (Right) Decoding runtime of FFN and attention modules across generation, showing attention's linear growth with context length.

3 Motivating Observation

Depth-wise context sharpening. Figure 1 (left) illustrates how attention patterns evolve with depth. In early layers, the model primarily attends to nearby tokens and exhibits diffuse, low-mass attention over the broader context, showing little focus on distant information. As depth increases, attention becomes progressively sharper and more selective, concentrating on a small set of far-away tokens that carry high relevance.

Layer-wise correlation. Empirical profiling of large reasoning models such as Qwen-7B reveals that consecutive layers exhibit highly correlated attention patterns. Tokens that receive high attention in one layer tend to remain salient in the next layers, as illustrated in Figure 1 (left), which visualizes layer-wise attention maps at decoding steps 900 and 1000 of a reasoning sequence. Each row corresponds to a Transformer layer, showing that deeper layers largely preserve the spatial configuration of attention established in earlier layers. This structural continuity suggests that adjacent layers refine rather than reconstruct attention, enabling later layers to reuse the relational patterns captured by their predecessors. As a result, computing full attention in every layer becomes redundant: once salient tokens are identified, subsequent layers can effectively operate on a reduced, high-recall subset of the context.

Sequential drift. While the overall structure of attention is stable across depth, it evolves gradually along the decoding trajectory. Between decoding steps 900 and 1000 in Figure 1, the regions of strongest attention shift across key positions, revealing how the model dynamically repositions its focus as new tokens are generated. We refer to this phenomenon as *sequential drift*.

This progressive movement reflects an adaptive retrieval process, where the model continuously updates which parts of the context are relevant to the current query embedding Q_t. Such behavior highlights the need for a *query-adaptive* sparse attention mechanism that dynamically adjusts token selection at each decoding step rather than relying on fixed or history-based heuristics.

Runtime dominated by attention. Figure 1 (right) presents the measured decoding runtimes of FFN and attention modules. While the FFN cost remains nearly constant, attention latency increases almost linearly with context length. Beyond 8k tokens, attention dominates total inference time, driven primarily by repeated KV-cache memory access rather than compute operations. These trends confirm that long-context decoding is bottlenecked by attention and motivate our approach: performing full attention only in a few strategically chosen layers to identify salient tokens, while letting the remaining layers operate on a compact, high-recall context subset.

4 DELTA: Dynamic Layer-Aware Token Attention

The empirical patterns described in Section 3 motivate a layer-aware sparse attention design that minimizes redundant computation while preserving reasoning accuracy. In early layers, attention maps are diffuse and unstable, requiring full-sequence attention to build reliable representations. In contrast, deeper layers exhibit high inter-layer correlation: once a small set of context tokens becomes salient, subsequent layers largely reuse them. Finally, as decoding proceeds, the regions of strong attention shift gradually along the sequence, a phenomenon we term *sequential drift*. Together, these observations suggest that only a few layers need to compute full attention to refresh salient tokens, while the rest can reuse them at low cost.

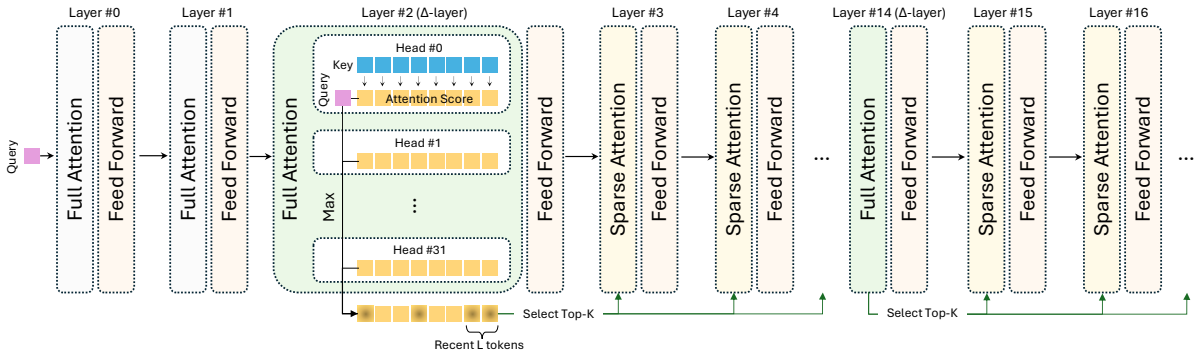

Figure 2: Overview of the DELTA decoding process. The first two layers perform full attention for initialization, Δ-layers (e.g., Layers 2 and 14) run full attention to select salient tokens, and subsequent sparse attention layers attend only to those selected tokens, as indicated by green arrows.

Core idea. DELTA operationalizes this principle through a structured three-tier layer design. The first few layers perform full attention to stabilize representations, as early layers show no consistent sparsity structure. A small number of intermediate Δ-layers act as *selection layers* that re-run full attention to identify a compact set of salient tokens carrying most of the attention mass. The remaining layers perform sparse attention restricted to those selected tokens, reusing them until the next Δ-layer updates the selection. This design removes redundant full-sequence computation across correlated layers while maintaining a high-recall context for reasoning.

Query-adaptive refresh. Because attention focus evolves with each new query embedding Q_t, the salient set must be updated at every decoding step. Skipping or caching old selections would cause stale focus and recall loss. Therefore, each Δ-layer recomputes full attention per generated token, ensuring that the reduced context remains aligned with the evolving query. However, this update occurs only at the sparse set of Δ-layers, keeping total computation cost low. Notably, DELTA never discards tokens from the KV cache: older tokens may regain relevance due to long-range reasoning dependencies. Instead, it restricts computation, not memory, by letting sparse attention layers attend only to the currently selected subset while retaining the full cache for later refreshes.

Token-level formulation. We first describe DELTA as selecting salient tokens under a fixed token budget. This formulation clarifies the selection objective. We then describe the page-based implementation used in practice for efficient KV-cache access on GPUs.

Selection mechanism. At each Δ-layer i, we form a reduced token set ρ by preserving a recency window of ℓ tokens and selecting the remaining $k-\ell$ tokens by importance. Let $\alpha_j^i = \mathrm{softmax}(A_j^i) \in \mathbb{R}^s$ denote the attention weights of head j (Eq. 4). We define the score of token t as its maximum attention weight across heads:

$$s_t = \max_{j=1,\ldots,m} \alpha_j^i(t).$$

We then select the top-$(k-\ell)$ tokens by s_t among $\{1,\ldots,s-\ell\}$ and merge them with the recency window:

$$\rho = \mathrm{Topk}\left(\{s_t : t \le s-\ell\},\, k-\ell\right) \cup \{s-\ell+1,\ldots,s\}.$$

Page-based DELTA. Token-level KV management fragments memory and hinders efficient GPU access. Following common practice, we store the KV cache in fixed-size pages of P tokens (Kwon et al., 2023; Dao, 2023). For the page-based implementation, let K denote the total page budget and L the recency-window budget in pages; the corresponding token budgets are $k = KP$ and $\ell = LP$. Let $\mathcal{P} = \{1,\ldots,\lceil s/P \rceil\}$ be the page set and let $p(t) \in \mathcal{P}$ map token t to its page. We score each page by summing the token scores it contains:

$$S_u = \sum_{t:\, p(t)=u} s_t.$$

We then keep the last L pages and select the top-$(K-L)$ remaining pages by S_u. The reduced context is the union of tokens in these pages, enabling coalesced memory access and lower gather/scatter overhead during decoding. All experiments in this work use the page-based implementation.

5 Experiments

Experimental setup. We evaluate DELTA on four distilled DeepSeek-R1 variants (Guo et al., 2025): DeepSeek-R1-Distill-Qwen-1.5B, 7B, and 14B, and DeepSeek-R1-Distill-Llama-8B, abbreviated as DS-Qwen-1.5B, DS-Qwen-7B, DS-Qwen-14B, and DS-Llama-8B, respectively. We assess reasoning performance on four open-source benchmarks: AIME-2024, AIME-2025 (AIME, 2025), GPQA-Diamond (Rein et al., 2024), and MATH500 (Hendrycks et al., 2021). The AIME datasets contain advanced high-school mathematics problems spanning algebra, geometry, number theory, and combinatorics. MATH500 is drawn from high-school competitions and covers five difficulty levels in the Art of Problem Solving framework, while GPQA-Diamond evaluates graduate-level scientific reasoning across biology, chemistry, and physics. We evaluate all 30 problems from AIME-2024 and AIME-2025, and the first 100 problems from GPQA-Diamond and MATH500.

Implementation details. We implement DELTA with the FlashInfer Just-In-Time (JIT) module (Ye et al., 2025), which extracts attention logits directly from the decoding kernel, and use the native PyTorch `topk` operator for page selection. This design keeps DELTA lightweight and easy to integrate across models. Unless otherwise specified, all experiments use the page-based implementation of DELTA with page size $P{=}16$, following common practice (Kwon et al., 2023; Dao, 2023). For readability, we sometimes report budgets in token-equivalent units; for example, a page budget of $K{=}64$ corresponds to 1k tokens. We reproduce the baselines, RaaS and Quest, in Hugging Face Transformers (Hugging Face, 2025) to ensure a consistent comparison. All experiments are conducted on a single node with eight NVIDIA A100 (SXM4, 40GB) GPUs.

Baselines. We compare DELTA against three approaches. **Full** denotes standard decoding where all layers attend to the entire KV cache. **Quest** (Tang et al., 2024) is a selection-based method that compresses each KV page into two representative vectors (element-wise min/max of keys), scores pages against the current query, and retrieves the top-k for attention; it preserves the full cache in HBM but incurs overhead from storing representatives (two key vectors per page, i.e., $1/8$ of KV memory for a page size of 16). **RaaS** (Hu et al., 2025) is an eviction-based method that removes pages with consistently low attention scores, lowering memory usage but risking the loss of tokens that may later become important.

Metrics. *Accuracy* measures whether the model's final answer is mathematically equivalent to the ground-truth answer, and is reported as the fraction of correctly solved problems in the evaluation set. *Decoding length* denotes the number of tokens generated before either the end-of-sequence token or the maximum generation limit is reached, capturing the length of the reasoning trajectory. *Throughput* is defined as the total number of generated tokens divided by the total decoding time. *Forward time* is defined as the time required for a single model forward pass.

5.1 Δ-layer configuration

We use a fixed Δ-layer configuration for each model across all experiments, unless otherwise specified. For all models, we use full attention in layers $[0, 1]$ during initialization, since the earliest layers exhibit diffuse attention and do not yet form stable sparsity patterns. Layer $[2]$ is always selected as the first Δ-layer to perform the initial salient-page selection. Two additional Δ-layers are placed later in the network to refresh the selected pages as attention evolves with depth.

Δ-layer calibration. To choose these later Δ-layers, we use a lightweight calibration procedure on a small calibration set. Specifically, we run full-attention decoding, record the attention maps of all layers, and compute the average inter-layer shift between consecutive layers using $1 - \text{cosine similarity}$, averaged over decoding steps and samples. We then select layers with large shifts while ensuring that they are well distributed across the network depth. The intuition is that large inter-layer shifts indicate transition points where the previous layer becomes less reliable for predicting salient pages for later layers, making these layers effective refresh points.

Model-specific Δ-layers. Following this procedure, DS-Qwen-1.5B uses layers $[2, 14, 23]$ out of [0-27], and DS-Qwen-7B uses layers $[2, 14, 22]$ out of [0-27]. DS-Qwen-14B uses layers $[2, 6, 42]$ out of [0-47], and DS-Llama-8B uses layers $[2, 8, 31]$ out of [0-31] as Δ-layers. We use the same calibrated layer configuration for all datasets evaluated for a given model.

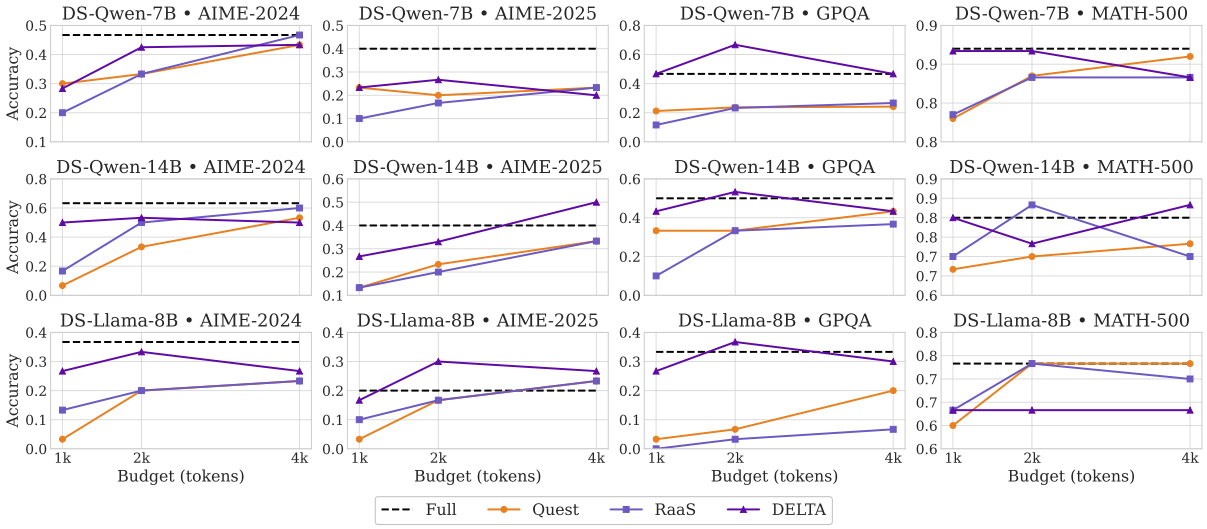

Figure 3: Accuracy of sparse attention methods on reasoning benchmarks. DELTA consistently matches or exceeds the accuracy of full attention under limited context budgets and remains robust across datasets.

5.2 Accuracy Results

For all accuracy experiments, we use a page budget of $K{=}64$ with page size $P{=}16$, corresponding to a total token budget of 1k, and reserve $L{=}8$ pages (128 tokens) for the recency window. Figure 3 compares the accuracy of Quest, RaaS, and DELTA, across four reasoning datasets and three models. Three consistent patterns emerge. First, with a 1k-token budget, DELTA consistently outperforms existing sparse methods and often matches or even surpasses the full attention baseline. For instance, on AIME-2024 with DS-Qwen-14B, Quest and RaaS achieve below 20% accuracy, whereas DELTA attains nearly 50%, approaching the 60% accuracy of full attention. Second, increasing the token budget from 1k to 2k often improves performance, reflecting the diminishing effect of sparsity-induced selection errors. In several cases, DELTA with a 2k-token budget even surpasses the full attention baseline; for example, on GPQA with DS-Qwen-7B, DELTA outperforms full attention by roughly 30%. Finally, expanding the budget further to 4k yields marginal or no improvement and occasionally a slight decline in accuracy. This plateau suggests that DELTA captures most salient context within small budgets, beyond which additional tokens primarily introduce redundancy rather than useful information.

5.3 Speedup Results

Figure 4 (left) shows the cumulative distribution function (CDF) of decoding lengths across evaluated samples, with the CDF on the x-axis and decoding length on the y-axis. Better methods achieve the same cumulative fraction at smaller decoding lengths, corresponding to curves that are shifted to the right. DELTA consistently matches or improves over full attention and outperforms other sparse-KV baselines, showing that its sparsity does not lengthen reasoning trajectories.

To measure runtime and throughput, we use synthetic decoding traces that increase the generated length from 1 token up to the target maximum length for each model. Figure 4 (right) shows the per-round decoding latency of full attention and DELTA with a 1k-token budget. Experiments are conducted on DS-Qwen-1.5B with batch size 64 and a maximum decoding length of 18k tokens. The gray vertical line marks the point where DELTA begins page selection. Beyond this point, latency grows much more slowly under DELTA: full attention increases from 7.5 ms to about 30 ms, whereas DELTA rises to only 13 ms, corresponding to about 4× smaller growth. Overall decoding time decreases from 403 to 261 seconds, while throughput increases from 2,921 to 4,517 tokens/s, a 55% improvement. These results show that DELTA improves end-to-end decoding efficiency without increasing output length.

6 In-depth Analysis

Unless otherwise specified, all experiments in this section use a mixed evaluation set of 120 samples, constructed from 30 samples from each benchmark: AIME-2024, AIME-2025, GPQA, and MATH500. We refer to this set as **Mixed120**.

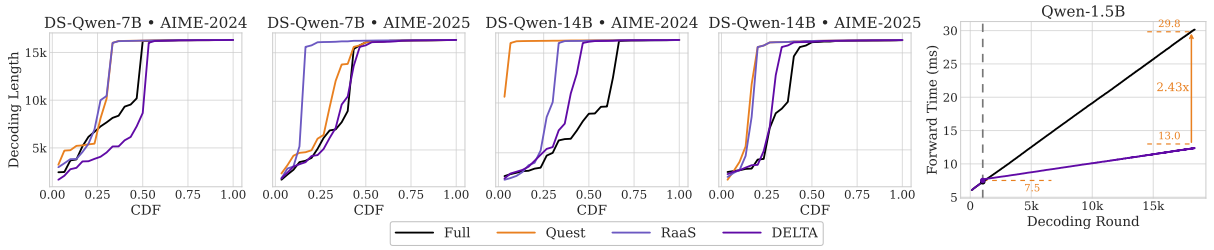

Figure 4: (Left) CDF of decoding lengths across evaluated samples, where DELTA achieves shorter or comparable decoding lengths than full attention and other sparse-KV baselines. (Right) Once token selection is activated (gray line), DELTA slows the growth of latency relative to full attention, increasing throughput.

6.1 Effect of Δ-layers Configuration

We first study how the number of Δ-layers affects runtime. We vary the number of Δ-layers from 1 up to the maximum possible value. Figure 5 shows the results for DS-Qwen-1.5B, DS-Qwen-7B, and DS-Qwen-14B. DS-Qwen-1.5B is evaluated on a single GPU with batch size 64 and generation up to 18k tokens, DS-Qwen-7B on 2 GPUs with tensor parallelism and batch size 64 up to 16k tokens, and DS-Qwen-14B on 4 GPUs with tensor parallelism and batch size 32 up to 19k tokens. We report both the forward time of the last decoding step, corresponding to the maximum context length, and the average forward time across all decoding steps. The dashed lines indicate the corresponding full attention latencies. Across all models, increasing the number of Δ-layers consistently increases runtime. This is because each additional Δ-layer performs full attention to refresh the selected pages. As the number of Δ-layers approaches the total number of layers, DELTA gradually reduces to full attention, and its runtime correspondingly approaches the full attention baseline.

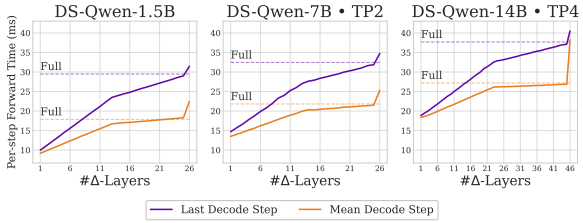

Figure 5: Adding more Δ-layers consistently increases runtime and approaches the full attention baseline.

Δ-layer configurations generalize across datasets, but affect overall accuracy. To study the sensitivity of accuracy to the choice of Δ-layers, we evaluate DS-Qwen-1.5B and DS-Qwen-7B using 10 different Δ-layer configurations on Mixed120. Figure 6 shows that overall accuracy can vary noticeably across configurations, indicating that selecting suitable Δ-layers is important for each model. This effect is more visible for the larger model, where accuracy is higher and more sensitive to the underlying configuration. At the same time, the relative trends across datasets remain largely consistent for a given model as the Δ-layer configuration changes. This suggests that, although the absolute accuracy depends on the selected Δ-layers, a good configuration generalizes across datasets and does not exhibit dataset-specific behavior.

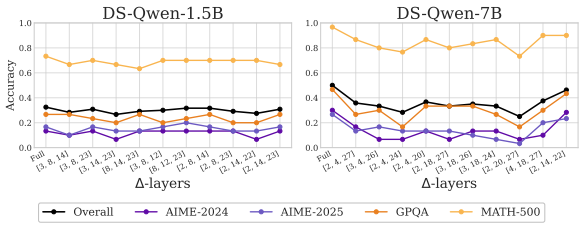

Figure 6: While overall accuracy varies across configurations, the trends across datasets remain largely consistent within each model.

6.2 Effect of Recency Window L

Figure 7 shows the effect of the recency-window size L on accuracy under different page budgets K for DS-Qwen-7B on Mixed120. We observe that accuracy is sensitive to the choice of L, with differences of up to 10% across settings. Under larger page budgets, such as $K{=}256$ and $K{=}512$ pages (4k and 8k tokens), smaller recency windows tend to perform best, with $L{=}8$ giving the highest accuracy. In contrast, under the smallest budget, $K{=}64$ pages (1k tokens), a larger recency window performs better. This trend reflects a trade-off between preserving very recent context and allocating budget to a broader set of salient tokens: when the budget is large, the model benefits more from broader contextual coverage, whereas under tighter

budgets, retaining recent context becomes more important.

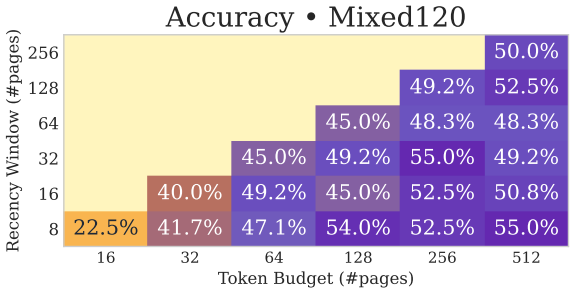

Figure 7: Accuracy under different page budgets K and recency-window sizes L for DS-Qwen-7B

7 Related Work

Efficient long-context inference. Long-context LLMs face quadratic compute and memory overhead from full self-attention, making inference increasingly dominated by KV-cache bandwidth. Even with optimized kernels such as FlashAttention (Dao, 2023) and paged caching (Kwon et al., 2023), decoding throughput scales poorly with sequence length. Modern architectures (e.g., Llama-3.1, GPT-4o, Claude 3.5 Sonnet) extend context to 128k–200k tokens through rotary positional encoding (Su et al., 2024), yet runtime remains bottlenecked by repeated KV reads rather than arithmetic compute, highlighting the need for structural sparsity that reduces redundant memory access.

Architectural and KV-compression methods. Architectural approaches such as Multi-Query and Grouped-Query Attention (Shazeer, 2019; Ainslie et al., 2023) reduce redundant KV heads, while recurrent alternatives like RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu and Dao, 2023) replace self-attention with stateful recurrence. These designs improve efficiency but require model retraining and often underperform Transformers on complex reasoning tasks. In contrast, KV-compression strategies optimize inference at runtime. Quantization (Xiao et al., 2023a; Yao et al., 2022; Dettmers et al., 2022; Liu et al., 2024b) lowers precision to save bandwidth, whereas pruning methods exploit sparsity to drop less important tokens. Eviction-based schemes such as H2O (Zhang et al., 2023), SnapKV (Li et al., 2024), TOVA (Oren et al., 2024), ScissorHands (Liu et al., 2023), R-KV (Cai et al., 2025), and RaaS (Hu et al., 2025) bound memory by discarding low-score pages, but may lose tokens that later become critical for reasoning continuity.

Sparse and selection-based attention. Selection-based methods preserve the full KV cache but compute attention only over a subset of salient tokens. Early static patterns in Sparse Transformers, Longformer, and BigBird (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020) established fixed sparsity layouts, later refined into adaptive mechanisms guided by query-dependent importance. Quest (Tang et al., 2024) scores KV pages against the current query to retrieve the most relevant subset, while TidalDecode (Yang et al., 2024) exploits the strong spatial coherence of attention across layers by performing full attention only in a few token-selection layers and reusing the selected tokens in intermediate sparse layers. SeerAttention-R (Gao et al., 2025) employs a self-distilled gating module to learn block-sparse attention, achieving near-lossless decoding but requiring additional training. However, existing sparse attention methods either incur notable accuracy degradation at low retention ratios or depend on costly post-training procedures to recover performance, both of which substantially increase decoding length and computational overhead for reasoning tasks. In contrast, **DELTA** is proposed as a selection-based, training-free approach that leverages inter-layer attention correlation during reasoning to maintain high accuracy under reduced token budgets, without extending the overall generation length.

8 Conclusion

We introduced DELTA, a training-free, layer-aware sparse attention mechanism that improves the efficiency of long-context reasoning in large language models. By leveraging cross-layer correlation and gradual evolution of token importance, DELTA computes full attention only in a few key Δ-layers and reuses their selected high-recall subsets across subsequent sparse attention layers. This design substantially reduces decoding-time bandwidth and latency while maintaining accuracy comparable to full attention. Experiments on multiple reasoning benchmarks confirm that DELTA achieves consistent speedups over state-of-the-art sparse and eviction-based methods without retraining, highlighting layer-aware reuse as a promising direction for efficient reasoning-time inference.

Limitations

While DELTA enables efficient long-context reasoning, it has the following limitations.

KV-memory footprint. DELTA preserves the full KV cache in HBM and reduces compute rather than peak memory. As a result, it does not directly address out-of-memory failures at extreme context lengths or on smaller GPUs. Future work includes integrating DELTA with complementary memory-saving techniques (e.g., quantization, eviction under guarantees, or offloading) while maintaining high selection recall.

Generality. Our results are limited to distilled DeepSeek-R1 models evaluated on reasoning benchmarks, mainly math and science QA. Generalization to other architectures, modalities, or workloads such as open-ended conversation and code generation remains unverified and may require re-tuning of the Δ-layer schedule and context budgets.

Sensitivity and overhead. Performance depends on Δ-layer placement and (K, L); the max-attention scoring adds small overhead and can lag under fast attention drift. Adaptive per-sample scheduling or lightweight learned selectors are promising fixes.

Acknowledgments

We sincerely thank all the reviewers for their time and constructive comments. This material is based upon work supported by NSF award number 2224319, REAL@USC-Meta center, and VMware gift.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

MAA AIME. 2025. American invitational mathematics examination-aime 2024, 2024.

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

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

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, and 1 others. 2025. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*.

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

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, and 1 others. 2025. Seerattention-r: Sparse attention adaptation for long reasoning. *arXiv preprint arXiv:2506.08889*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Google DeepMind. 2025. Gemini 2.5: Our most intelligent ai model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. Accessed: 2025-09-27.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. 2025. Raas: Reasoning-aware attention sparsity for efficient llm reasoning. *arXiv preprint arXiv:2502.11147*.

Hugging Face. 2025. Hugging face. https://huggingface.co/. Accessed: 2025-09-27.

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

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, and 1 others. 2024a. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

OpenAI. 2025a. Introducing GPT-OSS. https://openai.com/index/introducing-gpt-oss/. Accessed: 2025-09-27.

OpenAI. 2025b. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/. Accessed: 2025-09-27.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, and 1 others. 2023. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

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

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*.

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

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023b. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. 2024. Tidaldecode: Fast and accurate llm decoding with position persistent sparse attention. *arXiv preprint arXiv:2410.05076*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35:27168–27183.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and 1 others. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, and 1 others. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*.

Linan Yue, Yichao Du, Yizhi Wang, Weibo Gao, Fangzhou Yao, Li Wang, Ye Liu, Ziyu Xu, Qi Liu, Shimin Di, and 1 others. 2025. Don't overthink it: A survey of efficient r1-style large reasoning models. *arXiv preprint arXiv:2508.02120*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and 1 others. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Algorithm

At each refresh layer, DELTA converts attention weights into token importance scores by taking the maximum attention over heads for each token. It then aggregates token scores within each page, always preserves the most recent $\ell = \lceil L/P \rceil$ pages, and fills the remaining budget with the highest-scoring older pages. The resulting page set is reused by subsequent sparse layers until the next refresh layer.

Algorithm 1 DELTA decoding for one generation step

Require: current hidden state h_t^0, full KV cache $\mathcal{C}_t$, warm-up depth r, refresh-layer set $\mathcal{D}$, page budget k, recency window L, page size P

$\rho \leftarrow$ all pages in $\mathcal{C}_t$
$\ell \leftarrow \lceil L/P \rceil$ ▷ number of recency pages
for $i = 1$ to N **do**
  if $i \leq r$ **then**
    Run layer i with full attention over $\mathcal{C}_t$
    Obtain updated hidden state h_t^i
  else if $i \in \mathcal{D}$ **then**
    Run layer i with full attention over $\mathcal{C}_t$
    Obtain updated hidden state h_t^i and attention weights A^i
    $\rho \leftarrow$ RefreshPages(A^i, k, ℓ, P)
  else
    Run layer i with sparse attention over the selected pages ρ
    Obtain updated hidden state h_t^i
  end if
end for
return next-token prediction from h_t^N

Algorithm 2 Refreshing the selected pages in DELTA

procedure RefreshPages(A^i, k, ℓ, P)
  Convert the attention weights at layer i into a token importance score
    by taking, for each token, its largest attention weight across heads
  Group tokens into pages of size P
  Compute a page score by summing the importance scores of tokens in the same page
  Keep the last ℓ pages to preserve recency
  From the remaining older pages, select the top-$(k-\ell)$ pages by page score
  Return the union of the recent pages and the selected high-score pages
end procedure

B Extended Experimental Results

B.1 Speedup Results

Experimental Setup. Figures 8–10 present the per-decoding-round runtime breakdown for the speedup experiments in Figure 4 (right) for three model settings: DS-Qwen-1.5B on a single GPU, DS-Qwen-7B with tensor parallelism degree 2 (TP2, using two GPUs), and DS-Qwen-14B with tensor parallelism degree 4 (TP4, using four GPUs). In each case, the figure compares Full and DELTA with page budget K=64 side by side across decoding rounds, so the plots show not only the speed difference but also how the runtime composition evolves as generation proceeds and the effective context grows.

Breakdown Components. Each decoding step is decomposed into four measured components: `collect-pages`, `planning`, `model-forward`, and `post-processing`. The `collect-pages` term covers gathering the KV-page metadata and page indices needed for the current step. The `planning` term measures the preparation of the decode wrappers and related execution metadata. The `model-forward` term measures the actual decode computation, and for DELTA this also includes the planner-side attention dump, page-score computation, and top-k page selection logic that are executed as part of the forward path. Finally, `post-processing` covers the remaining bookkeeping after the forward pass, such as extracting the next token and updating request state. The black `step total` curve is the sum of these components, while the stacked plots visualize how each component contributes to the total latency at each decoding round.

Interpretation. Compared with Full, DELTA introduces higher overhead in `collect-pages` and `planning` because it must support two attention paths rather than one: a full-context planner/dump attention path used to obtain attention statistics for page selection, and a subset-attention path used after the selected pages have been determined. Consequently, DELTA must gather page information and prepare execution state for both the full and subset paths, which increases these overhead terms. Nevertheless, the dominant effect in all three settings is the reduction in `model-forward` time. Although the DELTA `model-forward` component still includes the page-scoring and top-k selection logic, it substantially reduces the later-layer attention workload by running those layers on a selected subset of KV pages instead of the full context. This difference becomes increasingly visible at later decoding rounds, where the cost of full-context attention continues to rise with sequence length, while DELTA keeps the later-layer attention cost much smaller,

yielding consistently lower total decoding latency for DS-Qwen-1.5B, DS-Qwen-7B TP2, and DS-Qwen-14B TP4.

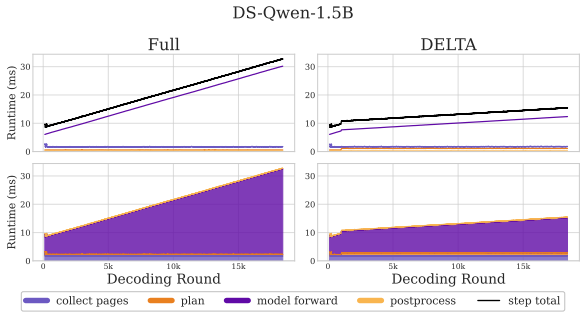

Figure 8: Runtime breakdown of DELTA and full attention on DS-Qwen-1.5B, corresponding to the speedup setting in Figure 4 (right). We report the latency of collect-pages, planning, model forward, postprocessing, and total step time across decoding steps. DELTA introduces a small overhead for page collection and planning, but substantially reduces model-forward time, which dominates the overall runtime.

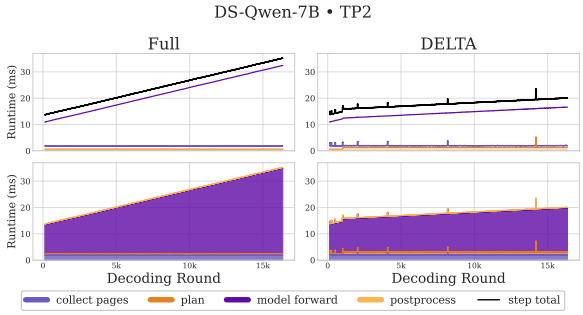

Figure 9: Runtime breakdown of DELTA and full attention on DS-Qwen-7B. As in DS-Qwen-1.5B, the additional overhead from collect-pages and planning remains small compared to the savings in model-forward time, leading to a clear reduction in total step latency.

C Effect of Recency Window L, Per-dataset Results

Figures 11–14 show the per-dataset version of Figure 7, breaking down the effect of recency-window size L across AIME-2024, AIME-2025, GPQA, and MATH-500. Only valid configurations with $L < K$ are shown. The dependence on L is visible in all four datasets, but the pattern is not uniform. On AIME-2024, small or moderate recency windows are usually best at intermediate budgets, while the best setting shifts to a larger L at the largest budget. On AIME-2025, smaller L values perform best at budgets $K = 32$–128, while a moderate recency window performs best at $K = 256$.

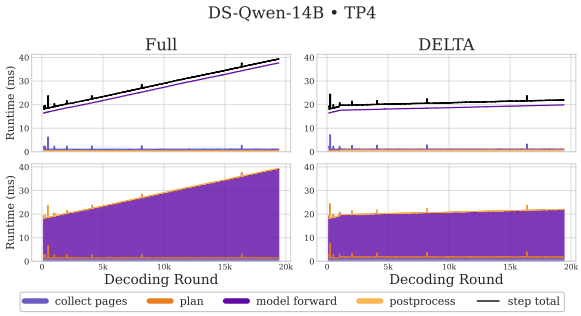

Figure 10: Runtime breakdown of DELTA and full attention on DS-Qwen-14B. The same trend holds at larger model scale: DELTA incurs modest non-forward overhead while consistently lowering forward latency, resulting in lower end-to-end decoding time per step.

On GPQA, the preferred L changes with budget, indicating a clearer trade-off between preserving recent context and retaining broader coverage. In contrast, MATH-500 is comparatively insensitive to L across most budgets, with only localized gains from particular settings. Overall, these per-dataset results show that the effect of L is dataset- and budget-dependent rather than following a single uniform trend.

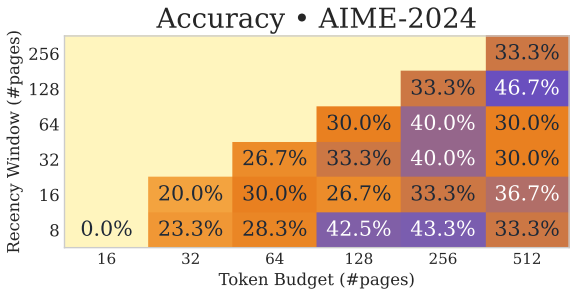

Figure 11: Accuracy on AIME-2024 under different page budgets K and recency-window sizes L for DS-Qwen-7B. The best setting varies with budget: L=8 is best at K=32, 128, 256, L=16 is best at K=64, and the largest budget $K = 512$ peaks at $L = 128$, indicating a non-monotonic dependence on L.

D Consecutive-Layer Attention Shift

Setup. Figure 15 plots the attention shift between consecutive layers as a function of layer index for three models: DS-Qwen-1.5B, DS-Qwen-7B, and DS-Qwen-14B, without tensor parallelism. The probe dataset is a fixed `Mixed40` artifact constructed from the `Mixed120` dataset by selecting exactly 10 prompts from each of AIME-2024, AIME-2025, GPQA, and MATH-500, for a total of 40 prompts. Each prompt is decoded for up to

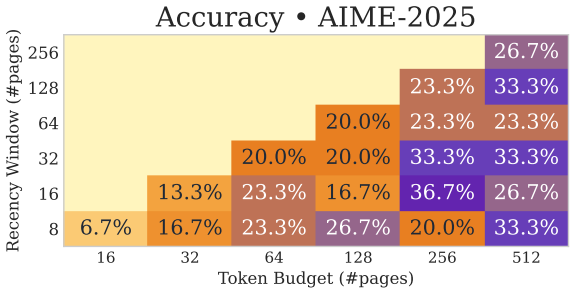

Figure 12: Accuracy on AIME-2025 under different page budgets K and recency-window sizes L for DS-Qwen-7B. Smaller recency windows are strongest at budgets K=32–128, L=16 gives the best result at K=256, and several L values tie at K=512, showing a mixed dependence on the recency window.

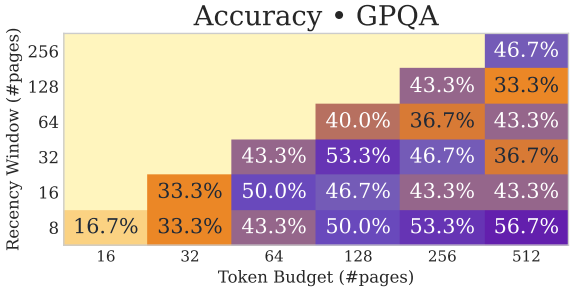

Figure 13: Accuracy on GPQA under different page budgets K and recency-window sizes L for DS-Qwen-7B. The preferred recency window shifts with budget: the best accuracy occurs at L=16 for K=64, at L=32 for K=128, and at L=8 for K=256 and K=512, indicating that the balance between recent-context preservation and broader page coverage matters for scientific reasoning.

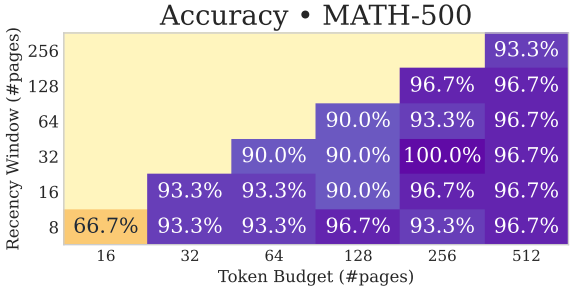

Figure 14: Accuracy on MATH-500 under different page budgets K and recency-window sizes L for DS-Qwen-7B. Compared with the other datasets, performance is relatively insensitive to L across most budgets, although K=256 reaches its peak at L=32.

1000 generated tokens.

Measurement Procedure. At each one-token autoregressive decode step, and for each transformer layer ℓ, we capture the attention distribution of the last query token over the valid context positions. Concretely, we collect the attention weights for all heads, apply a softmax over the context dimension, and flatten the resulting per-head maps into a single vector $a_\ell^{(t)}$ for decode step t. We first compute cosine similarity between consecutive layers,

$$s_{\ell-1,\ell}^{(t)} = \frac{a_{\ell-1}^{(t)} \cdot a_\ell^{(t)}}{\|a_{\ell-1}^{(t)}\| \, \|a_\ell^{(t)}\|},$$

and then define the plotted quantity as the corresponding attention shift,

$$d_{\ell-1,\ell}^{(t)} = 1 - s_{\ell-1,\ell}^{(t)}.$$

For each layer pair, we report the mean shift over all valid one-token decode steps and prompts. Thus, larger values correspond to larger changes in attention behavior from the previous layer.

Figure Structure. In each panel, the black curve shows the overall mean attention shift aggregated over all 40 prompts in `Mixed40`, while the colored curves show the per-dataset means for AIME-2024, AIME-2025, GPQA, and MATH-500. The horizontal axis is the layer index ℓ, corresponding to the right member of the consecutive pair $(\ell - 1, \ell)$, and the vertical axis is the mean value of $1 -$ cosine similarity between the attention distributions of those two neighboring layers. High values therefore mark layers whose attention pattern changes sharply relative to the previous layer, whereas low values indicate smoother transitions and more similar consecutive-layer behavior.

Overall Trends Across Models. The three models exhibit substantially different shift scales. Averaged over all consecutive layer pairs, the overall shift is approximately 0.733 for DS-Qwen-1.5B, 0.631 for DS-Qwen-7B, and 0.323 for DS-Qwen-14B. Thus, the larger model shows much smaller consecutive-layer shifts on average, indicating more aligned attention behavior across neighboring layers. At the same time, the shift is not monotonic with depth in any model. Instead, all three curves contain localized peaks that mark abrupt transitions in attention behavior. DS-Qwen-7B shows a particularly strong peak between layers $(3, 4)$ with shift ≈ 0.907, followed immediately by

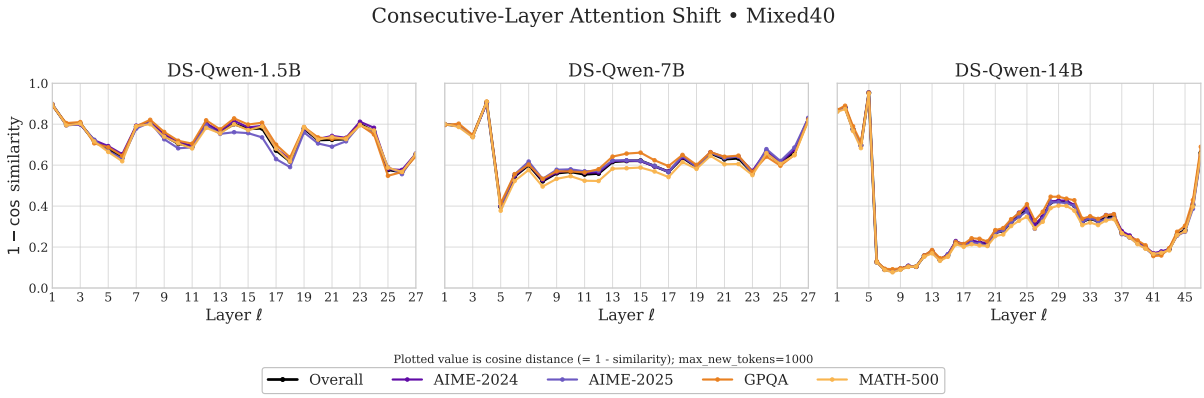

Figure 15: Attention shift between consecutive layers, measured as $1 -$ cosine similarity between their attention distributions during autoregressive decoding. Each panel shows a different model, with the black curve denoting the overall mean on `Mixed40` and the colored curves showing per-dataset means. Peaks indicate layers where attention behavior changes sharply relative to the previous layer, while low-shift regions indicate smoother transitions. The major peaks are largely consistent across datasets within each model, suggesting that consecutive-layer attention shift is primarily driven by model architecture and provides a useful signal for selecting Δ-layers.

a much smaller shift between $(4, 5)$ of about 0.397. DS-Qwen-14B shows its strongest peak between $(4, 5)$ with shift ≈ 0.953, after which a long middle block has very small shifts; for example, pairs such as $(6, 7)$, $(7, 8)$, and $(8, 9)$ are all near 0.10 or below. DS-Qwen-1.5B maintains comparatively high shift throughout much of the stack, with its largest shift appearing at the earliest pair $(0, 1)$ at about 0.896.

Dataset-wise Behavior. The per-dataset curves generally track the black overall curve closely within each model. This indicates that the dominant layerwise shift structure is driven more by model architecture than by the particular reasoning dataset. Quantitatively, the deviation from the overall curve is modest: the mean absolute difference between a dataset-specific curve and the overall curve is typically on the order of 0.004–0.019, depending on the model and dataset. GPQA and MATH-500 show somewhat larger deviations than the AIME subsets in some settings, but the major transition layers remain in essentially the same locations. In other words, the datasets affect the magnitude of the shift curve more than its global structure.

Interpretation. These results suggest that consecutive-layer attention behavior is organized into stages rather than changing smoothly and uniformly across depth. Peaks in the shift curve mark layers where the model substantially changes how it distributes attention over the context, while low-shift regions indicate stretches of neighboring layers with more similar attention behavior. This interpretation is especially relevant for DELTA: layers with large shift are natural candidates for Δ-layers, since they mark points where the model's attention behavior changes most sharply relative to the previous layer. Conversely, regions with consistently low shift suggest groups of layers with more redundant attention behavior, which are more natural candidates for stronger compression or subset-based treatment. Under this view, the probe provides an architectural signal for identifying promising DELTA layer placements by highlighting where the model undergoes its strongest layer-to-layer attention shift.

D.1 Per-model Accuracy Under Different Δ-layer Configurations

Extended results across three model sizes. Figure 16 extends Figure 6 by including DS-Qwen-14B. To construct the Δ-layer configurations evaluated here, we use the prominent shift points identified in the consecutive-layer attention-shift plots from Section D, selecting candidate layers around these peaks while ensuring coverage across network depth. Across all models, overall accuracy varies across Δ-layer configurations, showing that Δ-layer placement is important. This effect is stronger for larger models, where accuracy is more sensitive to the chosen configuration. However, within each model, the relative trends across datasets remain broadly consistent. This suggests that although the absolute accuracy depends on the selected Δ-layers, good configurations generalize across datasets rather than being dataset-specific.

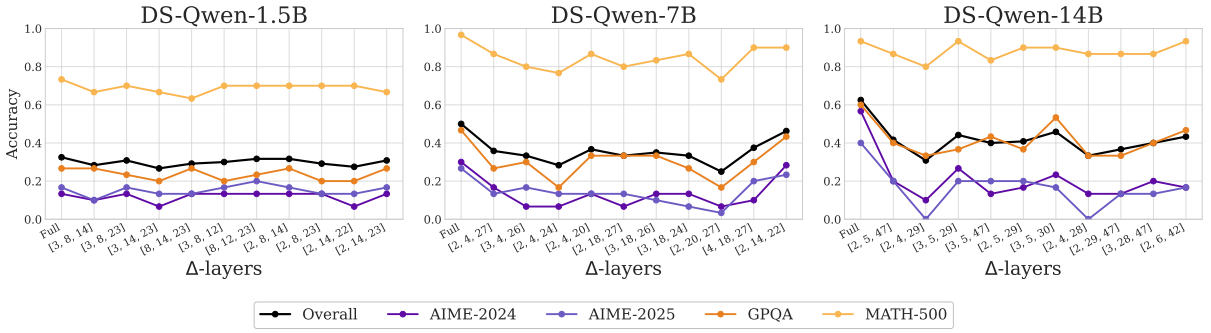

Figure 16: Accuracy across different Δ-layer configurations on Mixed120 for DS-Qwen-1.5B, DS-Qwen-7B, and DS-Qwen-14B. Each panel reports overall accuracy and the per-dataset breakdown on AIME-2024, AIME-2025, GPQA, and MATH-500.

E Breakdown of DELTA Runtime Overhead

To better understand the runtime overhead introduced by DELTA, we conduct a component-wise microbenchmark that isolates the decode-time attention kernel and the two downstream primitives used for page-aware selection. Specifically, on a single GPU without tensor parallelism, we measure four components: (1) the baseline FlashInfer decode attention kernel, (2) a JIT-instrumented FlashInfer variant that additionally dumps attention logits, (3) DELTA's fused page-scoring kernel, which aggregates token-level attention into page-level importance scores, and (4) DELTA's page-selection step, which selects the pages retained for sparse attention.

Figure 17 shows the resulting runtime breakdown. The bottom area corresponds to the baseline FlashInfer kernel, the second area to the incremental overhead of JIT logit dumping, and the top two areas to DELTA's page scoring and page selection. The results show that DELTA's relative overhead is highest at short contexts, where page selection contributes an almost fixed per-step cost while the baseline attention kernel is still small. At 1k context, the total overhead is about 154%, 81%, and 42% of the baseline FlashInfer runtime for the 1.5B, 7B, and 14B models, respectively. As context grows, the baseline attention cost increases much faster than the fixed portion of DELTA's control path, so the relative overhead decreases. By 32k context, the total overhead drops to about 25%, 29%, and 21% of the baseline runtime for the 1.5B, 7B, and 14B models, respectively.

The breakdown also reveals a shift in which component dominates. At 1k context, page selection is the largest source of overhead, accounting for

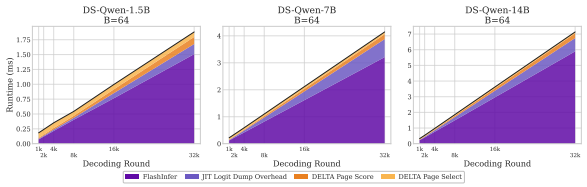

Figure 17: Stacked runtime breakdown of DELTA's auxiliary overhead across decoding rounds on a single GPU ($B = 64$). The areas show baseline FlashInfer attention, incremental JIT logit-dump overhead, DELTA page scoring, and DELTA page selection. The relative overhead is highest at short contexts due to the nearly fixed cost of page selection, but decreases as context length grows and the baseline attention kernel becomes dominant.

roughly 68–76% of the total extra cost across models. At longer contexts, the JIT logit-dump path becomes dominant, contributing roughly 46–72% of the overhead at 32k, while fused page scoring remains smaller and page selection stays nearly flat at around 0.08–0.09 ms. Overall, these results show that DELTA's overhead is most pronounced at short decoding lengths, where fixed control costs are less amortized, and becomes progressively less significant as context grows. This trend is favorable for DELTA's target setting, since long-context decoding is precisely where reducing attention cost matters most.