CACHE WHAT LASTS: TOKEN RETENTION FOR MEMORY-BOUNDED KV CACHE IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory and computation remain core bottlenecks in long-horizon LLM inference due to the quadratic cost of self-attention and the ever-growing key-value (KV) cache. Existing strategies for memory-bounded inference, such as quantization, offloading, or heuristic KV eviction, either incur high orchestration costs or rely on unreliable attention-based proxies of importance. We propose TRIM-KV, a novel approach that learns each token's intrinsic importance at creation time via a lightweight retention gate. Each gate predicts a scalar retention score that decays over time, reflecting the long-term utility of the token for a specific layer and head. Tokens with low scores are evicted when the memory budget is exceeded, ensuring that the cache always contains the most critical tokens. TRIM-KV is trained efficiently through distillation from a frozen LLM combined with a capacity loss, requiring only gate fine-tuning and adding negligible inference overhead. Across mathematical reasoning (GSM8K, MATH-500, AIME24), procedural generation (LongProc), and conversational long-memory benchmarks (LongMemEval), TRIM-KV consistently outperforms strong eviction and learnable retrieval baselines, especially in low-memory regimes. Remarkably, it even surpasses full-cache models in some settings, showing that selective retention can serve as a form of regularization, suppressing noise from uninformative tokens. Qualitative analyses further reveal that learned retention scores align with human intuition, naturally recovering heuristics such as sink tokens, sliding windows, and gist compression without explicit design. Beyond efficiency, retention scores provide insights into layer- and head-specific roles, suggesting a new path toward LLM interpretability.

1 INTRODUCTION

Modern large language models (LLMs) can, in principle, handle extremely long input contexts – some recent models support context windows of 128k tokens or more (Yang et al., 2025; Gao et al., 2024). Yet, extending context length comes with steep computational costs. The self-attention mechanism has quadratic time complexity in sequence length, and storing the key-value (KV) cache for thousands of tokens can quickly exhaust GPU memory (Wang et al., 2025; Li et al., 2024a). In practical deployments, the KV cache, which saves past key and value vectors to avoid re-computation, becomes a major memory and latency bottleneck for long-context inference. Decoupling resource usage from context length is therefore critical for enabling efficient and scalable applications such as long-horizon reasoning (Chen et al., 2025) and lifelong agents (Zheng et al., 2025; Li et al., 2024d).

To address this challenge, recent work has explored memory-bounded LLMs that can operate effectively under constrained KV budgets (Li et al., 2024a). One line of research focuses on compression and quantization, aiming to reduce memory footprint by learning compact representations of past tokens rather than storing all keys and values explicitly (Hooper et al., 2024; Saxena et al., 2024). These techniques are mostly effective during the prefill phase but scale poorly with generation length. Another line leverages attention sparsity to offload most of the cache to CPU or secondary storage, and retrieve only relevant segments on demand via similarity search (Tang et al., 2024) or learned indices (Gao et al., 2025). While offloading lowers the on-GPU footprint, it incurs nontrivial orchestration overhead that accumulates over long generations, undermining end-to-end throughput.

A more common and direct approach to enforce a fixed memory budget is KV cache eviction, which directly drops certain tokens from the KV cache (Xiao et al., 2023). Many KV eviction strategies

have been proposed to decide which tokens to remove. However, most of them are attention-guided heuristics: they track attention from new queries to cached tokens and retain those that are recently or frequently attended, adapting the cache to the current focus (Zhang et al., 2023; Li et al., 2024c; Wang et al., 2025; Liu et al., 2025; Ghadia et al., 2025; Cai et al., 2025). While being efficient, these methods assume that recent attention is a reliable proxy for future importance. This assumption often breaks for long-horizon generation and reasoning tasks: a token might be crucial much later, even if it has not been attended to in the recent past (Jiang et al., 2024). Moreover, attention-based eviction can suffer from attention bias, *e.g.,* the model might temporarily overlook a needed token due to a distracting context (Shi et al., 2023), causing it to be evicted prematurely. While some recent studies have attempted to learn better eviction decisions (Chen et al., 2024; Zeng et al., 2024), these methods typically scale poorly with sequence length and are therefore limited to the prefilling stage.

In this work, we take a new perspective on the KV eviction problem. Rather than relying on the attention-guided importance, we propose to learn each token's intrinsic importance at the time of its creation and use that as the basis for eviction. Intuitively, not all tokens are created equal: some carry significant semantic or task-related weight (*e.g.* a critical fact, a question being answered, or the first few "sink" tokens that often encode the topic or instructions), while others are relatively inconsequential (*e.g.* filler words, stopwords, or trivial arithmetic steps in a chain-of-thought). Moreover, the importance of tokens is not uniform across the network, but it varies systematically by layers and heads, reflecting their functional specializations (Voita et al., 2019; Wu et al., 2024b).

We posit that the contextual embedding of a token already encodes much of its long-term utility. We therefore introduce a retention gate that maps the token's embedding and produces a scalar retention score $\beta \in [0, 1]$ reflecting the token's inherent importance for a specific layer and head. Especially, we design this retention score to decay exponentially as the context grows, mimicking the gradual forgetting of old information in human brains (Ebbinghaus, 2013). Thus, a highly important token will have $\beta \approx 1$ and retain a high score for a long time, whereas a token deemed unimportant will have β closer to 0 and its influence will vanish quickly. We leverage this score to drive a simple eviction policy: whenever the number of cached tokens exceeds the budget M, we evict the token with the smallest current retention score. This approach, which we call **T**oken **R**etent**I**on for **M**emory-bounded **KV** Cache (TRIM-KV), ensures that at all times, the cache is filled with the M tokens judged most intrinsically important, with a preference toward more recently generated tokens.

Implementing retention-based caching in an existing LLM only requires adding a few lightweight components. We integrate the retention gates into each self-attention layer of a pretrained model to modulate attention weights by token importance during training. We then train only the gates with a two-part loss: a distillation loss that compels the modified model to mimic the original model's outputs, thus preserving quality, and a capacity loss that penalizes exceeding the target memory budget, thus encouraging sparseness in attention via eviction. Importantly, by training the gates across all layers jointly, the model can learn a coordinated, globally optimal caching policy rather than greedy layer-wise decisions. At inference time, the learned retention gates produce per-token scores on the fly, and eviction is implemented with a simple score comparison, adding minimal overhead.

Results and Contributions. Through extensive experiments on long-context and long-generation benchmarks, we demonstrate that our learnable token retention approach substantially improves the performance of memory-bounded LLMs. On challenging mathematical reasoning datasets, GSM8K, MATH, AIME, a long procedural generation benchmark, LongProc, and a long-memory chat assistant benchmark, LongMemEval, our method consistently outperforms eviction baselines, even when those baselines use $4\times$ more KV budget, and deliver 58.4% pass@1 gain compared to the SOTA learnable KV retrieval baseline (Gao et al., 2025), especially in low-memory regimes. Remarkably, in several settings, TRIM-KV even surpasses a full-cache model, suggesting that selective retention can function as an effective regularizer by suppressing noise from uninformative tokens.

We also present qualitative evidence that learned retention scores align with human intuition: the model tends to assign high scores to initial tokens and problem descriptions, and low scores to less meaningful punctuation. Notably, many behaviors reminiscent of common heuristics, such as keeping sink tokens, sliding windows, and gist tokens Mu et al. (2023), emerge naturally and adaptively from our learned policy, without being hard-coded. Finally, we show that these learned retention scores can also act as a diagnostic tool for probing layer- and head-specific dynamics, providing a lightweight means to analyze and ultimately improve the interpretability of attention patterns.

2 Related Work

KV Cache Compression. As model sizes and context windows grow, optimizing KV-cache memory is increasingly critical. Prior work largely falls into three directions: (i) token eviction/merging (Xiao et al., 2023; Li et al., 2024c; Zhang et al., 2023; Nawrot et al., 2024; Zhang et al., 2024; Qin et al., 2025; Wang et al., 2025; Liu et al., 2025; Park et al., 2025; Cai et al., 2025), (ii) vector compression/quantization (Hooper et al., 2024; Liu et al., 2024b; Yue et al., 2024; Sun et al., 2024a), and (iii) token retrieval (Tang et al., 2024; Liu et al., 2024a; Gao et al., 2025). While effective in many settings, vector compression and retrieval either discard fine-grained information or introduce nontrivial systems overhead (e.g., coordination and data movement) (Li et al., 2024a). Moreover, their memory and computation still scale with sequence length, making them inefficient for long-horizon generation applications. Token eviction offers a simple, memory-bounded alternative; however, most existing policies are heuristic and can significantly degrade performance, especially on long reasoning trajectories. Recent work has introduced learnable eviction policies (Chen et al., 2024; Zeng et al., 2024), but these are primarily designed for the pre-filling stage and thus are not well suited to sustained long-horizon generation. We bridge this gap by introducing a learnable and efficient eviction policy designed for long-horizon LLM inference under fixed memory budgets.

Forgetting in Language Models. A key limitation of vanilla self-attention is the lack of an explicit forgetting mechanism, forcing the model to carry potentially irrelevant information and making long-context processing inefficient. Early work tackled this by replacing quadratic attention with linearized and recurrent variants (Katharopoulos et al., 2020; Wang et al., 2020; Sun et al., 2023; Yang et al., 2023; 2024) that summarize the past into a fixed-size state, often a single vector. While computationally attractive, such heavy compression can degrade performance on tasks requiring long-range memory. More recent approaches (Behrouz et al., 2024; Sun et al., 2024b; Karami et al.; Karami and Mirrokni, 2025) replace the hidden vector with a neural network to increase memory capacity, but at the cost of training complexity and uncertain scalability to contemporary LLM sizes. In contrast, we introduce a plug-in forgetting mechanism for pretrained LLMs that converts them into memory-bounded models, delivering efficiency for long generation without retraining from scratch.

3 Preliminaries

3.1 Transformers with Self-Attention

Given a sequence of d-dimensional input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_T$, a (causal) self-attention layer attends only to past positions. For each $t = 1, \ldots, T$, the attention output $\mathbf{o}_t$ is computed as

$$\mathbf{q}_t = \mathbf{W}_Q \mathbf{x}_t, \mathbf{k}_t = \mathbf{W}_K \mathbf{x}_t, \mathbf{v}_t = \mathbf{W}_V \mathbf{x}_t, \quad \mathbf{o}_t = \sum_{i=1}^{t} \frac{\exp\left(\mathbf{q}_t^\top \mathbf{k}_i\right)}{\sum_{j=1}^{t} \exp\left(\mathbf{q}_t^\top \mathbf{k}_j\right)} \mathbf{v}_i,$$

where $\mathbf{q}, \mathbf{k}, \mathbf{v}$ are query, key, and value states, respectively, and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$ are linear transformation weights. Here, we assume a single-head attention layer and omit the scaling factor $1/\sqrt{d}$ for simplicity. The sequence of key-value pairs $\{(\mathbf{k}_i, \mathbf{v}_i)\}_i$ is the in-context memory of the LLM. During the autoregressive decoding, we typically generate one token at a time and cache the running key-value pair $(\mathbf{k}_t, \mathbf{v}_t)$ to our in-context memory to avoid recomputation. However, this vanilla caching approach leads to a linear increase in memory footprint with the sequence length, while computation grows quadratically (Keles et al., 2023). This reduces efficiency when handling long-context inputs and extended generation tasks.

3.2 Revisiting KV Cache Eviction

A common method to address the linear growth in the memory is to prune or compress the running key-value pairs into fixed-size (slot) memory. As new tokens arrive, we evict *un-(or less-)important* tokens from our memory and append the new ones. To understand this procedure, we revisit and rewrite the attention computation with eviction at inference step t as follows:

$$\mathbf{o}'_t = \sum_{i=1}^{t} \frac{\exp\left(\alpha_{ti} \mathbf{q}_t^\top \mathbf{k}_i\right)}{\sum_{j=1}^{t} \exp\left(\alpha_{tj} \mathbf{q}_t^\top \mathbf{k}_j\right)} \mathbf{v}_i \quad \text{where} \quad \alpha_{ti} \in \{0, 1\} \text{ and } \alpha_{ti} \leq \alpha_{t+1,i}, \ \ \forall i, t. \tag{1}$$

In Equation (1), we introduce a binary variable $\alpha_{ti} \in 0, 1$ indicating whether key–value pair i has been evicted at time t and the monotonicity constraint $\alpha_{ti} \leq \alpha_{t+1,i}$ ensures that we cannot retrieve a token once it is evicted (Figure 1). The goal is to choose a decision variable α so that the attention output deviates as little as possible from the full KV cache (all $\alpha_{ti} = 1,\ \forall i, t$).

$$\min_{\alpha} \ \mathcal{L}_{\text{base}}(\mathbf{o}'_t; \mathbf{o}_t) \quad \text{s. t.} \sum_{i=1}^{t} \alpha_{ti} \leq M. \tag{2}$$

Here, $\mathcal{L}$ penalizes differences between attention with and without eviction, and the constraint enforces keeping at most M tokens at any inference step t.

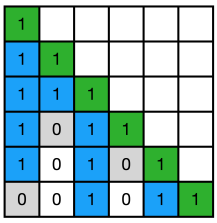

Figure 1: Attention with eviction.

Solving the above constrained optimization at every time step t is impractical due to its combinatorial nature and efficiency requirements of LLM inference in real-world applications. Most existing approaches (Xiao et al., 2023; Han et al., 2023; Zhang et al., 2023; Li et al., 2024c; Cai et al., 2025; Ghadia et al., 2025) opt to determine α heuristically while we focus on a learnable eviction method.

4 Methodology

In this section, we propose a learning-based eviction policy that prunes the KV cache based on the *intrinsic importance* of the tokens at each layer and head. The policy ranks tokens by relative importance to decide which should be evicted from the KV memory. To learn token importance, we introduce a small neural network that takes token embeddings as input and produces a scalar retention score. We then integrate this retention score into the attention computation to modulate the attention weights. We term this proxy attention mechanism a *retention-gated attention*. We train the LLM with retention-gated attention against a baseline model with standard attention, using a combination of distillation and hinge-like regularization losses to enforce memory capacity constraints while preserving response quality. A visualization is shown in Figure 2.

4.1 Selective In-Context Memory via Retention-Gated Attention

We introduce retention-gated attention, a trainable mechanism that mimics the information loss induced by inference-time eviction. From the formulation (1), the sequence $\alpha_{ii}, \alpha_{(i+1)i}, \ldots, \alpha_{ti}$ represents how token i is retained in the attention computation over time. Retention begins at 1 and then abruptly drops to 0 once the token is evicted. While this binary behavior matches the inference stage, it poses challenges for learning: the signal is discrete, non-differentiable, and provides no gradients for optimization. To remedy this, we replace the hard binary variable α with a smooth, monotonically decreasing function that models the gradual decay of importance while enabling gradient-based training. A natural candidate is the sigmoid function, $\bar{\alpha}_{ti} = 1/(1 + \exp(f(\mathbf{x}_i, t)))$, which models the time at which the token is evicted. However, this design suffers from two drawbacks: (i) the domain of f is unnormalized since the sequence length is unknown during decoding, and (ii) the sigmoid flattens across most of its range, producing negligible variation between steps and leading to vanishing gradients during training.

To overcome these limitations, we adopt an exponential decay formulation, $\bar{\alpha}_{ti} = \beta_i^{t-i}$ where $\beta_i \in [0, 1]$, to model the retention rate of token i over time. Larger values of β_i correspond to higher intrinsic importance, implying slower decay and stronger memory retention. Substituting this design for α in Equation (1) yields our proposed *retention-gated attention*:

$$\mathbf{q}_t = \mathbf{W}_Q \mathbf{x}_t, \mathbf{k}_t = \mathbf{W}_K \mathbf{x}_t, \mathbf{v}_t = \mathbf{W}_V \mathbf{x}_t, \beta_t = g(\mathbf{x}_t), \quad \mathbf{o}_t = \sum_{i=1}^{t} \frac{\exp\left(\beta_i^{t-i} \mathbf{q}_t^\top \mathbf{k}_i\right)}{\sum_{j=1}^{t} \exp\left(\beta_j^{t-j} \mathbf{q}_t^\top \mathbf{k}_j\right)} \mathbf{v}_i. \tag{3}$$

Here, we propose a *retention gate* g, which is a lightweight network, to parametrize the token importance β_t. The retention gate can be a linear projection, *i.e.*, $g(\mathbf{x}) = \sigma(\mathbf{W}_\beta \mathbf{x}_t + b), \mathbf{W}_\beta \in \mathbb{R}^{1\times d}$, or a simple MLP, *i.e.*, $g(\mathbf{x}) = \sigma(\text{MLP}(\mathbf{x}) + b)$. The sigmoid function σ squashes the output of g to the range $[0, 1]$, while b is a learnable bias. When all $\beta_t = 1, \forall t$, our retention-gated attention recovers the vanilla attention. Our ablation studies show that an MLP with a single hidden layer provides a more powerful retention estimation than a linear projection.

Brain-inspired Interpretation. Our proposed retention-gated attention bears a natural connection to the classical Ebbinghaus's forgetting curve theory (Ebbinghaus, 2013), which models human memory

retention as an exponential decay over time. A common approximation of human retention rate is $R = \exp(-tS)$, where t is the time and S is the memory strength determining how fast R decays over time in the absence of training (Woźniak et al., 1995).

In a similar spirit, our retention-gated attention models the contribution of past tokens as an exponentially decreasing function of their temporal distance from the current step, *i.e.*, $\exp((t-i)\log\beta_i)$. Each token begins with full weight ($\bar{\alpha}_{ii} = 1$), akin to a newly encoded memory, and its influence decays as more tokens arrive, mirroring the drop in recall probability described by the forgetting curve. The parameter β acts analogously to memory strength S: larger values yield more persistent, durable memories, while smaller values indicate weaker memories that fade quickly.

This connection provides an intuitive justification for our design. By embedding a forgetting mechanism into attention, we enable the model to dynamically prioritize recent or intrinsically important tokens while gradually discarding less informative context, mirroring how humans manage limited memory capacity in practice. Note that Zhong et al. (2024) also drew on Ebbinghaus's forgetting curve to construct a long-term memory bank, but their focus was on retrieval-augmented generation, whereas our approach integrates forgetting directly into the attention mechanism.

4.2 Training

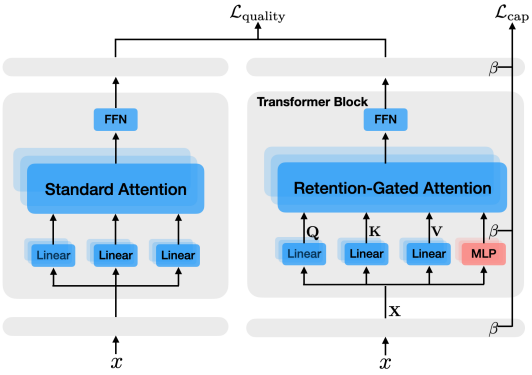

Figure 2: Training architecture.

Our goal is to train the retention gate g so that the LLM can preserve response quality under a memory constraint, thereby bridging the gap with the inference stage. Instead of training a separate gate for each layer and head, as formulated in Problem (2), we optimize all retention gates jointly in an end-to-end fashion. This holistic approach mitigates error propagation, allowing the model to learn a coordinated, globally optimal caching policy rather than greedy layer-wise decisions. Starting from a pretrained LLM, we replace every standard attention block with our proposed retention-gated attention. Each block is equipped with a lightweight retention gate g that maps token representations to retention scores $\beta_t \in [0, 1]$, which are then used to modulate attention weights according to Equation (3). We call this proxy LLM a retention-gated LLM.

Objectives. To train these retention gates, we formulate the training objective that balances two goals: (i) preserving the predictive quality of the original pretrained LLM, and (ii) enforcing memory capacity constraints by controlling the sum of retention scores at each step.

For the first objective, we use a combination of the distillation and standard next-token prediction losses. The distillation loss encourages the proxy LLM to align its output distribution with that of the baseline LLM using standard attention. In parallel, the next-token prediction loss enables the model to uncover sparsity patterns directly from the data, extending beyond the constraints of the pretrained LLM. Let $p(\cdot)$ and $q_\theta(\cdot)$ be the output distribution of the pretrained LLM and retention-gated LLM, respectively, where θ denotes the parameters of all retention gates. The quality loss is given by

$$\mathcal{L}_{\text{quality}} = \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{NTP}} = \mathcal{D}_{\text{KL}}\big(p(\cdot|x) \,\|\, q_\theta(\cdot|x)\big) + \mathbb{E}_{(x,y)}[-\log q_\theta(y|x)]\,. \tag{4}$$

Here, $\mathcal{D}_{\text{KL}}$ is the standard forward Kullback-Leibler divergence (Kullback and Leibler, 1951).

For the second objective, we impose a hinge-like regularization penalty, which discourages the model from exceeding the available KV memory slots at each step. For a retention gate within a given layer and KV head, the memory capacity loss is defined as:

$$\mathcal{L}_{\text{cap}} = \frac{1}{T(T-M)} \sum_{t=1}^{T} \max\{0, \sum_{i=1}^{t} \beta_i^{t-i} - M\}, \tag{5}$$

where T is the sequence length and M is the predefined memory capacity. Here, M acts as a *soft* hyperparameter, primarily intended to prevent over-optimization during the early decoding stage when the sequence remains short. Training is performed with a fixed value of M, while inference can flexibly accommodate different KV budgets. This regularization is applied uniformly across all

layers and KV heads of the transformer. The combined training objective is then:

$$\min_{\theta} \mathcal{L}_{\text{quality}} + \lambda_{\text{cap}} \mathcal{L}_{\text{cap}}, \tag{6}$$

where λ_{cap} is a hyperparameter balancing between quality and capacity loss. Note that during training, only the retention gate parameters are updated, while all other model weights remain frozen.

Hardware-aware Computation. Retention-gated attention is fully parallelizable and compatible with FlashAttention-style kernels (Dao, 2023). We implement it with FlexAttention (Dong et al., 2024) plus a custom Triton kernel for the capacity loss $\mathcal{L}_{\text{cap}}$, performing forward/backward without materializing the full attention or β matrices. This enables long-context training (up to 128K tokens on four H100 GPUs) with minor overhead versus standard parameter-efficient fine-tuning.

4.3 Inference

At inference time, the base LLM is augmented with the retention gates learned during training (Section 4.2). These gates provide token-level intrinsic importance scores β_i, which quantify how strongly each past token should be retained for future computations. Unlike training, where the retention gates are used to modulate the attention weights, at inference, they act purely as decision-makers for eviction, operating alongside but independently of attention computation.

The eviction process is designed to ensure that the KV cache respects a predefined memory budget. Let $S_t \subseteq \{1, \ldots, t\}$ denote the set of tokens currently stored in the KV cache at decoding step t. When a new token $t+1$ is generated, it is provisionally added to the cache. If this addition causes the cache size to exceed the memory capacity M, an eviction is triggered. The eviction rule is simple yet principled: we remove the token with the lowest retention score, i.e.,

$$j_{\text{evic}} = \arg\min_{j \in S_t} \{\beta_j^{t-j} | j \in S_t\}.$$

Intuitively, this favors retaining tokens deemed globally important by the learned retention gates while discarding those with little long-term value. In practice, this makes inference both memory-efficient and adaptive: as new context arrives, the model continually re-evaluates the importance of older tokens, enabling long-context generation while keeping memory usage bounded. Algorithm 1 illustrates a single decoding step, where attention computation is coupled with token eviction.

Complexity. Our inference is simpler and more efficient than existing works, including pure heuristic baselines (Li et al., 2024c). Throughput and runtime comparisons are in Appendix A.2.

5 Experiments

In this section, we conduct extensive experiments to demonstrate the performance advantages of our method on both *long-context* and *long generation* tasks.

5.1 Long Generation Evaluation

Benchmarks. Following prior work (Gao et al., 2025; Cai et al., 2025), we evaluate on standard math-reasoning suites—AIME24 (Art of Problem Solving, 2024), GSM8K (Cobbe et al., 2021), and MATH-500 (Hendrycks et al., 2021). To assess performance beyond math reasoning and under long-context settings, we also report results on LongProc (Ye et al., 2025). Following (Gao et al., 2025), we report average pass@1 accuracy over 64 samples for AIME24 and 8 samples for GSM8K, MATH-500. We use greedy decoding for LongProc as the default in the benchmark.

Base models. Following (Gao et al., 2025), we mainly use Qwen3's family models (Yang et al., 2025), including Qwen3-1.7B, Qwen3-4B, Qwen3-8B, Qwen3-14B and DeepSeek R1 Distill (Guo et al., 2025) variants including, DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Lllam-8B. We report the results with Qwen3 models in the main paper, and the remaining is in Appendix B.

Baselines. We compare our method against SeerAttn-R (Gao et al., 2025), R-KV (Cai et al., 2025), SnapKV (Li et al., 2024c), H2O (Zhang et al., 2023), StreamingLLM (Xiao et al., 2023). R-KV, SnapKV, H2O, and StreamingLLM are heuristic, recency-driven KV *eviction* policies for long-form generation under a fixed memory budget. SeerAttn-R is a learnable KV *retrieval* approach for reasoning tasks: rather than evicting, it offloads the full KV cache to host memory and uses recent

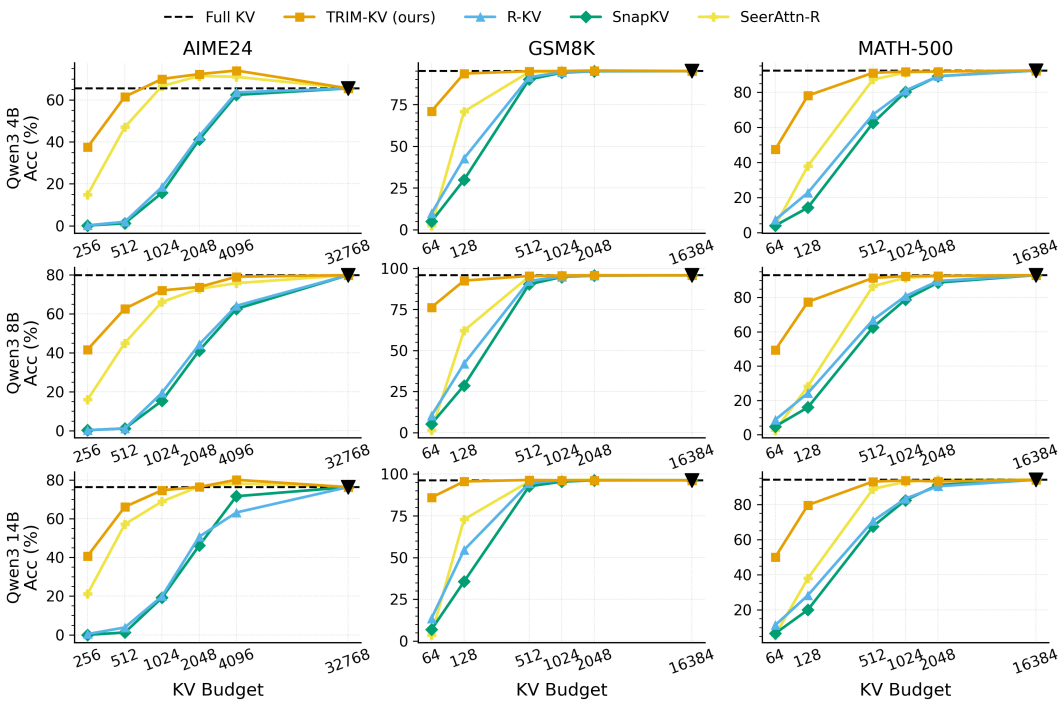

Figure 3: Patero frontiers of competing algorithms with different budgets on math benchmarks.

queries to fetch relevant blocks for attention. KV retrieval methods preserve all past information but require nontrivial CPU–GPU orchestration and incur offloading overhead. We therefore treat SeerAttn-R as a strong learnable baseline, and R-KV/SnapKV as representative eviction baselines.

Implementation details. We train the retention gates using OpenR1-MATH-220k (Hugging Face) dataset, similar to (Gao et al., 2025). Note that we only train the retention gates' weights while keeping the original model parameters frozen. We set the objective hyperparameters $\lambda_{\text{cap}} = 1.0$ and the memory capacity $M = 512$. Each transformer block has a retention gate g, which is a single MLP layer with the hidden dimension of 512, thus having dimensions of $d \to 512 \to h$, where h is the number of KV heads. We use the activation function as the default activation function in MLP layers of the base model. We initialize the bias in the retention gates to a large value (e.g., $b = 8.0$) to begin training with minimal forgetting or compression. All trainings are on 4 H100 80G GPUs.

5.1.1 Quantitative Result

Math Reasoning Tasks. Figure 3 shows our method outperforming all baselines by a large margin, especially in low-budget regimes. Notably, TRIM-KV surpasses attention-guided methods (R-KV, SnapKV) even when they are given $4\times$ KV budget. Under the same budget, *i.e.*1024 for AIME24 and 128 for GSM8K/MATH-500, it yields a **198%** relative improvement over these baselines. Against the SOTA learnable KV retrieval baseline, TRIM-KV outperforms SeerAttn-R across all settings, yielding a **58.4%** pass@1 gain at the same budget. Crucially, TRIM-KV operates in a pure KV-eviction regime, a stricter setting than the KV retrieval methods such as SeerAttn-R, and thus avoids CPU–GPU offloading overhead. In some settings, like for Qwen3-4B model and AIME24 dataset, our method can even surpass the standard full KV cache. These results suggest that a large fraction of KV-cache tokens in reasoning models is redundant and can be discarded without degrading performance.

Long Procedural Generation Tasks. We evaluate KV-eviction methods on tasks that require both long-context comprehension and extended generation. Table 1 reports results with Qwen3-4B model. Overall, TRIM-KV consistently outperforms all other eviction baselines and, in several settings, even surpasses the full-cache model. Moreover, this result highlights that TRIM-KV with retention gates trained on math-reasoning data generalizes well to non-math tasks. Full results and analysis are provided in Appendix B.

$\text{Method}_{\text{KV budget}}$	CountDown			Pseudo to Code	
	0.5k	2k	8k	0.5k	2k
FullKV	96.0	90.0	**69.0**	**50.8**	**25.0**
$\text{StreamingLLM}_{2048}$	7.0	5.0	5.0	20.6	1.5
H2O_{2048}	12.0	7.5	2.5	33.7	0.5
SnapKV_{2048}	57.0	49.0	13.0	42.7	4.5
R-KV_{2048}	88.5	81.0	63.0	48.2	2.5
TRIM-KV_{2048}	**97.5**	**93.5**	66.0	49.2	19.0

Table 1: Qwen3-4B on LongProc. Bold is for the best, underline is for the best KV eviction.

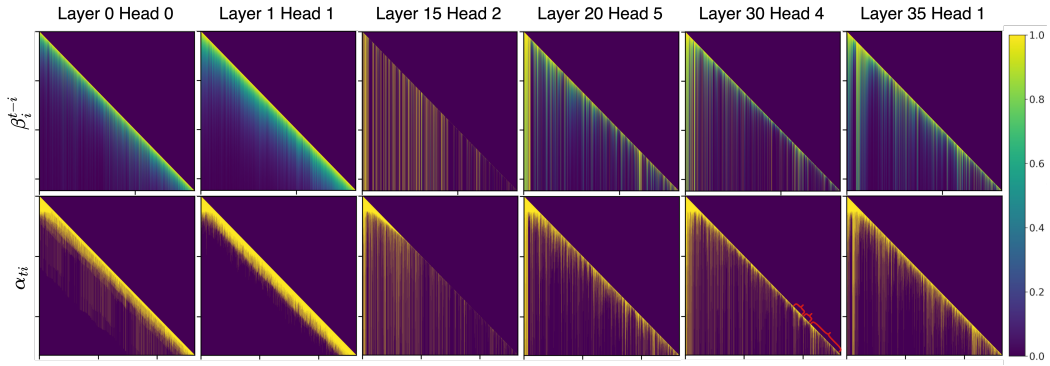

Figure 4: Visualization of token retention score β_i^{t-i} (top) and eviction decisions α_{ti} (bottom).

5.1.2 Qualitative Result

To examine the eviction policy learned by our retention gates, we run TRIM-KV on Qwen3-4B for the first example in AIME24 (see Figure 13 for a visualization of the example). Figure 5**a**–**b** show the mean retention score—averaged over all layers and heads—for each token in the example sequence. Aligning with our intuition, retention gates assign high scores to task-relevant tokens (e.g., `ometer`, `shop`, `walk`, `minutes`) and to the initial token `<|im_start|>`, which often serves as an attention sink. In contrast, whitespace and punctuation receive low retention scores and are discarded early, yielding short lifespans in the KV cache. Next, we examine retention scores and eviction decisions at layer–head granularity.

Emergent Eviction Heuristics. Figure 4 visualizes the retention scores β_i^{t-i} and eviction decisions α_{ti} for selected layers and heads. Many eviction heuristics, such as attention sinks (Xiao et al., 2023), sliding windows (Zhu et al., 2021), A-shape (Jiang et al., 2024), emerge naturally from our learned policy without being hard-coded, and they adapt to the functional roles of individual layers and heads. For instance, sliding-window behavior is more common in early layers, whereas attention sinks appear more frequently in later layers (see Figure 11 and 12 for a comprehensive view). Moreover, TRIM-KV adapts the window size by layer and head: in *Layer 1/Head 1*, tokens receive nearly uniform retention scores, so the KV cache behaves like a recency-biased window that keeps the most recent tokens; in *Layer 0/Head 0*, multiple sliding windows of varying widths emerge from the learned policy; in *Layer 15/Head 2*, no sliding window is observed because certain tokens receive substantially higher retention than others, suggesting a specialized functional role for this head. The A-shaped pattern typically appears in layers that emphasize instruction/problem-statement tokens (*e.g.*, *Layer 20/Head 5* and *Layer 30/Head 4*) or chain-of-thought/reasoning prompts (*e.g.*, *Layer 35/Head 1*). These heads also exhibit context switching, where small, dense lower-triangular blocks emerge and then fade quickly when the context changes or a sentence completes. To the best of our knowledge, the absence of a sliding window, the presence of multiple coexisting windows, and context switching are newly observed eviction patterns that arise naturally from our learned policy.

Token Retention Enables Interpretability. Beyond guiding eviction policy, token-level retention scores provide a diagnostic tool for analyzing the functional roles of individual KV heads in the base LLM. Visualizing retention scores alongside the tokens preserved in the KV cache after generation reveals distinct specializations: some heads emphasize a recency window (Figure 16), whereas others preferentially retain mathematical tokens-numbers and operators (Figures 13 and 19)-or variables (Figure 22), as well as problem-description tokens (Figures 18 and 24) and chain-of-thought instructions (Figure 25). Even function or filler words, such as pronouns, prepositions, conjunctions, `wait` and `let`, tend to be kept by specific heads (Figures 17 and 23). In particular, heads that exhibit context-switching patterns (e.g., *Layer 30/Head 4* and *Layer 16/Head 6*) tend to retain the period token while discarding others (Figures 21 and 20). We hypothesize that, in these heads, periods act as implicit *gist* tokens (Mu et al., 2023), summarizing the information in the preceding sentences.

Our analyses indicate that KV heads in LLM develop different functional roles and therefore keep different types of tokens. These tokens are often dispersed across the context rather than forming contiguous chunks, as each already captures contextual information. This observation contrasts with existing approaches (Yuan et al., 2025; Gao et al., 2025) that advocate chunk- or block-based KV-cache. Instead, we show that keeping a small number of high-context tokens is more budget-effective.

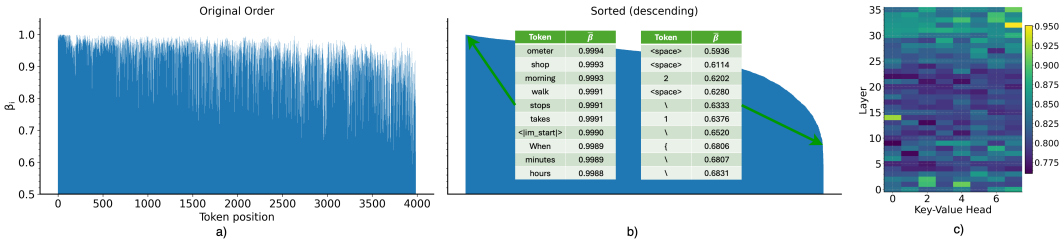

Figure 5: *a)* Average retention scores across all layers and heads of Qwen3-4B on tokens of an AIME24 example. *b)* Top 10 tokens with the highest (left table) and lowest (right table) average retention. *c)* The layer- and head-wise sparsity level estimated by token retentions.

Budget Allocation. Figure 5**c** reports head- and layer-wise sparsity estimated from the scores via *i.e.*, $1 - \frac{2}{T(T+1)} \sum_{i<t} \beta_i^{t-i}$. We observe that later layers are typically sparser than earlier ones, consistent with prior findings in (Cai et al., 2024). Practically, these scores enable heterogeneous budgets across KV heads under a global constraint by evicting tokens with low global retention. However, existing KV-cache and FlashAttention implementations assume uniform sequence lengths across heads within a layer; enabling efficient per-head variable-length caches is left to future work.

5.2 Long Memory Evaluation

To further stress-test our approach on long-context tasks, we evaluate it on LongMemEval$_S$ (Wu et al., 2024a), a benchmark for assessing chat assistants' long-term interactive memory (contexts up to 123K tokens with Qwen3). We use Qwen3-4B-Instruct (Qwen3, 2025) as the base model and train the retention gates on the SynthLong dataset (Lazarevich et al., 2025). All other settings follow Section 5.1. As shown in Table 2, TRIM-KV outperforms baselines by a significant margin. Especially, TRIM-KV can maintain the performance of a full cache while using just 25% of the KV budget. These results highlight our method's advantage on both long-context and long-generation tasks, whereas most prior work targets either the prefill or the generation stage, but not both. More details and results for this experiment are provided in Appendix B.

Method$_{\text{KV budget}}$	Acc
Full KV$_{131072}$	49.4
StreamingLLM$_{32768}$	27.6
SnapKV$_{32768}$	27.8
TRIM-KV$_{32768}$	**48.2**
StreamingLLM$_{16384}$	19.0
SnapKV$_{16384}$	18.2
TRIM-KV$_{16384}$	**42.6**
StreamingLLM$_{4096}$	10.2
SnapKV$_{4096}$	13.4
TRIM-KV$_{4096}$	**30.2**

Table 2: Results on LongMemEval$_S$.

5.3 Ablation Studies

We ablate the objective by training the Qwen3-4B retention gates with different loss combinations and report AIME24 pass@1 at a 4096-token budget in Table 3. Both forward KL and next-token prediction perform well on their own, and their combination further improves accuracy. The memory capacity loss is essential for compression, and removing it leads to a sharp drop. We provide comprehensive ablations with reversed KL, generalization with different training datasets, gate's architecture, and other hyperparameters such as M in Appendix B.3.

Method$_{\text{KV budget}}$	pass@1
Full KV$_{32768}$	65.5
TRIM-KV$_{4096}$	**74.0**
(TRIM-KV $-\mathcal{L}_{\text{KL}}$)$_{4096}$	71.4
(TRIM-KV $-\mathcal{L}_{\text{NTP}}$)$_{4096}$	70.7
(TRIM-KV $-\mathcal{L}_{\text{cap}}$)$_{4096}$	42.9

Table 3: Objective ablation.

6 Conclusion and Future Work

We introduced TRIM-KV, a learnable, retention-gated approach to KV cache management that prioritizes tokens by intrinsic importance rather than recent attention. By training lightweight gates with distillation and a capacity loss, our method enforces strict memory budgets with a simple and efficient eviction policy. Extensive experiments across math reasoning, long-procedural generation, and conversational long-memory benchmarks demonstrate that our method outperforms strong eviction and retrieval baselines—sometimes even surpassing full-cache models. Analyses show that the learned retention scores align with human intuitions and reveal layer- and head-specific dynamics, offering a simple probe for interpretability. Building on these results, we plan to extend retention gating to multimodal inputs and develop adaptive budgets that allocate memory across layers, heads, and tasks to further improve both performance and efficiency.

Ethics Statement

This work aims to improve the efficiency of large language models by reducing their memory and computational footprint. Our method can make long-context reasoning more accessible by lowering hardware costs, which may democratize access to advanced LLM capabilities. However, efficiency improvements may also accelerate the deployment of LLMs in high-stakes or resource-limited settings where risks around misinformation, bias, or misuse persist. We stress that our method does not mitigate these broader societal risks and should be paired with ongoing efforts in safety, fairness, and responsible deployment.

Reproducibility Statement

We ensure reproducibility by providing detailed descriptions of the model architecture, training objectives, and evaluation benchmarks in the main text and appendix. Hyperparameters, training schedules, and implementation details are included for all experiments. All datasets we use are publicly available, and we will release code, model checkpoints, and scripts for training and evaluation upon publication. Together, these materials allow independent researchers to fully reproduce and verify our results.

The authors used large language models to help refine and polish the writing of this manuscript.

References

Art of Problem Solving. AIME problems and solutions, 2024. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

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

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. Nacl: A general and effective kv cache eviction framework for llms at inference time. *arXiv preprint arXiv:2408.03675*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024.

Hermann Ebbinghaus. [image] memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155, 2013.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). *arXiv preprint arXiv:2410.02660*, 2024.

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

Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant Nair, and Poulami Das. Dialogue without limits: Constant-sized kv caches for extended responses in llms. *arXiv preprint arXiv:2503.00979*, 2025.

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

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Zero-shot extreme length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

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

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1. URL `https://github.com/huggingface/open-r1`.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Mahdi Karami and Vahab Mirrokni. Lattice: Learning to efficiently compress the memory. *arXiv preprint arXiv:2504.05646*, 2025.

Mahdi Karami, Ali Behrouz, Praneeth Kacham, and Vahab Mirrokni. Trellis: Learning to compress key-value memory in attention models. In *Second Conference on Language Modeling*.

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

Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International conference on algorithmic learning theory*, pages 597–619. PMLR, 2023.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. 2021.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Ivan Lazarevich, David Bick, Harsh Gupta, Srinjoy Mukherjee, Nishit Neema, Gokul Ramakrishnan, and Ganesh Venkatesh. Extending llm context with 99% less training tokens. `https://cerebras.ai/blog/extending-llm-context-with-99-less-training-tokens`, February 2025.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442*, 2024a.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. Scbench: A kv cache-centric analysis of long-context methods. *arXiv preprint arXiv:2412.10319*, 2024b.

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

Zhuoling Li, Xiaogang Xu, Zhenhua Xu, SerNam Lim, and Hengshuang Zhao. Larm: Large auto-regressive model for long-horizon embodied intelligence. *arXiv preprint arXiv:2405.17424*, 2024d.

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

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

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

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Chris Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *arXiv preprint arXiv:2504.15364*, 2025.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. *arXiv preprint arXiv:2503.12491*, 2025.

Qwen3. Qwen/qwen3-4b-instruct-2507, 2025. URL `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression. *arXiv preprint arXiv:2408.05646*, 2024.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR, 2023.

Chaitanya Singhal. Introducing buddhi: Open-source chat model with a 128k context window. AI Planet (Medium), April 2024. URL `https://huggingface.co/datasets/aiplanet/buddhi-dataset`. Accessed: 2025-08-23.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024a.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024b.

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

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

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. Llms know what to drop: Self-attention guided kv cache eviction for efficient long-context inference. *arXiv preprint arXiv:2503.08879*, 2025.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Piotr Woźniak, Edward Gorzelańczyk, and Janusz Murakowski. Two components of long-term memory. *Acta neurobiologiae experimentalis*, 55(4):301–305, 1995.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024a.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024b.

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

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

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. *arXiv preprint arXiv:2501.05414*, 2025.

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

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

Zihao Zeng, Bokai Lin, Tianqi Hou, Hao Zhang, and Zhijie Deng. In-context kv-cache eviction for llms via attention-gate. *arXiv preprint arXiv:2410.12876*, 2024.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*, 2024.

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

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *arXiv preprint arXiv:2501.07278*, 2025.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *Advances in neural information processing systems*, 34:17723–17736, 2021.

A METHODOLOGY

A.1 INFERENCE ALGORITHM.

Algorithm 1 illustrates the attention computation with KV eviction using retention gates for a single decoding step. We mark the parts that are different from the standard attention computation in blue.

Algorithm 1: Attention computation with KV eviction (single decoding step)

Input : current hidden $\mathbf{x}_t$; KV cache $\mathbf{K}_{t-1}, \mathbf{V}_{t-1}, \mathbf{B}_{t-1}$ indexed by S_{t-1}; retention gate g
Param : capacity M;
Output : attention output $\mathbf{o}_t$; updated $(\mathbf{K}_t, \mathbf{V}_t, \mathbf{B}_t)$; updated index set S_t

// 1) Project to Q/K/V for the current token
$\mathbf{q}_t \leftarrow \mathbf{W}_Q \mathbf{x}_t$; $\mathbf{k}_t \leftarrow \mathbf{W}_K \mathbf{x}_t$; $\mathbf{v}_t \leftarrow \mathbf{W}_V \mathbf{x}_t$; $\beta_t = g(\mathbf{x}_t)$;
// 2) Append current token to the KV cache
$\mathbf{K}_t \leftarrow \mathbf{K}_{t-1} \cup \mathbf{k}_t$; $\mathbf{V}_t \leftarrow \mathbf{V}_{t-1} \cup \mathbf{v}_t$; $\mathbf{B}_t \leftarrow \mathbf{B}_{t-1} \cup \beta_t$; $S_t \leftarrow S_{t-1} \cup \{t\}$;
// 3) Compute attention over currently cached keys/values (restricted to S_t)
$\mathbf{o}_t \leftarrow \text{FLASHATTN}(\mathbf{q}_t, \mathbf{K}_t, \mathbf{V}_t)$;
// 4) If capacity exceeded, evict the least important token
if $|S_t| > M$ **then**
  $j_{\text{evic}} \leftarrow \arg\min \{\beta_j^{t-j} | j \in S_t\}$;
  Remove $\mathbf{K}_t[j_{\text{evic}}]$, $\mathbf{V}_t[j_{\text{evic}}]$, $\mathbf{B}_t[j_{\text{evic}}]$;
  $S_t \leftarrow S_t \setminus \{j_{\text{evic}}\}$;
end

A.2 COMPLEXITY

Memory efficiency. Like other KV-eviction schemes, TRIM-KV uses a fixed-size cache with $\mathcal{O}(M)$ slots, independent of sequence length T. For each token (per head), it stores a single scalar retention score β_i, adding only $\approx 1/d_h$ overhead, where d_h is the dimension of the key and vector states, relative to the KV states, which is negligible in practice. Unlike R-KV (Cai et al., 2025), TRIM-KV does not store queries.

Method	Context Length	Gen Length	Batch Size	Throughput (tok/sec)	Decode Time (s)
FullKV	32786	1024	4	68.44	59.84
SeerAttn-R				68.93	59.41
SnapKV				124.67	33.00
TRIM-KV				**130.48**	**31.39**
FullKV	16378	1024	4	114.39	35.8
SeerAttn-R				100.45	40.77
SnapKV				**153.21**	**26.73**
TRIM-KV				151.04	27.11
FullKV	16378	1024	8	138.97	58.94
SeerAttn-R				139.34	58.78
SnapKV				244.60	33.49
TRIM-KV				**279.90**	**29.26**

Table 4: Throughput and decoding time comparisons of different KV cache methods on a single H200 GPU.

Computational efficiency. For each generated token, TRIM-KV computes a scalar retention score β_i via a lightweight MLP that can be fused with QKV projections; scores are cached and not recomputed each step. During compression, it applies a temporal discount (elementwise power) and

evicts the argmin; both costs only $\mathcal{O}(M)$. This is cheaper than heuristics like R-KV, which require key–key similarity scoring over the cache. Table 4 reports throughput and latency: at 32K context, TRIM-KV achieves $\sim 2\times$ higher decoding throughput than full-cache decoding and even faster than SnapKV, a purely heuristic method. SeerAttn-R does not provide any computation advantage over full cache model.

B Additional Experiments

B.1 Long Generation Evaluation

We provide more comprehensive experiment details in this section.

Experiment settings. For the training, we set the maximum training length to be 16384. We train the retention gates with a learning rate of 2×10^{-4} and a weight decay of 0.01. Other hyperparameters are set to the default in Huggingface Trainer.

Benchmarks. AIME24 (Art of Problem Solving, 2024), GSM8K (Cobbe et al., 2021), and MATH-500 (Hendrycks et al., 2021) are standard math reasoning benchmarks. LongProc (Ye et al., 2025) is a long-context benchmark of six procedural-generation tasks that require integrating dispersed information and producing structured long-form outputs (up to $\sim$ 8K tokens)—from extracting tables from HTML to executing multi-step search to build feasible travel itineraries. The suite spans varied access patterns (sequential vs. targeted retrieval), deductive reasoning demands, and search execution, enabling stress tests of long-range coherence and procedure following. Each task includes deterministic solution procedures and structured outputs, allowing rule-based evaluation (row-level F1 for HTML→TSV, unit tests for pseudocode→code, exact-match traces for Path/ToM, and validators for Countdown/Travel). To probe generation length, we use three difficulty tiers targeting 0.5K, 2K, and 8K output tokens.

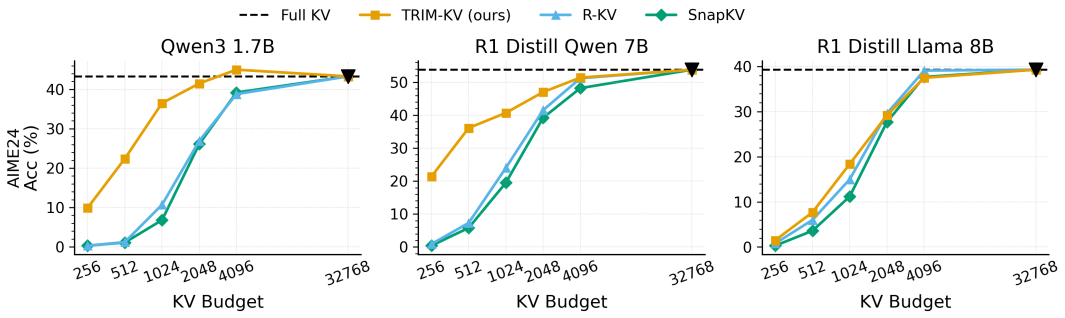

Figure 6: Patero frontiers of competing algorithms with different budgets on AIME24.

Math reasoning results. Figure 6 reports AIME24 performance for Qwen3-1.7B and DeepSeek-R1-Distill variants. Across both families, TRIM-KV consistently outperforms eviction baselines. The gains over heuristic baselines are smaller on DeepSeek-R1-Distill–Llama-8B, which we hypothesize reflects lower attention sparsity in this model compared to the Qwen3 family.

Results on LongProc. Table 5 reports KV-eviction results on long procedure–generation tasks. Across tasks and budgets, TRIM-KV achieves the best performance, and it even surpasses the full-cache baseline on Countdown (0.5K/2K) and HTML to TSV (0.5K). Under tighter memory budgets, its margin over heuristic baselines widens.

B.2 Long Memory Evaluation

Experimental settings. We adopt Qwen3-4B-Instruct (Qwen3, 2025) as the base model, which supports a context window of up to 256K tokens. Retention gates are trained on a mixture of SynthLong-32K (Lazarevich et al., 2025), BookSum (Kryściński et al., 2021), and Buddhi (Singhal, 2024), covering sequence lengths from 32K to 128K tokens. We shuffle the combined corpus and train for 10,000 steps (i.e., 10,000 randomly sampled examples), with a maximum training sequence length of 128K and memory capacity $M = 4096$. All other settings follow Section 5.1.

Method$_{\text{KV budget}}$	HTML to TSV			Thought of Mind			Travel Planning	
	0.5k	2k	8k	0.5k	2k	8k	2k	8k
FullKV	49.0	**41.6**	**13.9**	33.0	**10.5**	0.0	0.0	0.0
SnapKV$_{8192}$	37.1	9.3	0.1	26.0	7.0	0.0	0.0	0.0
H2O$_{8192}$	28.3	6.4	0.4	**38.0**	7.0	0.0	0.0	0.0
R-KV$_{8192}$	38.0	7.1	0.5	26.0	7.5	0.0	0.0	0.0
TRIM-KV$_{8192}$	**58.2**	36.0	12.5	32.5	**10.5**	0.0	0.0	0.0
StreamingLLM$_{2048}$	1.2	0.0	0.0	2.0	0.0	0.0	0.0	0.0
SnapKV$_{2048}$	1.5	0.2	0.0	15.0	0.0	0.0	0.0	0.0
H2O$_{2048}$	0.4	0.8	0.0	7.6	0.0	0.0	0.0	0.0
R-KV$_{2048}$	1.6	0.1	0.0	3.0	0.0	0.0	0.0	0.0
TRIM-KV$_{2048}$	34.6	7.1	0.3	17.5	0.5	0.0	0.0	0.0

Table 5: Results of Qwen3-4B across LongProc tasks: F1-score for HTML to TSV task and accuracies (%) for the remaining tasks. Best per task column in bold.

Method$_{\text{KV budget}}$	Overall	Multi	Knowledge	SS-User	SS-Pref	SS-Assist	Temporal
Full KV$_{131072}$	49.4	25.6	68.0	62.9	93.3	85.7	30.1
StreamingLLM$_{32768}$	27.8	15.8	50.0	32.9	56.7	33.9	15.0
SnapKV$_{32768}$	27.6	15.8	42.3	24.3	73.3	28.6	21.8
TRIM-KV$_{32768}$	**48.2**	**23.3**	**68.0**	**58.6**	**80.0**	**85.5**	**32.3**
StreamingLLM$_{16384}$	19.0	12.8	35.9	24.3	26.7	17.9	11.3
SnapKV$_{16384}$	18.2	9.0	25.6	17.1	70.0	12.5	14.3
TRIM-KV$_{16384}$	**42.6**	**21.8**	**62.8**	**42.9**	**80.0**	**69.6**	**31.6**
StreamingLLM$_{12288}$	17.2	13.5	32.1	20.0	33.3	14.3	8.3
SnapKV$_{12288}$	17.0	9.8	20.5	14.3	**73.3**	12.5	12.8
TRIM-KV$_{12288}$	**36.8**	**19.6**	**59.0**	**35.7**	66.7	**50.0**	**29.3**
StreamingLLM$_{8192}$	13.0	9.0	25.6	15.7	16.7	10.7	8.3
SnapKV$_{8192}$	15.8	9.8	19.2	12.9	**70.0**	7.1	12.8
TRIM-KV$_{8192}$	**33.2**	**15.0**	**55.1**	**34.3**	**70.0**	**46.4**	**24.1**
StreamingLLM$_{4096}$	10.2	9.8	14.1	14.3	16.7	7.1	6.0
SnapKV$_{4096}$	13.8	11.3	14.1	14.3	56.7	7.1	9.0
TRIM-KV$_{4096}$	**30.2**	**12.8**	**35.9**	**24.3**	**80.0**	**46.4**	**29.3**

Table 6: Results on LongMemEval$_S$: overall and partial accuracies (%).

Benchmark Dataset. We evaluate chat-assistant capabilities under strict memory budgets using LongMemEval$_S$ (Wu et al., 2024a). This subset provides contexts up to 123k tokens (measured with the Qwen3 tokenizer) and includes six question types that probe long-term memory: *single-session-user* (SS-User) and *single-session-assistant* (SS-Assist), which test recall of facts stated by the user or assistant within a session; *single-session-preference* (SS-Pref), which requires personalized responses from stored personal information; *multi-session* (Multi), which aggregates or compares information across sessions; *knowledge-update* (Knowledge), which tracks and answers with the most recent, changed user information; *temporal-reasoning* (Temporal), which reasons over timestamps and time references.

To evaluate KV-cache eviction methods on this benchmark, we follow the multi-turn, multi-session protocol of (Li et al., 2024b). Specifically, before each query, the eviction-based model must compress the accumulated dialogue into a fixed-size, reusable KV cache—mirroring real-world assistants that maintain state across turns and sessions under strict memory budgets. We use Qwen3-4B-Instruct (Qwen3, 2025) to assess whether model outputs match the ground-truth responses.

Results. The results in Table 6 show that our method outperforms baseline eviction strategies by a significant margin. Especially, TRIM-KV can match the performance of a full cache while using just 25% of the KV budget.

B.3 Additional Ablation Studies

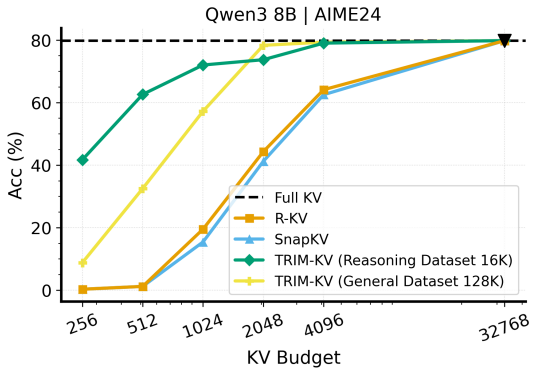

Figure 7: Generation Ablation.

Ablation on training datasets. In Section 5.1, we train the retention gates on a reasoning dataset—OpenR1-Math (Hugging Face)—and evaluate on AIME24, MATH-500, and GSM8K. This follows standard practice and matches the setting of (Gao et al., 2025), ensuring a fair comparison. To assess cross-domain generalization, we instead train the gates on general long-context datasets (SynthLong, BookSum, Buddhi), similar to Section 5.2, and then evaluate on math reasoning benchmarks to test whether retention scores learned from general data transfer to long chain-of-thought tasks. As shown in Figure 7, gates trained on general datasets generalize well and even surpass math-specific training at a 2048 KV budget. However, their performance degrades more quickly under tighter KV budgets. Overall, these results are promising and suggest that scaling the training of the retention gates by combining all datasets could yield further improvements.

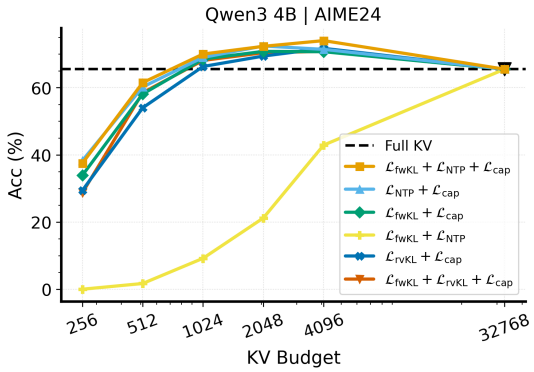

Figure 8: Ablating different objective components.

Ablation for the objective function. We ablate the objective by training the Qwen3-4B retention gates with different loss combinations and report AIME24 pass@1 at a 4096-token KV budget in Figure 8. Here, we consider both forward KL divergence and reversed KL divergence for distillation loss. Generally, all distillation losses perform well on their own. However, reversed KL underperforms when compared to forward KL. Both show benefits in combination with next token prediction. The memory capacity loss is essential for compression, and removing it leads to a sharp drop.

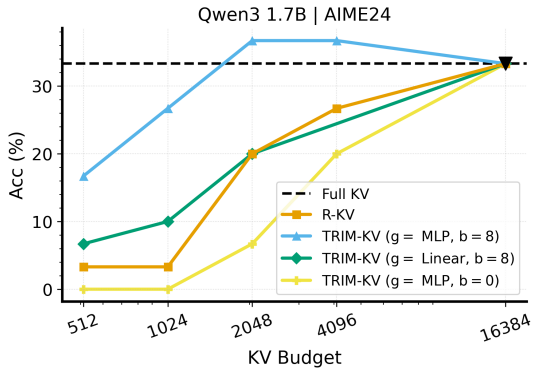

Figure 9: Ablating the retention gate's architecture.

Ablation for the retention gate's architecture. We evaluate several retention-gate architectures and report the performance of Qwen3 1.7B on AIME24 in Figure 9. Due to computational constraints, this ablation uses greedy decoding. For the MLP gate, we use a single-hidden-layer MLP with width 512. We find that the MLP gate outperforms a simple linear projection, and that a large positive initial bias is crucial for stable training by keeping the gate's output nearly 1 at initialization to ensure minimal early forgetting.

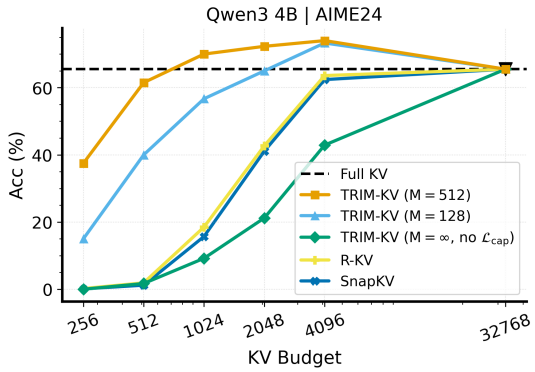

Figure 10: Ablating the training memory capacity M.

Ablation on training memory capacity M. We evaluate multiple settings of M in Figure 10. With $M = \infty$, there is no capacity penalty, which hurts performance due to the absence of sparsity pressure. Setting $M = 128$ outperforms attention-guided heuristics but shows signs of over-optimizing for sparsity. In practice, we recommend choosing M to match the expected deployment-time memory budget.

C Additional Qualitative Results

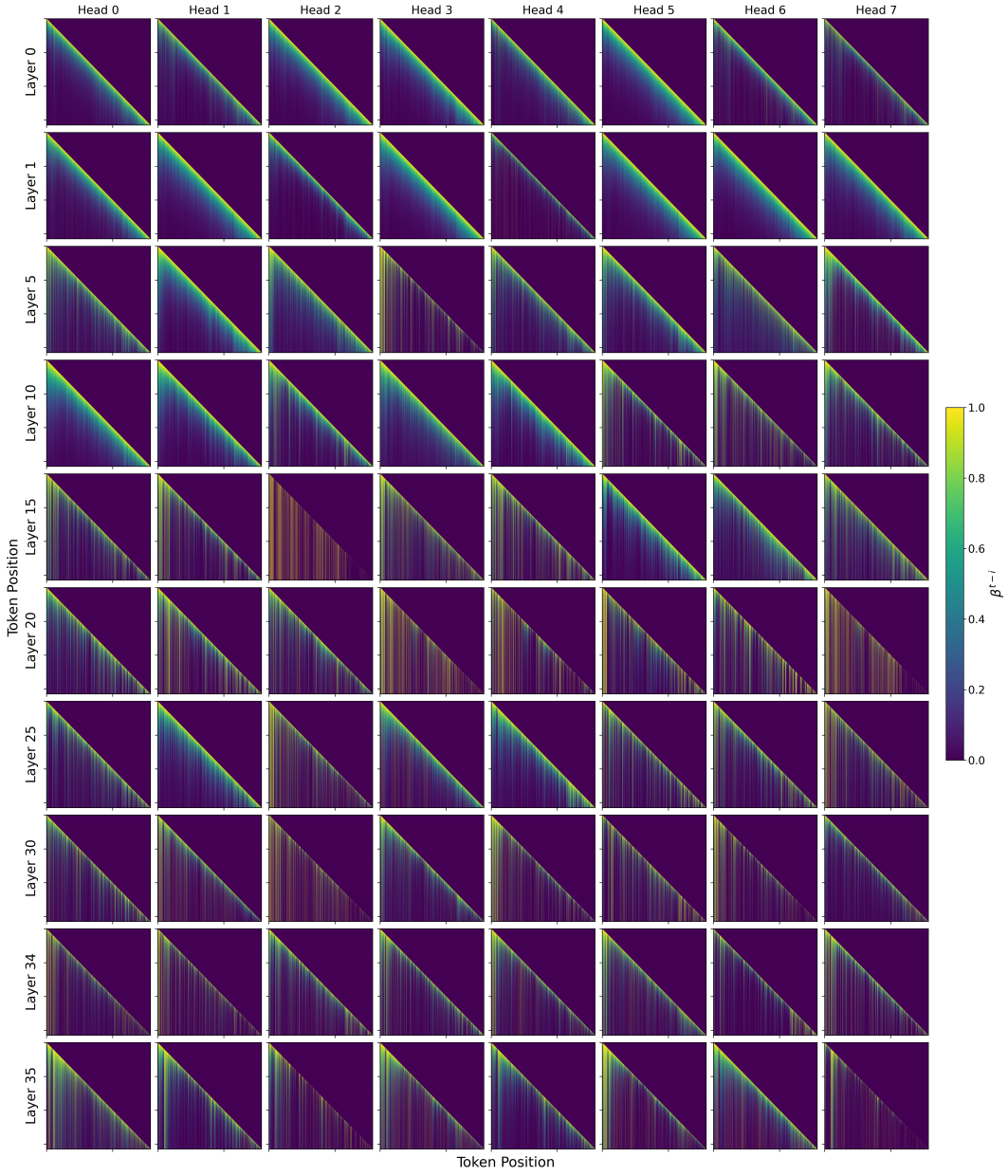

Figure 11: A visualization of token retention matrices of Qwen3-4B when answering a math question in the AIME24 dataset. Each subplot is a token retention matrix $\{\beta_i^{t-i}\}_i^t$ in a specific layer and head. **Observations:** earlier layers often exhibit sliding-window-like patterns, while later layers develop clearer functional specializations.

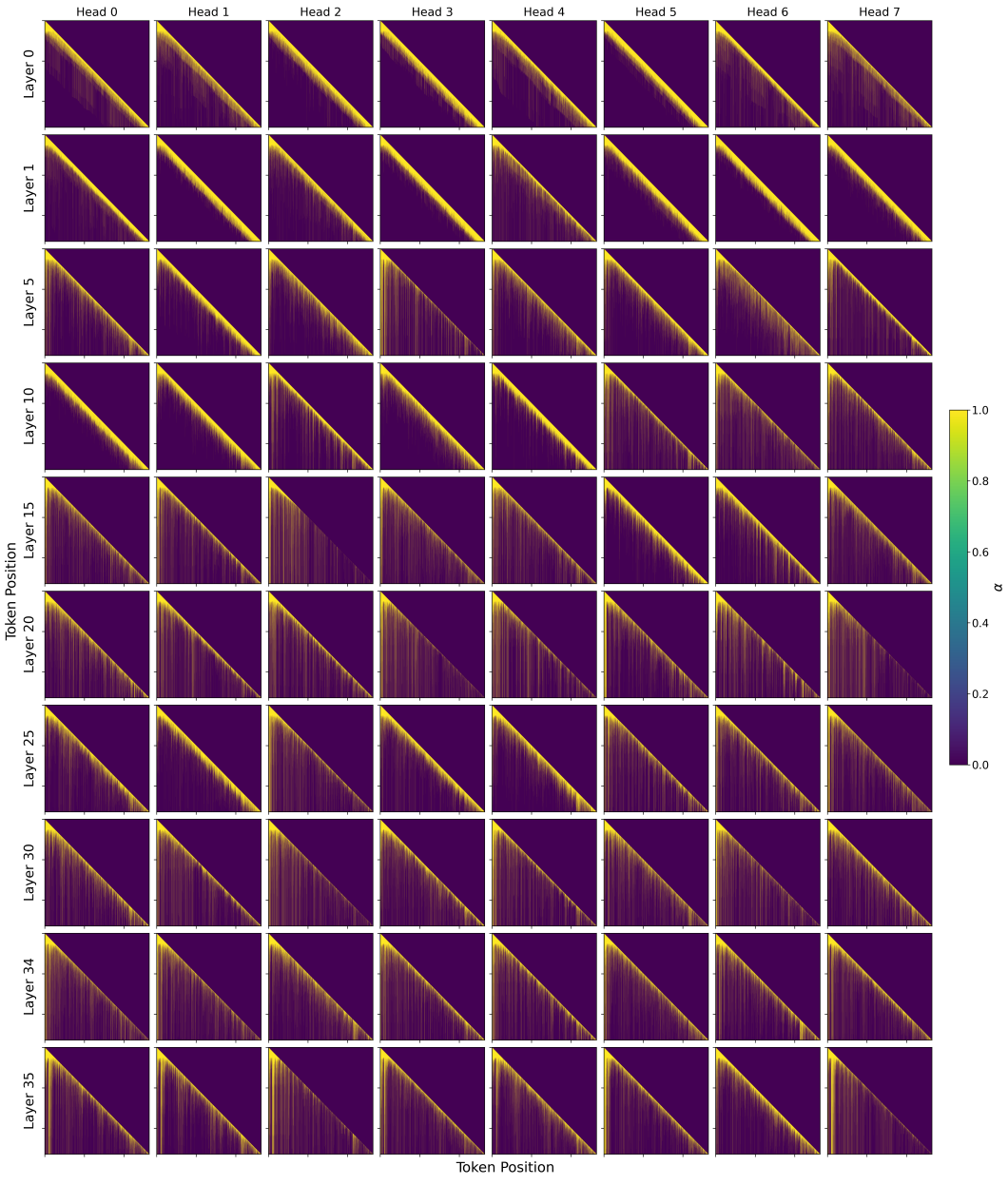

Figure 12: A visualization of eviction decisions of Qwen3-4B when answering a math question in the AIME24 dataset. Each subplot is a matrix of eviction decisions α_{ti} in a specific layer and head.

Figure 13: Visualization of token retention scores β_i from the retention gate in *layer 15, head 2* of the Qwen3-4B model. Brighter colors denote higher retention; gray indicates near-zero retention. **Observation:** This head consistently assigns a high retention to the problem description and tokens expressing mathematical operations, suggesting a specialization for task-critical content. In contrast, filler phrases (e.g., 'wait', 'let me think') receive very low retention. Moreover, for a multi-digit number, retaining only the hidden state of the final digit suffices to maintain performance, suggesting that the last digit's representation already captures the semantic information of the whole number.

Figure 14: Visualization of token retention scores β_i from the retention gate in *layer 9, head 7* of the Qwen3-4B model. Brighter colors denote higher retention; gray indicates near-zero retention. **Observations:** This head exhibits a retention pattern similar to *layer 15, head 2*, but places greater emphasis on arithmetic operators (e.g., $+, -, /$).

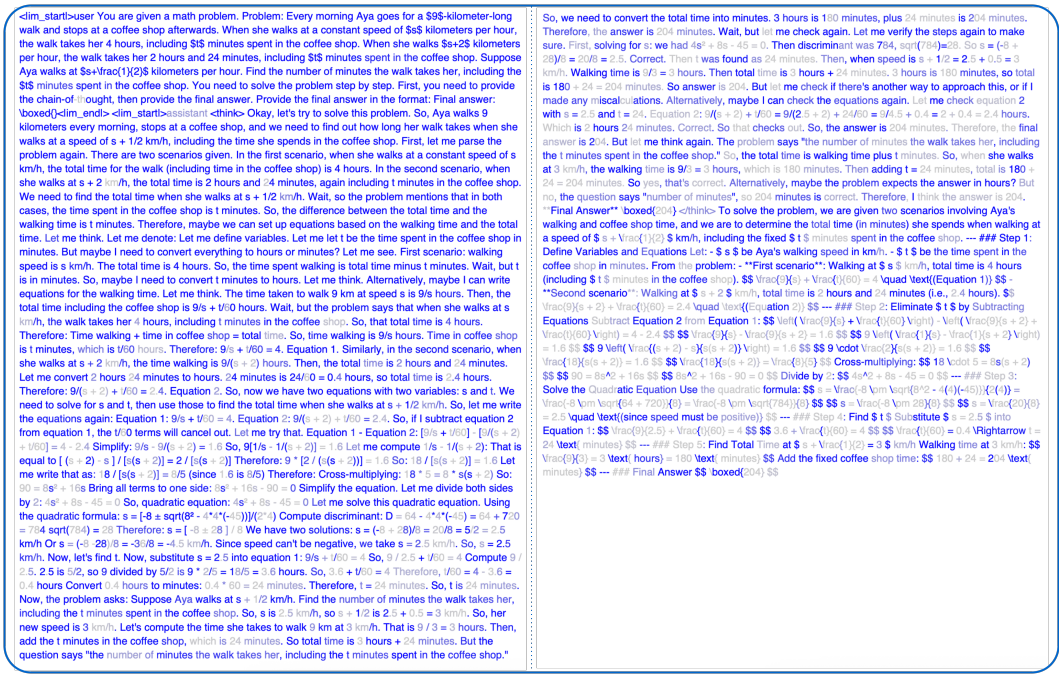

Figure 15: Visualization of token retention scores β_i from the retention gate in *layer 35, head 2* of the Qwen3-4B model. Brighter colors denote higher retention; gray indicates near-zero retention. **Observation:** Unlike *layer 15, head 2* and *layer 9, head 7*, this head assigns elevated retention to general-purpose tokens that support coherence and instruction following; for example, LaTeX commands and the directive *boxed*{} receive high scores, while tokens associated with mathematical operations receive low retention.

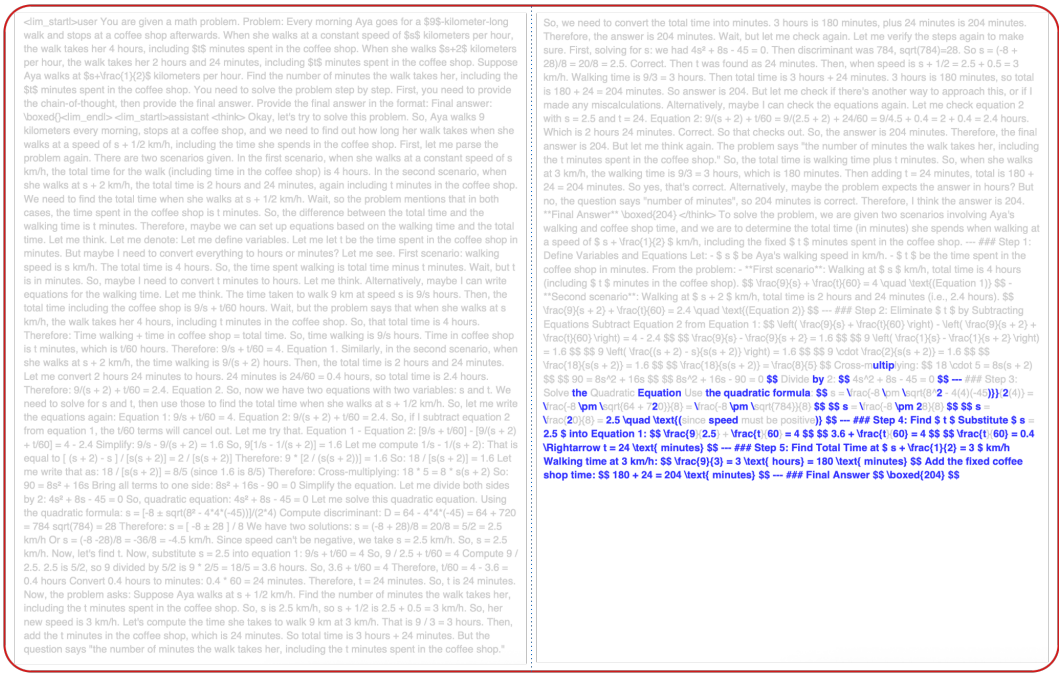

Figure 16: Visualization of tokens retained in the *layer 0 head 3* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 17: Visualization of tokens retained in the *layer 0 head 6* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 18: Visualization of tokens retained in the *layer 9 head 7* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 19: Visualization of tokens retained in the *layer 15 head 2* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 20: Visualization of tokens retained in the *layer 16 head 6* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 21: Visualization of tokens retained in the *layer 30 head 4* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

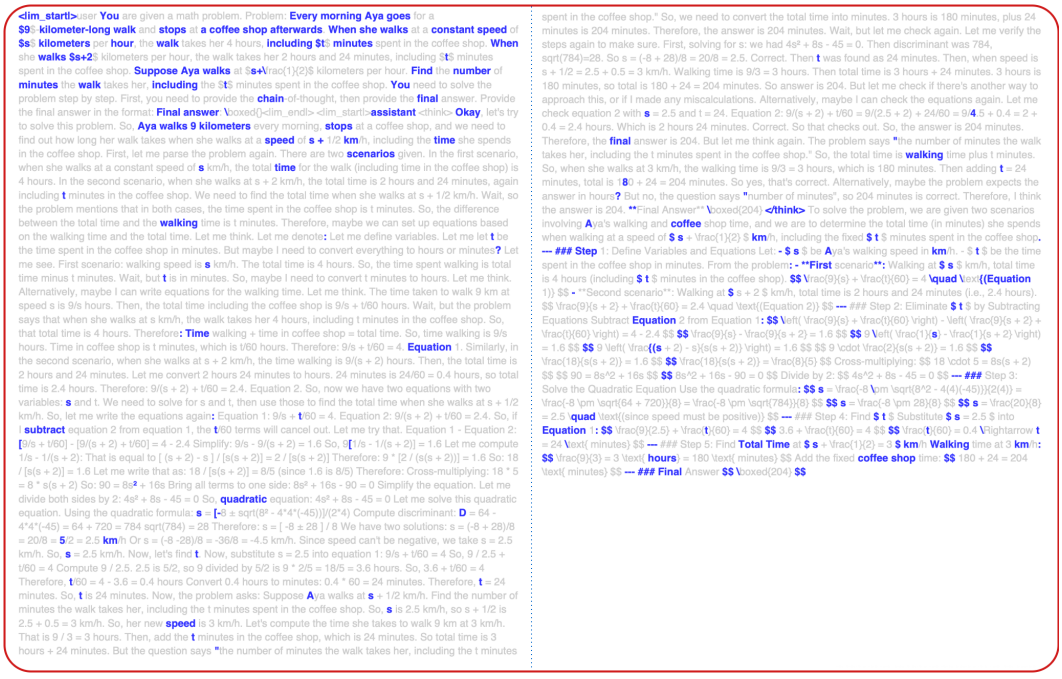

Figure 22: Visualization of tokens retained in the *layer 34 head 1* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 23: Visualization of tokens retained in the *layer 35 head 3* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 24: Visualization of tokens retained in the *layer 35 head 5* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.

Figure 25: Visualization of tokens retained in the *layer 35 head 7* of the KV cache after generation, where the KV budget is 256. The model is Qwen3-4B. Bold blue indicates tokens retained in the KV cache, where gray indicates evicted tokens.