KVPOP — KEY-VALUE CACHE COMPRESSION WITH PREDICTIVE ONLINE PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Key-value (KV) cache growth is a major bottleneck in autoregressive decoding, as memory and bandwidth scale linearly with context length. Existing KV eviction methods often rely on static heuristics or proxy scores, which poorly track future token utility and cause brittle eviction as relevance shifts. To address this, we introduce KVPOP, which learns a fixed-budget KV eviction policy by directly supervising the keep-or-drop decision. We evaluate both stateless and delayed memory-based scorers, showing that boundary supervision succeeds with lightweight MLPs while recurrent scoring leverages near-future context before eviction. On AIME and HMMT mathematical reasoning, KVPOP retains 98% of full-attention performance on QWEN3-4B at 75% KV cache compression and 97% at 88% compression, consistently outperforming established eviction baselines. QWEN3-8B shows even stronger results, reaching near-full teacher performance. These results show that supervising eviction with future-attention signals cuts memory costs while maintaining quality.

1 INTRODUCTION

Transformer-based large language models (LLMs) rely on a key–value (KV) cache to make autoregressive decoding efficient. At each generation step, the model stores the key and value representations of previous tokens, allowing future queries to attend to the past without recomputing the full sequence history (Vaswani et al., 2017). Although this mechanism is essential for practical token-by-token generation, the KV cache grows linearly with context length, becoming a bottleneck for long-context inference (Kwon et al., 2023). A natural way to reduce this cost is to *evict* unimportant tokens and retain only a bounded subset, but token utility is hard to predict. Locally salient tokens may quickly become irrelevant, while tokens that receive little immediate attention may matter many steps later. KV cache reduction is therefore a question of which tokens will be useful for future queries.

KV cache reduction with supervised eviction policies. Existing approaches address this prediction problem in different ways. Sliding-window and sink-token methods preserve a small set of initial tokens together with a recent window exempt from eviction (Xiao et al., 2023; 2024). Score-based methods estimate token importance online from attention-derived or query-local signals (Zhang et al., 2023; Oren et al., 2024; Li et al., 2024). Learned approaches train retention policies during retrofitting. DMC, for example, merges token representations with learned compression policies (Nawrot et al., 2024), while DMS trains binary eviction gates and delays eviction until tokens leave a protected window of recent tokens (Łańcucki et al., 2025). Future-attention supervision has also been explored for prompt pruning (Ahn et al., 2026) and decoding-time eviction (Dong et al., 2026). We study its use for token-by-token admission under a fixed per-head budget, with scoring optionally delayed to incorporate context accumulated within a protected window.

KVPOP. We introduce KVPOP (**KV** compression with **P**redictive **O**nline **P**runing), a sparse-attention retrofit for boundary-supervised online eviction. Each KV head retains a small set of sink tokens, a protected window of recent tokens, and a learned long-range top-k cache for older tokens. The long-range cache is populated by lightweight head-wise scoring modules that assign importance scores to tokens and rank them under the top-k budget. This gives a bounded KV cache for inference without changing the base architecture. Our design combines boundary-focused supervision with flexible scoring time. First, scoring module supervision is anchored to the future-attention mass a token receives after it leaves the protected window. A pairwise loss compares each newly eligible

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

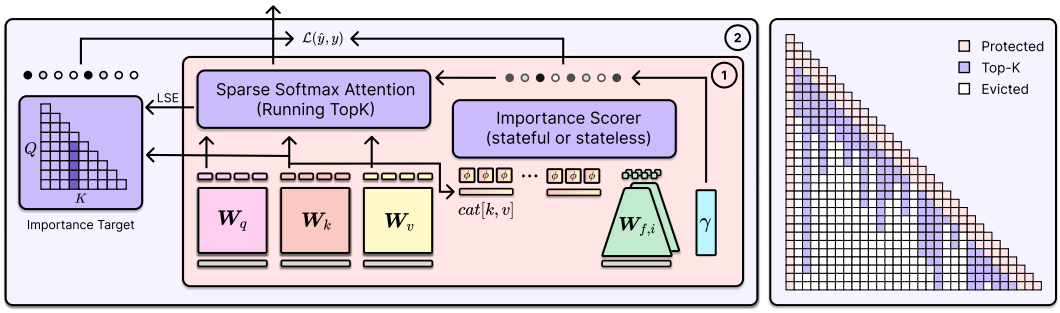


Figure 1: **Overview of KVPOP. (1) Training and inference:** In each attention layer, keys and values are passed to a lightweight per-KV-head scoring module. The cache always retains sink tokens and a protected recent window, and uses predicted scores to select the remaining long-range top-k entries. Attention is computed over the retained set. **(2) Training only:** Future-attention targets are computed on the fly and used to supervise the eviction decision induced by the fixed budget. **Sparse Attention Pattern:** KVPOP yields a sparse attention pattern for efficient decoding, maintaining constant memory per head during inference.

token against the oracle top-k cutoff, and the target is computed during training with a transposed-attention pass that avoids materializing the dense attention map. Second, scoring need not happen when a token enters the cache. KVPOP also supports stateful scorers that delay scoring until the end of the protected window, accumulating evidence while the token remains in the protected window. Compared with DMS-style insertion-time gates, delayed scoring incorporates context observed before a token becomes evictable. An overview of KVPOP is shown in Figure 1.

Contributions. We make the following contributions: **(i)** we formulate fixed-budget KV eviction as supervised prediction at the eviction boundary, using a future-attention target and boundary-aware loss to train long-range top-k decisions; **(ii)** we compute this target with a training-only transposed-attention pass, avoiding dense attention-map materialization and inference-time overhead; **(iii)** we explore eviction timing, evaluating how delayed memory-based scoring leverages near-future context before a token becomes evictable; **(iv)** we show empirically that KVPOP retains >93% of dense-attention accuracy on QWEN3-4B and QWEN3-8B at 75% and 88% KV cache reduction, outperforming the evaluated heuristic and learned eviction baselines on average across the mathematical reasoning benchmarks, and generalizing across domains and model families. In terms of efficiency, its bounded per-head cache guarantees constant decoding memory and yields up to 3× faster decoding on long generation traces.

2 Related Work

Prior work on KV cache reduction can be broadly organized into three categories: *sparse retrieval methods* reduce the number of tokens read per query while keeping the full cache available, *heuristic eviction methods* reduce the persistent cache by removing tokens according to fixed rules or online scores, and *learned eviction methods* train parametric compression policies or importance predictors.

Sparse retrieval over full KV cache. Rather than permanently evicting tokens, sparse retrieval methods select pages, blocks, landmarks, or token groups likely to matter for the current step. Quest retrieves relevant memory pages using lightweight approximations to token importance (Tang et al., 2024), and Landmark Attention introduces special tokens that route attention to relevant blocks (Mohtashami & Jaggi, 2023). Other approaches learn the selection rule itself: Native Sparse Attention combines coarse block selection with fine token selection in a trained sparse-attention pattern (Yuan et al., 2025), DeepSeek Sparse Attention scales this idea to a deployed frontier model with a lightning indexer driving fine-grained token selection (DeepSeek-AI et al., 2025). TokenButler trains a lightweight query-aware predictor of fine-grained token importance (Akhauri et al., 2025). These methods reduce attention work and avoid irreversible deletion, but because the full history remains stored, they do not enforce a bounded KV cache.

Heuristic and prompt eviction methods. A second line of work imposes cache constraints via structural priors or importance heuristics. Prompt-compression methods prune the cache once after prefill, using observation-window attention (SnapKV; Li et al. 2024, PyramidKV; Cai et al. 2025) or context self-reconstruction (KVzip; Kim et al. 2025). However, they do not manage newly generated tokens during decoding, leaving memory unbounded over long generation traces. For online decoding, structural policies retain initial sinks and a sliding window (Xiao et al., 2023; 2024), while dynamic policies evict tokens based on observed attention (Zhang et al., 2023; Oren et al., 2024) or analytic expected future attention (Devoto et al., 2025). Yet, heuristic proxies struggle when early reasoning steps are revisited after delays, and periodic recompression without a local window can prematurely evict essential tokens.

Learned eviction methods. Learned methods train parametric compression policies or importance predictors. DMC (Nawrot et al., 2024) learns online token merging, while DMS (Łańcucki et al., 2025) trains binary eviction gates with differentiable relaxations. KVzap (Jegou & Jeblick, 2026) trains per-layer MLPs from hidden states and applies threshold-based eviction, yielding input-dependent cache sizes rather than a fixed capacity. LookaheadKV (Ahn et al., 2026) learns prompt-token importance from future-response attention for cache pruning during prefill. ForesightKV (Dong et al., 2026) trains periodic decoding-time eviction using future-attention-based pairwise ranking followed by reinforcement learning. Concurrent work on Neural Garbage Collection (NGC) (Li et al., 2026) learns eviction actions end-to-end using reinforcement learning with sparse sequence-level rewards.

KVPOP combines future-attention supervision with token-by-token admission into a strictly bounded per-head cache. Its target measures the attention mass a token receives after exiting the protected window, and its pairwise loss compares each newly eligible token against the oracle top-k cutoff. The target is computed during training without materializing dense attention maps. Scoring can be delayed until the end of the protected window to incorporate accumulated near-future context, after which each token's predicted score is reused for subsequent retention decisions.

3 KVPOP

We propose KVPOP (**KV** compression with **P**redictive **O**nline **P**runing), a sparse-attention retrofit that enforces a fixed per-head KV budget in pretrained models. Each KV head retains s sink tokens and a protected window of w recent tokens, an established structural prior for sparse autoregressive attention (Xiao et al., 2024). The remaining tokens compete for a long-range top-k budget, with attention evaluated solely over the retained set:

$$B = s + w + k \tag{1}$$

The retention policy is guided by lightweight scorers that rank eligible tokens online. In each layer, the scorer receives keys and values with gradients detached from the backbone. During training, scorers regress against future-attention targets and during inference, only budget B is enforced. Figure 1 illustrates the architecture, and Algorithm 1 details distillation, Appendix J provides PyTorch-style pseudocode.

Require:
student model f_θ, dense teacher $f_{\bar{\theta}}$
sequence $x_{0:S-1}$, budget (s, w, k)
$H^0 \leftarrow \mathrm{Embed}(x_{0:S-1})$
$\mathcal{L}_{\text{score}} \leftarrow 0$
for $\ell = 1, \dots, L$ **do**
  $(Q^\ell, K^\ell_{\text{pre}}, K^\ell, V^\ell) \leftarrow \mathrm{Proj}^\ell(H^{\ell-1})$
  $\hat{r}^\ell \leftarrow \mathrm{Scorer}^\ell(K^\ell_{\text{pre}}, K^\ell, V^\ell)$
  $\hat{r}^\ell(q) \leftarrow \mathrm{ApplyDecay}(\hat{r}^\ell, q)$
  $M^\ell \leftarrow \mathrm{TopKMask}(\hat{r}^\ell(q), s, w, k)$
  $(H^\ell, \widetilde{\mathrm{LSE}}^\ell) \leftarrow \mathrm{SparseAttn}(Q^\ell, K^\ell, V^\ell, M^\ell)$
  $r^\ell_{\text{tgt}} \leftarrow \mathrm{FutureTarget}(Q^\ell, K^\ell, \widetilde{\mathrm{LSE}}^\ell, w)$
  $\mathcal{L}_{\text{score}} \leftarrow \mathcal{L}_{\text{score}} + \mathrm{BndLoss}(\hat{r}^\ell(q), r^\ell_{\text{tgt}}, s, w, k)$
end for
$\mathcal{L}_{\text{score}} \leftarrow \mathcal{L}_{\text{score}} / L$
$\mathcal{L} \leftarrow \mathrm{KL}\big(f_{\bar{\theta}}(x_{0:S-1}) \,\big\|\, \mathrm{LMHead}(H^L)\big) + \mathcal{L}_{\text{score}}$
Update θ using $\nabla_\theta \mathcal{L}$
return $\mathcal{L}$

Algorithm 1: **KVPOP distillation.** A scorer predicts importance scores used to construct a fixed-budget sparse attention mask (K^ℓ_{pre}: pre-RoPE keys). Future attention targets supervise eviction decisions during training with zero inference overhead.

3.1 IMPORTANCE SCORER SUPERVISION

To have the scoring target reflect future utility, we define supervision in terms of the future attention mass a key receives after it has left the protected window. The target is independent of the scorer architecture and can supervise both stateless and stateful policies (see Section 3.3).

Future-attention target. Let S be the training sequence length and let $g \in \{1, \dots, G\}$ index the query heads that share KV head h under grouped-query attention (Ainslie et al., 2023). Let $p_{d\to t}^{(h,g)}$ denote the dense causal attention probability from query head (h, g) at position d to key (h, t). Since token t is protected until queries are at least w positions ahead, we define its mean future-attention mass per group as

$$m_t^{(h,g)} = \frac{1}{N_t} \sum_{d=t+w}^{S-1} p_{d\to t}^{(h,g)} \qquad \text{where } N_t = \max(1, S - (t + w)) \tag{2}$$

Since a KV entry is shared by all G query heads, we aggregate the per-group log-masses to obtain the future-utility target.

$$r_{h,t}^{\text{tgt}} = \text{Agg}_g \left[\log \left(m_t^{(h,g)} + \epsilon \right) \right] \tag{3}$$

where ϵ ensures numerical stability when tokens receive negligible attention. In the experiments, Agg_g is max aggregation. Alternatives are discussed in Appendix B.

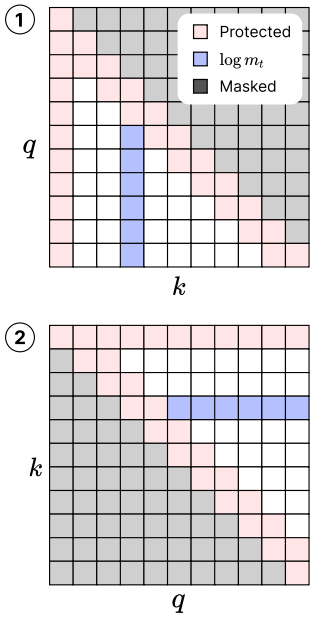


Figure 2: **Transposed Attention.** **(1)** Blue tiles show future attention mass for token t after leaving the protected window w. **(2)** Transposing queries q and keys k converts per-key column-sums into per-query row-sums, which fused attention kernels return as auxiliary LSE during training.

Effective scores and teacher policy. At query position q, only tokens outside the protected sink region and recent window compete for the long-range budget. We protect sinks unconditionally to prevent temporal decay from evicting vital prompt anchors:

$$\mathcal{E}(q) = \{t \mid s \le t \le q - w\} \tag{4}$$

The token $t_{\text{new}} = q - w$ has just left the protected window and becomes newly eligible. The teacher ranks eligible tokens by target effective scores

$$r_{h,t}^{\text{tgt}}(q) = r_{h,t}^{\text{tgt}} + (q - t) \log \gamma_h \tag{5}$$

where $\gamma_h \in (0, 1)$ is a learned per-head decay factor, corresponding to a unit decay step ($n = 1$; see Appendix B.3). The decay term introduces a recency bias that prevents tokens with very high scores from occupying cache slots indefinitely, allowing newer tokens to compete for retention over time. The teacher and student share the same γ_h, ensuring supervision matches inference ranking. When $|\mathcal{E}(q)| \le k$ (i.e., $q < s + w + k$), all eligible tokens are retained. Once the cache is full ($q \ge s + w + k$), the candidate t_{new} competes against the incumbent cutoff t_{bnd}, defined as the k-th ranked token in $\mathcal{E}(q) \setminus \{t_{\text{new}}\}$. Admitting t_{new} into the oracle top-k yields the binary label:

$$y_{q,h} = \begin{cases} +1, & \text{if } r_{h,t_{\text{new}}}^{\text{tgt}}(q) \ge r_{h,t_{\text{bnd}}}^{\text{tgt}}(q) \\ -1, & \text{otherwise} \end{cases} \tag{6}$$

Boundary-aware retention loss. Let $\hat{r}_{h,t}(q) = \hat{r}_{h,t} + (q - t) \log \gamma_h$ be the predicted effective score. At a fixed query q, the query term $q \log \gamma_h$ cancels and comparisons depend on static priorities $\bar{r}_{h,t} = \hat{r}_{h,t} - t \log \gamma_h$, with difference

$$\hat{r}_{h,a}(q) - \hat{r}_{h,b}(q) = (\hat{r}_{h,a} - \hat{r}_{h,b}) + (b - a) \log \gamma_h \tag{7}$$

The position offset $(b - a) \log \gamma_h$ provides gradient to γ_h. We train the scorer with a pairwise logistic loss at the retention boundary:

$$\mathcal{L}_{\text{score}} = \mathbb{E}_{q,h} \left[\omega_{q,h} \, \text{softplus} \left(-y_{q,h} \frac{\hat{r}_{h,t_{\text{new}}}(q) - \hat{r}_{h,t_{\text{bnd}}}(q)}{\tau} \right) \right] \tag{8}$$

where τ is a temperature and $\omega_{q,h}$ is an optional weighting term. Setting $\omega_{q,h} = 1$ gives the unweighted objective. In our implementation, we use $\omega_{q,h}$ to downweight ambiguous teacher decisions and balance keep/drop decisions across heads. Details are given in Appendix B. The boundary loss focuses capacity on the single comparison that changes cache membership and costs $O(1)$ per query position once the cutoff is known; cutoffs are updated online across all queries via a Fenwick tree, while loss is evaluated only at sampled saturated positions.

3.2 Efficient Target Implementation

The target in Eq. (3) is defined through dense causal attention probabilities, but we never materialize the corresponding $S \times S$ probability matrices. The student forward pass already computes a fixed-budget sparse attention pattern and returns per-query log-normalizers from the sparse attention kernel. We reuse these sparse log-normalizers as an approximation to the dense causal normalizers, and compute the future-attention target with one additional transposed-attention pass implemented with efficient attention kernels such as FlexAttention (Dong et al., 2025).

Transposed-attention target computation. Let

$$\ell^{(h,g)}(d,t) = \langle \boldsymbol{q}_d^{(h,g)}, \boldsymbol{k}_t^{(h)} \rangle / \sqrt{d_k} \tag{9}$$

be the pre-softmax attention logit, and let $\mathrm{LSE}_d^{(h,g)}$ be the causal log-sum-exp (LSE) normalizer for query position d. The mean future-attention mass $m_t^{(h,g)}$ from Eq. (2) can equivalently be written as

$$\log m_t^{(h,g)} = \log \sum_{d=t+w}^{S-1} \exp\left(\ell^{(h,g)}(d,t) - \mathrm{LSE}_d^{(h,g)}\right) - \log N_t \tag{10}$$

The first term is a log-sum-exp over future query positions for a fixed key t, corresponding to a column-wise reduction over the dense logit matrix. We evaluate it by swapping the roles of queries and keys in a second attention-like call (Figure 2). Original key positions t become queries, original query positions d become keys, and the dot products recover the original logits $\ell^{(h,g)}(d,t)$. We subtract the per-query normalizer as a score modifier and apply a block mask enforcing $d \geq t + w$. The auxiliary LSE returned by the attention kernel then gives the first term of Eq. (10) for all keys in parallel, after which we subtract $\log N_t$ to obtain $\log m_t^{(h,g)}$. Adding ϵ inside the log and aggregating across query heads sharing KV head h yields $r_{h,t}^{\mathrm{tgt}}$ as in Eq. (3). Using the dense causal $\mathrm{LSE}_d^{(h,g)}$ would make this identity exact. Instead, we reuse the sparse query LSE values

$$\widetilde{\mathrm{LSE}}_d^{(h,g)} = \log \sum_{t'=0}^{d} M_{d,t'} \exp\left(\ell^{(h,g)}(d,t')\right) \tag{11}$$

already returned by the student attention pass, where M is the fixed-budget sparse mask. The transposed pass therefore only requires a masked LSE through the same kernel interface, adding no inference-time overhead. We find empirically that this sparse-LSE approximation matches the dense-LSE target in downstream performance (see ablation in Appendix D.3), while avoiding the cost of a separate dense-attention pass during training. The per-group targets are aggregated across query heads sharing KV head h to produce $r_{h,t}^{\mathrm{tgt}}$ as in Eq. (3). Appendix D provides the full derivation.

Running top-k sparse attention. At each query q, sparse attention retains the union of sink tokens, the recent window, and the top-k tokens from $\mathcal{E}(q)$ under effective scores $\hat{r}_{h,t}(q) = \bar{r}_{h,t} + q \log \gamma_h$. Because $q \log \gamma_h$ is shared across eligible tokens, ranking is invariant to q and reduces to static priorities $\bar{r}_{h,t} = \hat{r}_{h,t} - t \log \gamma_h$. Crucially, once a token exits the prefix top-k, arriving candidates cannot restore it, allowing parallel training masks to represent irreversible online eviction.

Concretely, we sort tokens once by their static priority and compute a rank for each token. As q increases, exactly one new token, $t = q - w$, enters the eligible set. We insert its rank into an online Fenwick tree (Fenwick, 1994) and retrieve the rank of the k-th best eligible token by binary lifting, implemented as a custom Triton kernel parallelized over batch elements and KV heads. This gives a query-specific cutoff rank τ_q, so the retained long-range keys are exactly those eligible tokens with rank at most τ_q.

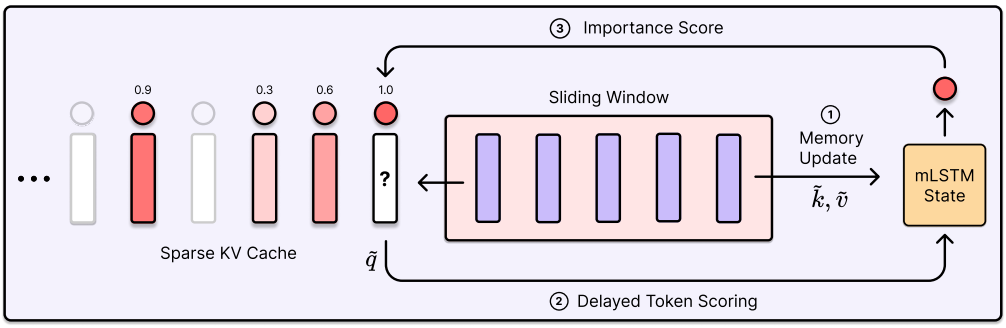


Figure 3: **KVPOP stateful eviction policy. (1)** At each position, the MLSTM memory is updated with the newest KV pair. **(2)** Pipelined delayed scoring reads the updated state with the token on its last step inside the protected window; its score is cached. **(3)** When that token exits the window at the next position, its cached score is ranked against the weakest incumbent in the long-range cache to decide admission.

The resulting predicate is evaluated inside FLEXATTENTION, which generates the sparse mask on the fly inside the fused attention kernel rather than storing a dense $S \times S$ mask. The cutoff computation costs $O(S \log S)$ time and $O(S)$ space per head, while the attention computation itself is performed by the sparse kernel. Appendix J contains the full algorithm.

3.3 ONLINE IMPORTANCE SCORERS

The target and loss above can supervise any module that assigns scalar importance scores to cached tokens. In each layer, the scorer receives a concatenation of keys and values for each KV head. Crucially, the scorer inputs are detached from the computation graph, ensuring there is no gradient flow from that point back to the rest of the network: the retention loss in Eq. (8) updates only the scorer parameters, while the base model is trained solely by the distillation objective.

Stateless scorers. The simplest scorer predicts token importance from the token's own representation. We concatenate key and value at position t as $\boldsymbol{x}_{h,t} = [\boldsymbol{k}_{h,t}; \boldsymbol{v}_{h,t}]$ and compute $\hat{r}_{h,t} = f_\theta^{(h)}(\boldsymbol{x}_{h,t})$, where $f_\theta^{(h)}$ is a small headwise scorer. In our experiments we use a two-layer MLP with a SiLU activation. Stateless scorers are cheap, but they score each token using only local information.

Stateful scorers. Stateless scorers assign an importance score from the token's representation at insertion time. The token's own representation, however, may not capture how it relates to earlier context, and it cannot incorporate context that accumulates afterward. A stateful scorer instead maintains a memory shaped by the retention objective, and the protected window enables *delayed* scoring. A token does not need a score when it enters the KV cache, only when it leaves the window and begins competing for the long-range budget. In our implementation, token t is scored after position $t + w - 1$ (scoring delay $d = w - 1$) on its final step inside the protected window, and its score is cached. At position q, admission uses this precomputed score for $t_{\text{new}} = q - w$, while the updated memory produces the score for $q - w + 1$ which becomes eligible next (Figure 3). This is distinct from delayed eviction in DMS (Łańcucki et al., 2025), which postpones eviction without accumulating context. We use an MLSTM (Beck et al., 2024) scorer per KV head. Let $\boldsymbol{C}_{h,q-1}$ and $\boldsymbol{z}_{h,q-1}$ denote the MLSTM state and normalizer after position $q - 1$ (pre-RoPE writes), and let $\tilde{\boldsymbol{q}}_{h,t_{\text{new}}}$ be projected features of t_{new}'s post-RoPE key and value. The pipelined delayed readout is

$$\boldsymbol{h}_{h,t_{\text{new}}} = \frac{\boldsymbol{C}_{h,q-1}^\top \tilde{\boldsymbol{q}}_{h,t_{\text{new}}}}{\tilde{\boldsymbol{q}}_{h,t_{\text{new}}}^\top \boldsymbol{z}_{h,q-1}} \tag{12}$$

and the raw importance score is produced by a small headwise projection

$$\hat{r}_{h,t_{\text{new}}} = \boldsymbol{a}_h^\top \operatorname{SiLU}(\boldsymbol{h}_{h,t_{\text{new}}}) + b_h \tag{13}$$

Appendix E gives the full MLSTM formulation, feature maps, gates, initialization, and implementation variants. Appendix A provides background on MLSTM and Linear Attention.

Table 1: **Pass@1** on **AIME** and **HMMT** shown as absolute scores (Abs) and w.r.t. teachers (Rel). KVPOP denotes the stateful $\text{KVPOP}_{\text{MLSTM}}$ variant.

	Variant	Qwen3 4B						Qwen3 8B					
		AIME		HMMT		Average		AIME		HMMT		Average	
		2024	2025	2502	2511	Abs.	Rel.	2024	2025	2502	2511	Abs.	Rel.
	Teacher	.61	.46	.30	.43	.45	1.00	.58	.49	.28	.37	.43	1.00
CR = 75%	StreamLLM	.47	.33	.24	.33	.34	.76	.13	.11	.03	.05	.08	.19
	TOVA	.56	.38	.30	.38	.41	.90	.37	.23	.15	.28	.26	.60
	Exp. Attention	.16	.15	.10	.06	.12	.27	.30	.22	.11	.19	.21	.49
	KVzap	.58	.37	.28	.38	.40	.89	.56	.37	.25	.30	.37	.86
	StreamLLM+	.55	**.44**	.30	.36	.41	.92	.53	.41	.25	.35	.39	.90
	DMS	**.62**	**.44**	.28	.38	.43	.96	.56	.45	.27	.35	.41	.95
	$\text{KVPOP}_{\text{mlp}}$	**.62**	.43	.29	**.40**	**.44**	.97	**.59**	.46	.30	**.38**	.43	1.00
	KVPOP	**.62**	**.44**	**.31**	.39	**.44**	**.98**	.57	**.48**	**.31**	**.38**	**.44**	**1.01**
CR = 88%	StreamLLM	.30	.23	.15	.17	.21	.47	.13	.11	.03	.05	.08	.19
	TOVA	.36	.23	.19	.28	.27	.59	.08	.07	.09	.08	.08	.19
	Exp. Attention	.01	.03	.00	.00	.01	.02	.03	.04	.04	.00	.03	.07
	KVzap	.31	.19	.10	.12	.18	.40	.24	.19	.09	.11	.16	.37
	StreamLLM+	.45	.33	.26	.29	.33	.74	.42	.30	.18	.24	.29	.66
	DMS	.58	.41	.27	.35	.40	.89	.52	.38	.22	.31	.36	.83
	$\text{KVPOP}_{\text{mlp}}$	.59	**.44**	.27	.38	.42	.93	.57	.39	**.31**	**.40**	.42	.97
	KVPOP	**.61**	**.44**	**.30**	**.39**	**.44**	**.97**	**.58**	**.44**	**.31**	.39	**.43**	**1.00**

4 Experiments

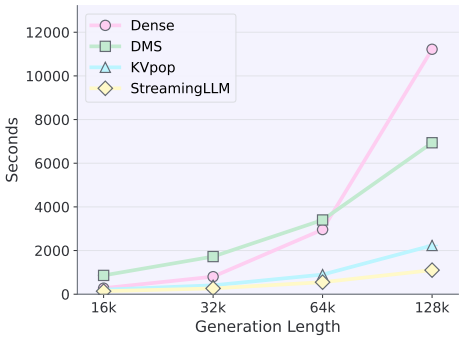
We apply KVPOP distillation to Qwen3-4B-Instruct-2507 and Qwen3-8B (Yang et al., 2025). We use a training sequence length of $S = 16384$. For sparsification, we use the Nemotron-Math v2 dataset (Du et al., 2025), selecting the subset with high reasoning effort and filtering for sequences of length at most S under the Qwen tokenizer. We apply sequence packing to improve token utilization.

We train both a stateless variant, $\text{KVPOP}_{\text{mlp}}$, and a stateful variant using the mLSTM scorer, $\text{KVPOP}_{\text{MLSTM}}$ (referred to simply as KVPOP in tables and figures), for 2,000 steps. Since recurrent state introduces additional memory overhead for stateful scorers, we reduce the top-k budget of the stateful variants to match the memory footprint of the stateless scorers. We use a cosine schedule for the scorer parameters with peak learning rate 10^{-3}, keep the base-model learning rate fixed at 8×10^{-5}, and include a KL-divergence loss. Full hyperparameters are provided in Appendix F.

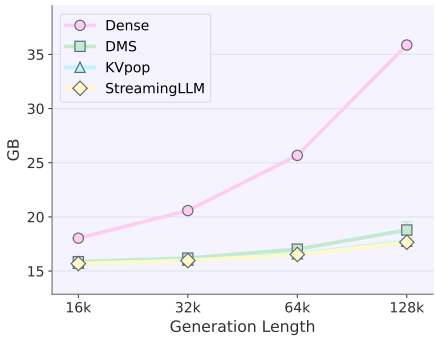
Baselines. We compare against training-free and trained KV cache baselines and full-attention teachers (sparse-retrieval methods are excluded as they retain the full cache; Qwen3-8B enables thinking mode). As training-free methods, we evaluate StreamingLLM (Xiao et al., 2023) (attention sinks and sliding window) and TOVA (Oren et al., 2024) (eviction by attention scores). To isolate learned scoring from training under a fixed sparsity pattern, we train StreamingLLM+ with the same budget and setup as KVPOP. For learned baselines, we evaluate DMS (Łańcucki et al., 2025) (eviction via differentiable relaxations), matching its budget and parameter count to KVPOP. Finally, we compare against Expected Attention (Devoto et al., 2025) and KVzap (Jegou & Jeblick, 2026) via `kvpress`. Expected Attention uses the authors' periodic recompression without a local window. KVzap scores tokens with per-layer MLPs and evicts below an adaptive threshold τ, which we sweep to match realized compression ratios.

4.1 Results

Mathematical Reasoning. We report **pass@1** on AIME24/25 and HMMT (Feb/Nov 2025) in Table 1 under two cache budgets $B \in \{2048, 4096\}$, corresponding to compression ratios (CR) of 88% and 75%. Pass@1 is estimated using 16 rollouts per prompt. We report absolute and relative scores w.r.t. teachers.



(a) End-to-end latency.



(b) Peak allocated VRAM.

Figure 4: **Inference efficiency on QWEN3-8B.** Decoding latency and peak allocated VRAM at batch size 1 up to 128k tokens (88% KV compression ratio). Sparse methods maintain a drastically reduced memory footprint ($\leq$ 19 GB vs. 36 GB for dense at 128k). Stateful KVPOP approaches static StreamingLLM speed and outperforms DMS at long sequences.

At CR=88%, training-free StreamingLLM and TOVA retain decent performance on QWEN3-4B, but all training-free methods collapse on QWEN3-8B ($\leq$ 19%). Fine-tuned StreamingLLM+ recovers partially (66-74%). Among learned baselines, threshold-based KVzap struggles (37-40%), while DMS recovers 83-89% of teacher performance. KVPOP performs best, retaining 97% (4B) and 100% (8B) of dense accuracy and yielding 8-17 points higher relative recovery than DMS. Both scorers perform strongly: stateless $\text{KVPOP}_{\text{mlp}}$ already outperforms DMS on both models, while stateful scoring provides modest additional gains.

Table 2: **SMOLLM3-3B on AIME and HMMT.** Absolute and relative pass@1 w.r.t. teachers across cache compression ratios.

CR	Variant	SMOLLM3-3B AIME 2024	AIME 2025	HMMT 2502	HMMT 2511	Average Abs.	Average Rel.
Teacher		.48	.33	.19	.23	.31	1.00
75%	DMS	.33	.26	.16	.19	.24	.76
	KVPOP	.33	**.32**	**.19**	.19	**.26**	**.84**
88%	DMS	.28	.23	.14	.16	.20	.66
	KVPOP	**.38**	**.30**	**.17**	.16	**.25**	**.82**

At CR=75%, all methods improve with consistent ranking. KVPOP outperforms all sparse baselines on average, matching dense accuracy on QWEN3-8B. Paired tests across all 120 problems confirm gains over DMS are statistically significant at 88% compression for both models and at 75% for QWEN3-8B (see Appendix F.2).

Sparsification generalization. To verify that the learned eviction policy is not overfitted to mathematical reasoning or to the Qwen architecture, we evaluate generalization along two dimensions. First, across out-of-domain tasks, we evaluate the sparsified QWEN3-4B models on GPQA Diamond (GPQA-D) (Rein et al., 2024) and LiveCodeBench v6 (LCB) (Jain et al., 2025), following the evaluation protocol of DMS (Łańcucki et al., 2025). As shown in Table 3, KVPOP retains near-teacher accuracy on both benchmarks despite being trained solely on mathematical reasoning data. Second, across model families, we evaluate KVPOP on SMOLLM3-3B (Allal et al., 2025) on AIME and HMMT under matched cache budgets (Table 2). KVPOP retains 8 points more teacher performance than DMS at 75% compression and 16 points at 88%, confirming that boundary-supervised eviction generalizes to distinct model families.

Table 3: **QWEN3-4B on GPQA-D and LCB.** Out-of-domain evaluation on science and coding benchmarks.

	Variant	GPQA-D	LCB
	Teacher	.59	.35
CR = 75%	StreamLLM	.54	.36
	TOVA	.58	.34
	StreamLLM+	.55	.32
	DMS	.55	**.37**
	$\text{KVPOP}_{\text{mlp}}$	**.59**	.35
	KVPOP	.57	.33
CR = 88%	StreamLLM	.49	**.35**
	TOVA	.54	**.35**
	StreamLLM+	.52	.29
	DMS	.54	**.35**
	$\text{KVPOP}_{\text{mlp}}$	**.57**	**.35**
	KVPOP	.56	.34

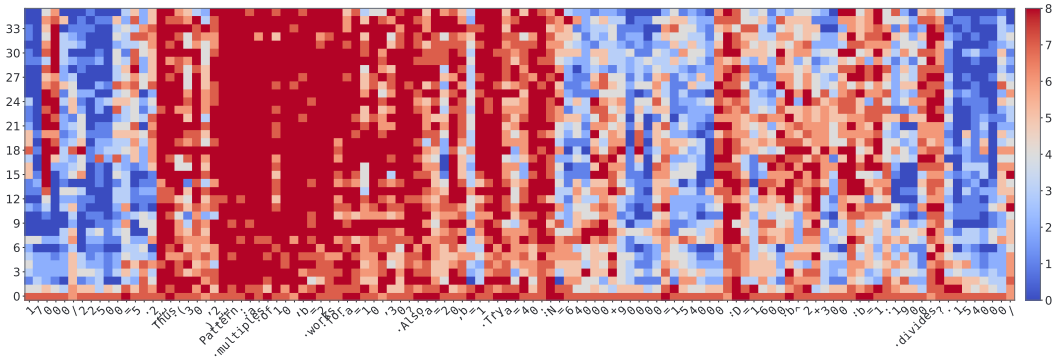


Figure 5: **Eviction patterns in a mathematical reasoning trace.** Last 112 evictable tokens of a sampled sequence. Rows are layers, columns are tokens, and color indicates head retention count. KVPOP drops numeric values earlier while preserving structural reasoning tokens.

Long-context retrieval. We evaluate single-needle retrieval on RULER (NIAH1-3, Appendix F.3), scaling cache budgets to maintain the 75% and 88% compression ratios across 4k and 16k contexts. KVPOP achieves 1.00 accuracy at 4k and 0.80-0.95 at 16k context. DMS achieves 1.00 across all settings. We find its dynamic allocation collapses certain heads to dense attention (Figure 8), which is well suited for single-needle retrieval, whereas KVPOP enforces a uniform per-head budget.

Inference efficiency. We benchmark end-to-end autoregressive decoding on a single NVIDIA H100 GPU with PyTorch, FlashAttention, and `torch.compile` at batch size 1 from a 1,024-token prompt across generation lengths up to 128k tokens. For each generation length, cache budgets are scaled proportionally to maintain a constant 88% compression ratio ($B = L/8$ tokens per head, matching the 8× compression target of DMS; see Appendix F.4 for full configuration and memory breakdown). Figure 4 reports peak allocated VRAM and latency averaged over five steady-state runs (excluding warmup). For baselines, we use official Hugging Face implementations (Appendix F.4). Dense attention exhibits the expected linear KV growth, with peak VRAM rising from 18 GB at 16k tokens to 36 GB at 128k tokens. In contrast, sparse methods scale at an 8× reduced rate, growing only to ≈ 19 GB at 128k tokens. In decoding latency, StreamingLLM is fastest due to its static attention pattern without scorer overhead. Stateful KVPOP remains very close in speed despite dynamic recurrent scoring while DMS incurs a larger overhead. A likely reason is that KVPOP enforces the same fixed budget on *every* KV head, whereas DMS uses dynamic gates that concentrate capacity in a few heads, yielding ragged per-head caches that compile less efficiently (Figure 8).

Eviction policy analysis. Figure 5 visualizes token retention over the final 112 evictable tokens of a sampled reasoning trace. KVPOP does not retain tokens uniformly: numerical intermediate values are evicted more frequently, whereas tokens that structure the derivation—such as discourse markers (*Thus*), operation words (*multiplies*), and symbols (=) are preserved across heads and layers. Early layers retain nearly all tokens in the 112-token crop (Tenney et al., 2019), consistent with content-dependent retention that preserves structural anchors over simple recency (Appendix G).

5 Conclusion

We introduced KVPOP, a fixed-budget KV cache compression method that learns predictive online decisions from future-attention supervision. Designed as an efficient post-training retrofit for Transformers with dense attention rather than a compressed-cache architecture trained from scratch, KVPOP trains scorers at the eviction boundary and optionally delays stateful scoring until near-future context is available (Appendix I). By directly targeting the token-retention decision that determines cache membership, KVPOP bounds inference memory while improving the quality-efficiency trade-off over prior eviction methods. Across AIME and HMMT, KVPOP retains 98% of dense performance on QWEN3-4B at 75% KV compression (97% at 88% compression), and ≈100% on QWEN3-8B at both ratios. The learned policy also transfers to code generation, STEM reasoning, and retrieval benchmarks, showing that predictive learned eviction preserves long-context reasoning under aggressive compression.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods and editing research code. Additionally, we used generative AI tools for drafting paragraphs in the appendix, as well as for editing and refining the manuscript to improve language, clarity, and readability. All AI-assisted code was inspected, debugged, and verified for correctness through empirical execution by the authors, and all AI-generated drafts and textual revisions were critically reviewed, verified, and refined. The authors take full responsibility for the final content of this work, including all text, code, claims, and experimental results.

REPRODUCIBILITY STATEMENT

We describe KVPOP in Section 3 and provide target and loss definitions, sparse-attention algorithms, and scorer architecture details in Appendices B–E, with PyTorch-style pseudocode in Appendix J. Section 4 describes the models, training data, preprocessing, and baselines. Appendix F and Table 6 document training hyperparameters, compute resources, numerical precision, and decoding settings. Cache-budget accounting, statistical testing, retrieval evaluation, and efficiency benchmarking are detailed in Appendices F.1–F.4. Appendix C.1 describes unit tests verifying agreement between parallel sparse-attention masks and step-by-step online eviction.

REFERENCES

Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. LookaheadKV: Fast and accurate KV cache eviction by glimpsing into the future without generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RVLMGPXt2i.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Yash Akhauri, Ahmed F. AbouElhamayed, Yifei Gao, Chi-Chih Chang, Nilesh Jain, and Mohamed S. Abdelfattah. TokenButler: Token Importance is Predictable, March 2025. arXiv:2503.07518 [cs].

Benedikt Alkin, Maximilian Beck, Korbinian Pöppel, Sepp Hochreiter, and Johannes Brandstetter. Vision-lstm: xlstm as generic vision backbone. *arXiv preprint arXiv:2406.04303*, 2024.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. SmolLM3: Open, efficient, and capable small language models, 2025. URL https://huggingface.co/blog/smollm3.

Andreas Auer, Patrick Podest, Daniel Klotz, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. Tirex: Zero-shot forecasting across long and short horizons with enhanced in-context learning. *arXiv preprint arXiv:2505.23719*, 2025.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended Long Short-Term Memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, December 2024. doi: 10.52202/079017-3417.

Maximilian Beck, Korbinian Pöppel, Phillip Lippe, Richard Kurle, Patrick M Blies, Günter Klambauer, Sebastian Böck, and Sepp Hochreiter. xlstm 7b: A recurrent llm for fast and efficient inference. *arXiv preprint arXiv:2503.13427*, 2025.

Ricardo Buitrago and Albert Gu. Understanding and improving length generalization in recurrent models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2OEb20dy7B.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution, 2025. URL https://arxiv.org/abs/2510.00636.

Juechu Dong, BOYUAN FENG, Driss Guessous, Yanbo Liang, and Horace He. Flexattention: A programming model for generating fused attention variants. In *Eighth Conference on Machine Learning and Systems*, 2025.

Zican Dong, Peiyu Liu, Junyi Li, Zhipeng Chen, Han Peng, Shuo Wang, and Xin Zhao. ForesightKV: Optimizing KV cache eviction for reasoning models by learning long-term contribution. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=znV8JHv8b8.

Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision. *arXiv preprint arXiv:2512.15489*, 2025.

Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software—Practice and Experience*, 24(3):327–336, 1994. doi: 10.1002/SPE.4380240306.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Conference on Language Modeling*, volume 1, Philadelphia, PA, USA, 2024. OpenReview.

Lukas Hauzenberger, Niklas Schmidinger, Thomas Schmied, Anamaria-Roberta Hartl, David Stap, Pieter-Jan Hoedt, Maximilian Beck, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. Effective distillation to hybrid xlstm architectures. *arXiv preprint arXiv:2603.15590*, 2026.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dmytro Okhonko, Boris Ginsburg, Igor Gitman, and Mohammad Shoeybi. RULER: What's the real context size of your long-context language models? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024. URL https://arxiv.org/abs/2404.06654.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Simon Jegou and Maximilian Jeblick. Kvzap: Fast, adaptive, and faithful kv cache pruning, 2026. URL https://arxiv.org/abs/2601.07891.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 5156–5165, virtual, 2020. PMLR. arXiv: 2006.16236.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction, 2025. URL https://arxiv.org/abs/2505.23416.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 979-8-4007-0229-7. doi: 10.1145/3600006.3613165.

Michael Y. Li, Jubayer Ibn Hamid, Emily B. Fox, and Noah D. Goodman. Neural garbage collection: Learning to forget while learning to reason, 2026. URL https://arxiv.org/abs/2604.18002.

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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=poE54GOq2l.

Amirkeivan Mohtashami and Martin Jaggi. Random-access infinite context length for transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=7eHn64wOVy.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 37396–37412. PMLR, July 2024. ISSN: 2640-3498.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18724–18741, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1043. URL https://aclanthology.org/2024.emnlp-main.1043/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Niklas Schmidinger, Lisa Schneckenreiter, Philipp Seidl, Johannes Schimunek, Pieter-Jan Hoedt, Johannes Brandstetter, Andreas Mayr, Sohvi Luukkonen, Sepp Hochreiter, and Günter Klambauer. Bio-xlstm: Generative modeling, representation and in-context learning of biological and chemical sequences. *arXiv preprint arXiv:2411.04165*, 2024.

Thomas Schmied, Thomas Adler, Vihang Patil, Maximilian Beck, Korbinian Pöppel, Johannes Brandstetter, Günter Klambauer, Razvan Pascanu, and Sepp Hochreiter. A large recurrent action model: xlstm enables fast inference for robotics tasks. *arXiv preprint arXiv:2410.22391*, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 47901–47911. PMLR, July 2024. ISSN: 2640-3498.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, Long Beach, CA, USA, 2017. Curran Associates, Inc. arXiv: 1706.03762.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. October 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

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

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024a.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Attention Transformers with Hardware-Efficient Training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 56501–56523, Vienna, Austria, 2024b. PMLR. ISSN: 2640-3498.

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

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Re. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4g02l2N2Nx.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *Advances in Neural Information Processing Systems*, 36:34661–34710, December 2023.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Ponti. Inference-Time Hyper-Scaling with KV Cache Compression. October 2025.

APPENDIX

A BACKGROUND: LINEAR ATTENTION AND MLSTM

Linear attention replaces the softmax similarity $\kappa_{\exp}(\boldsymbol{q}, \boldsymbol{k}) = \exp\left(\boldsymbol{q}^\top \boldsymbol{k} / \sqrt{d_{qk}}\right)$ with a kernel $\kappa_\phi(\boldsymbol{q}, \boldsymbol{k}) = \phi(\boldsymbol{q})^\top \phi(\boldsymbol{k})$ that admits an explicit feature representation, where $\phi : \mathbb{R}^{d_{qk}} \to \mathbb{R}^{d_{qk}}$ (Katharopoulos et al., 2020). We use $d_{qk} = d_k$ for the shared query/key head dimension.

Through associativity, this factorization yields two mathematically equivalent causal attention implementations: (i) a (chunkwise) parallel computation suited for training and prefill, and (ii) an online recurrent update used for step-by-step decoding. By switching between these views, one obtains linear-time prefill and training and constant-memory autoregressive generation (Yang et al., 2024b).

In the recurrent formulation, each head maintains a running key–value summary $\boldsymbol{C}_t \in \mathbb{R}^{d_{qk} \times d_v}$, optionally together with a normalizer $\boldsymbol{z}_t \in \mathbb{R}^{d_{qk}}$. Given the token at time t, the state is updated via rank-one outer products:

$$\begin{aligned} \boldsymbol{C}_t &= \boldsymbol{C}_{t-1} + \phi(\boldsymbol{k}_t) \otimes \boldsymbol{v}_t \\ \boldsymbol{z}_t &= \boldsymbol{z}_{t-1} + \phi(\boldsymbol{k}_t) \end{aligned}$$

where $\otimes$ denotes the outer product. For a query $\boldsymbol{q}_t$, the head output is obtained by reading from the current summary with normalization:

$$\boldsymbol{h}_t = \frac{\boldsymbol{C}_t^\top \phi(\boldsymbol{q}_t)}{\phi(\boldsymbol{q}_t)^\top \boldsymbol{z}_t}$$

Here $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^{d_{qk}}$ and $\boldsymbol{v}_t \in \mathbb{R}^{d_v}$.

MLSTM. Recently, modern recurrent architectures, such as XLSTM (Beck et al., 2024), Mamba (Gu & Dao, 2024), and Gated Delta Networks (Yang et al., 2024a) have emerged as competitive linear-complexity alternatives to Transformers. Inspired by the gating structure of the original LSTM cell (Hochreiter & Schmidhuber, 1997), these operators augment the outer product update of linear attention with expressive gates. In this work, we choose MLSTM, which was introduced as a sublayer of XLSTM, as a stateful importance scorer. MLSTM has been shown to perform well in language modelling (Beck et al., 2025), computer vision (Alkin et al., 2024), biological modeling Schmidinger et al. (2024), decision-making (Schmied et al., 2024), and time series forecasting (Auer et al., 2025). The MLSTM introduces three input-dependent gates into the state update of linear attention, each governing a different part of the computation. Let $\boldsymbol{w}_i, \boldsymbol{w}_f \in \mathbb{R}^d$ parameterize scalar input and forget gates, and let $\boldsymbol{W}_{og} \in \mathbb{R}^{d_v \times d}$ parameterize a vector-valued output gate. Given a token representation $\boldsymbol{x}_t \in \mathbb{R}^d$, we compute

$$i_t = \exp(\boldsymbol{w}_i^\top \boldsymbol{x}_t), \qquad f_t = \sigma(\boldsymbol{w}_f^\top \boldsymbol{x}_t), \qquad \boldsymbol{o}_t = \sigma(\boldsymbol{W}_{og} \boldsymbol{x}_t),$$

where i_t scales the new key–value write, f_t attenuates the running state, and $\boldsymbol{o}_t$ gates the readout. The resulting recurrent updates are

$$\begin{aligned} \boldsymbol{C}_t &= f_t\, \boldsymbol{C}_{t-1} + i_t\, \phi(\boldsymbol{k}_t) \otimes \boldsymbol{v}_t \\ \boldsymbol{z}_t &= f_t\, \boldsymbol{z}_{t-1} + i_t\, \phi(\boldsymbol{k}_t) \end{aligned}$$

with numerical stabilization for the exponential input gate omitted here for clarity. A query then performs the usual normalized retrieval, and the output gate modulates it elementwise:

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \frac{\boldsymbol{C}_t^\top \phi(\boldsymbol{q}_t)}{\phi(\boldsymbol{q}_t)^\top \boldsymbol{z}_t} \tag{14}$$

Lightweight stateful scoring with MLSTM. We tailor the MLSTM scorer for minimal retrofit overhead in a pretrained Transformer. We remove the MLSTM output gate, reuse the Transformer's existing attention projections, and avoid storing un-rotated representations across long sequences by separating write and read inputs: recurrent writes and gates use pre-RoPE representations $\boldsymbol{x}_{h,t}^{\text{write}} = [\boldsymbol{k}_{h,t}^{\text{preRoPE}}; \boldsymbol{v}_{h,t}]$, while delayed read queries use cached post-RoPE representations $\boldsymbol{x}_{h,t}^{\text{read}} =$

$[\boldsymbol{k}_{h,t}^{\mathrm{RoPE}}; \boldsymbol{v}_{h,t}]$. For each KV head, head-specific linear projections $W_k, W_q \in \mathbb{R}^{(d_{qk}/2)\times(d_{qk}+d_v)}$ are applied followed by a *Hedgehog* feature map $\phi : \mathbb{R}^{d_{qk}/2} \to \mathbb{R}^{d_{qk}}$ (Zhang et al., 2024):

$$\phi(\boldsymbol{x}) = \left[\mathrm{softmax}(\boldsymbol{x}); \mathrm{softmax}(-\boldsymbol{x})\right] \in \mathbb{R}^{d_{qk}}$$

where the softmax is taken over the feature dimension. Values $\tilde{\boldsymbol{v}}_{h,t} = \boldsymbol{v}_{h,t}$ are used directly without an additional value projection. Crucially, the scorer adds only two small projection matrices (W_k, W_q), two scalar gate projection vectors ($\boldsymbol{w}_i, \boldsymbol{w}_f$), and the final score projection ($\boldsymbol{a}_h, b_h$; Eq. (13)) per KV head, introducing negligible parameter and memory overhead. We base these mLSTM adaptations on Hauzenberger et al. (2026).

B Target Variants and Boundary-Loss Details

This appendix provides details omitted from Section 3.1: GQA aggregation variants, normalization of the future-attention target, temporal score decay, the optional weighting and sampling choices used by the boundary-aware loss, and empirical sensitivity and component ablations of the loss weighting.

B.1 Future-Attention Target Variants

For each KV head h, multiple query heads $g = 1, \ldots, G$ attend to the same key and value vectors. We first compute a per-group future-attention mass,

$$\bar{m}_t^{(h,g)} = \sum_{d=t+w}^{S-1} p_{d\to t}^{(h,g)} \tag{15}$$

and its count-normalized mean future-attention mass $m_t^{(h,g)} = \bar{m}_t^{(h,g)}/N_t$ from Eq. (2), where $N_t = \max(1, S-(t+w))$ is the count of future query positions. Because a single KV entry is shared by all G query heads, these per-group masses are aggregated into one target score per KV head and token. Our default choice is max aggregation over the normalized future attention mass,

$$m_{h,t} = \max_g m_t^{(h,g)} = \frac{1}{N_t} \max_g \bar{m}_t^{(h,g)}, \qquad r_{h,t}^{\mathrm{tgt}} = \log(m_{h,t} + \epsilon) \tag{16}$$

with numerical stability constant $\epsilon = 10^{-6}$. This implements an existential criterion: a token should be retained if any query head that shares the KV entry strongly relies on it. This is the variant used in our main experiments unless specified otherwise.

As an alternative, we also consider probability-space mean aggregation,

$$m_{h,t} = \frac{1}{G}\sum_{g=1}^{G} m_t^{(h,g)} = \frac{1}{N_t}\frac{1}{G}\sum_{g=1}^{G} \bar{m}_t^{(h,g)}, \qquad r_{h,t}^{\mathrm{tgt}} = \log(m_{h,t} + \epsilon) \tag{17}$$

which rewards tokens that are broadly useful across the query heads sharing a KV head. Mean aggregation can be preferable when average shared utility is more important than preserving rare head-specific dependencies, but it may underweight tokens that are crucial for only one query head. When applied to log inputs $a_g = \log(m_t^{(h,g)} + \epsilon)$, as in Eqs. (3) and (19) and Algorithm 3, mean aggregation is $\mathrm{Agg}_g^{\mathrm{mean}}(a_g) = \mathrm{LSE}_g(a_g) - \log G$, which recovers Eq. (17) exactly.

B.2 Count Normalization

The unnormalized future-attention mass in Eq. (15) can be larger for early tokens because they have more future query positions. To prevent early-token bias, we normalize by the number of future queries that can attend to a key after it leaves the protected window:

$$N_t = \max(1, S-(t+w)). \tag{18}$$

As defined in Eq. (3), the target score is computed on the count-normalized mass $m_t^{(h,g)} = \bar{m}_t^{(h,g)}/N_t$:

$$r_{h,t}^{\mathrm{tgt}} = \mathrm{Agg}_g\left[\log\left(\frac{\bar{m}_t^{(h,g)}}{N_t} + \epsilon\right)\right], \tag{19}$$

ensuring that the numerical floor $\epsilon = 10^{-6}$ applies uniformly to the average per-query probability across all positions t. If a finite lookahead window L is used in the target computation, the count is clipped to $N_t \le L+1$.

B.3 Temporal Decay and Static Priorities

Both the teacher and learned retention policies rank eligible tokens with an effective score

$$r_{h,t}(q) = r_{h,t} + (q - t)\log\gamma_h, \qquad \gamma_h \in (0, 1) \tag{20}$$

corresponding to a unit decay step ($n = 1$). This decomposes additively into a query-independent static priority and a token-independent query term:

$$r_{h,t}(q) = \underbrace{(r_{h,t} - t\log\gamma_h)}_{\bar{r}_{h,t}} + q\log\gamma_h. \tag{21}$$

At fixed q, the term $q\log\gamma_h$ is common to all eligible tokens, so the top-k ranking is strictly equivalent to sorting by the static priority

$$\bar{r}_{h,t} = r_{h,t} - t\log\gamma_h. \tag{22}$$

Crucially, in the boundary loss (Eq. (8)), pairwise score differences satisfy

$$\hat{r}_{h,a}(q) - \hat{r}_{h,b}(q) = \bar{r}_{h,a} - \bar{r}_{h,b} = (\hat{r}_{h,a} - \hat{r}_{h,b}) + (b - a)\log\gamma_h. \tag{23}$$

While the common query term $q\log\gamma_h$ cancels, the position offset $(b - a)\log\gamma_h$ remains, providing gradient supervision to the learned per-head decay parameter γ_h.

Step size $n = 1$ versus $n > 1$. If a step size $n > 1$ with age buckets $\lfloor (q - t)/n \rfloor$ were used, the difference $\lfloor (q - a)/n \rfloor - \lfloor (q - b)/n \rfloor$ would vary with $q \pmod n$, altering relative order across queries and preventing a single static priority list. We therefore use $n = 1$ throughout all experiments, providing exact ranking invariance with no runtime re-sorting.

In our implementation, the decay factor is learned per KV head by mapping an unconstrained parameter into a fixed log-decay range,

$$\log\gamma_h = \log\gamma_{\min} + \sigma(\alpha_h)\,(\log\gamma_{\max} - \log\gamma_{\min}) \tag{24}$$

with $0 < \gamma_{\min} < \gamma_{\max} < 1$. This constraint stabilizes training and prevents degenerate decay values. We initialize all heads uniformly to $\gamma_h = 0.9999$ (with $\gamma_{\min} = 0.999$ and $\gamma_{\max} = 0.999999$). Across training runs, the learned per-head values drift toward 1, indicating that only a mild recency bias is preferred.

B.4 Margin Weighting

Some boundary decisions are ambiguous because the newly eligible candidate and the incumbent boundary token have nearly equal target scores. We optionally downweight such examples using the teacher margin

$$\Delta^{\text{tgt}}_{q,h} = y_{q,h}\left(r^{\text{tgt}}_{h,t_{\text{new}}}(q) - r^{\text{tgt}}_{h,t_{\text{bnd}}}(q)\right) \geq 0 \tag{25}$$

The margin weight is

$$\omega^{\text{margin}}_{q,h} = \omega_{\min} + (1 - \omega_{\min})\sigma\left(\frac{\Delta^{\text{tgt}}_{q,h}}{\tau_\omega}\right) \tag{26}$$

where τ_ω controls the margin temperature and $\omega_{\min}$ parameterizes the weighting strength. Because the teacher margin is nonnegative ($\Delta^{\text{tgt}}_{q,h} \geq 0$), the sigmoid satisfies $\sigma(\Delta^{\text{tgt}}_{q,h}/\tau_\omega) \geq \sigma(0) = 0.5$, so the margin-weight component ranges from $(1 + \omega_{\min})/2$ toward one. With the default parameter $\omega_{\min} = 0.2$, a zero-margin decision receives an effective margin weight of $(1+0.2)/2 = 0.60$ before keep/drop balancing and normalization, while large-margin decisions approach weight 1.0. In all our main experiments, we set $\tau_\omega = 1.0$ and $\omega_{\min} = 0.2$ to reduce the influence of ambiguous decisions and near-ties (see sensitivity analysis in Appendix B.6).

B.5 Keep/Drop Balancing

The boundary label distribution may be imbalanced for a given head, especially when the scorer or decay strongly favors either old or newly eligible tokens. We optionally apply a headwise balancing

factor so that keep and drop decisions have similar aggregate weight. Let ρ_h be the fraction of valid sampled boundary decisions with $y_{q,h} = +1$ for head h. We use

$$\omega_{q,h}^{\text{bal}} = \begin{cases} \text{clip}\left(\frac{1}{2\rho_h}, c_{\min}, c_{\max}\right), & y_{q,h} = +1 \\ \text{clip}\left(\frac{1}{2(1-\rho_h)}, c_{\min}, c_{\max}\right), & y_{q,h} = -1 \end{cases} \tag{27}$$

and normalize the resulting weights by their mean over valid examples. We set $c_{\min} = 0.5$ and $c_{\max} = 2.0$. The final weight in Eq. (8) is the product of the margin and balancing weights (see ablation and sensitivity in Appendix B.6).

B.6 Sensitivity and Ablations of Loss Weighting

To assess how sensitive KVPOP is to the loss-weighting hyperparameters and whether these auxiliary terms drive its downstream gains, we conduct sensitivity and component-removal ablations on QWEN3-4B at 75% KV cache compression (CR $= 75\%$). The default configuration employs $\tau_\omega = 1.0$, $\omega_{\min} = 0.2$, and enables both margin weighting ($\omega_{q,h}^{\text{margin}}$) and keep/drop balancing ($\omega_{q,h}^{\text{bal}}$) with class-weight clipping bounds $[c_{\min}, c_{\max}] = [0.5, 2.0]$. These default constants were selected during model development using training diagnostics and held-out validation loss, without tuning against the downstream evaluation benchmarks.

Table 4 reports pass@1 accuracy across the four mathematical reasoning benchmarks (AIME 2024, AIME 2025, HMMT 2502, and HMMT 2511) under variations of these parameters. In these ablation experiments, pass@1 is estimated using 4 stochastic rollouts per prompt (in contrast to the 16 rollouts per prompt used in the main evaluation in Table 1), which accounts for the minor numerical differences in the reference configuration ($0.61, 0.42, 0.31, 0.43$ with an average of 0.4425 versus $0.62, 0.44, 0.31, 0.39$ in Table 1). For reporting transparency, the four-task aggregate averages in Table 4 (and the corresponding delta differences discussed below) are computed directly as the arithmetic mean of the displayed two-decimal rounded benchmark scores (e.g., $(0.61 + 0.42 + 0.31 + 0.43)/4 = 0.4425$). Within each ablation sweep, all training configurations, evaluation seeds, and other hyperparameters are held strictly fixed, and only the ablated component is varied. We observe the following findings:

- **Margin-weight sensitivity ($\omega_{\min}$):** Varying the margin-weight parameter $\omega_{\min} \in \{0.0, 0.2, 0.5\}$ (corresponding to effective margin-weight floors of 0.50, 0.60, and 0.75) has negligible effect on downstream accuracy, yielding an identical four-benchmark average of 0.4400 for both $\omega_{\min} = 0.0$ and $\omega_{\min} = 0.5$, compared to the default 0.4425. This indicates that downstream performance is robust across this range of low-margin downweighting strengths.

- **Margin sharpness sensitivity (τ_ω):** Scaling the margin temperature across $\tau_\omega \in \{0.5, 1.0, 2.0\}$ shows moderate sensitivity: setting $\tau_\omega = 0.5$ (sharper transition) yields 0.4175, while $\tau_\omega = 2.0$ (smoother transition) reaches 0.4325, within 1 percentage point of the default (0.4425).

- **Component removal:** Disabling margin weighting ($\omega_{q,h}^{\text{margin}} = 1$) reduces the observed four-benchmark average from 0.4425 to 0.4200 (a reduction of 2.25 percentage points), while disabling keep/drop balancing ($\omega_{q,h}^{\text{bal}} = 1$) reduces it to 0.4300 (a reduction of 1.25 percentage points). Both components therefore improve observed mean performance in these runs, with margin temperature and component removals producing larger differences than variations in $\omega_{\min}$. These experiments do not separately establish an effect on optimization stability or determine how much of the performance difference relative to baseline policies is attributable to either weighting term.

We note that these experiments evaluate the isolated removal of each weighting term and sweeps over τ_ω and $\omega_{\min}$, while the class-weight clipping values $[c_{\min}, c_{\max}] = [0.5, 2.0]$ were held fixed across all runs, and we did not evaluate a completely unweighted run ($\omega_{q,h} = 1$ simultaneously disabling both terms).

Table 4: **Sensitivity and ablation of boundary-loss weighting.** Downstream pass@1 accuracy (evaluated using 4 rollouts per prompt) on QWEN3-4B at 75% KV cache compression (CR = 75%) under variations of the margin temperature τ_ω, margin-weight parameter $\omega_{\min}$, and individual removal of margin weighting or keep/drop balancing. Averages are computed as the exact mean of the reported two-decimal benchmark scores.

	AIME		HMMT		Average
Setting	2024	2025	2502	2511	Abs.
Default ($\tau_\omega = 1.0, \omega_{\min} = 0.2$)	.61	.42	**.31**	.43	**.4425**
$\tau_\omega = 0.5$	.60	.42	.28	.37	.4175
$\tau_\omega = 2.0$	.57	**.48**	.28	.40	.4325
$\omega_{\min} = 0.0$	**.63**	.45	.28	.40	.4400
$\omega_{\min} = 0.5$	**.63**	.46	.30	.37	.4400
No margin weighting ($\omega_{q,h}^{\text{margin}} = 1$)	.56	.40	.28	**.44**	.4200
No keep/drop balancing ($\omega_{q,h}^{\text{bal}} = 1$)	.59	.43	.29	.41	.4300

B.7 Sampling Query Positions and Boundary Token Handling

For query positions $q < s + w + k$, the number of eligible tokens does not exceed the long-range budget ($|\mathcal{E}(q)| \le k$). All eligible tokens are unconditionally retained without eviction, so no boundary comparison exists and no boundary loss is computed. Once the cache saturates ($q \ge s + w + k$), there are at least k previously eligible tokens ($|\mathcal{E}(q) \setminus \{t_{\text{new}}\}| \ge k$). The boundary token t_{bnd} is defined as the k-th highest-ranked token among previously eligible entries under target score $r_{h,t}^{\text{tgt}}(q)$ (ties broken by larger token index):

$$t_{\text{bnd}} = \arg\operatorname{rank}_k \left(\mathcal{E}(q) \setminus \{t_{\text{new}}\}\right). \tag{28}$$

Defining t_{bnd} over $\mathcal{E}(q) \setminus \{t_{\text{new}}\}$ ensures $t_{\text{bnd}} \neq t_{\text{new}}$ and directly models the online admission decision: t_{new} is admitted if and only if its score is greater than or equal to that of the incumbent cutoff token t_{bnd} (with ties broken in favor of the newer token $t_{\text{new}} > t_{\text{bnd}}$). Importantly, this comparison defines the oracle's admission decision; because the student's realized cache can diverge from the oracle cache during online execution, t_{bnd} represents the cutoff of the teacher policy rather than the student's actual weakest incumbent.

While the running Fenwick tree (Appendix C.2) recomputes the cutoff rank at every query position $q \in \{0, \dots, S-1\}$ to determine the exact sparse attention mask, evaluating $\mathcal{L}_{\text{score}}$ at every saturated position is computationally redundant. We therefore evaluate the boundary loss only on a subsampled set of saturated positions

$$q \ge s + w + k. \tag{29}$$

Our default sampler mixes uniform samples with samples biased toward later positions. This improves coverage of both early saturation behavior and the long-range regime where cache pressure is strongest. The teacher ranks and cutoffs are computed for all positions once, so evaluating the loss at sampled positions costs only a gather and a pairwise logistic term.

C Efficient Running Top-k Sparse Attention

This appendix describes the fixed-budget sparse mask used during training and prefill. During autoregressive decoding, the cache itself contains only retained entries, so attention can be computed directly over the compact cache. During training, however, we need a parallel sparse-attention mask over the full sequence.

C.1 Mask Definition

Fix a layer and KV head h; we suppress these indices on $\operatorname{rank}(t)$, τ_q, and $M_{q,t}$ below. For each query position q, the retained keys are the union of three sets:

1. sink tokens $\{t \mid t < s\}$
2. causal recent-window tokens $\{t \mid q - w < t \leq q\}$
3. the top-k tokens in $\mathcal{E}(q) = \{t \mid s \leq t \leq q - w\}$ under predicted static priorities $\bar{r}_{h,t}$

Let $\mathrm{rank}(t)$ be the rank of token t under the static priority from Eq. (22), with smaller ranks indicating higher priority. Let τ_q be the cutoff rank of the k-th best eligible token at query q. The sparse mask is

$$M_{q,t} = \mathbf{1}[t \leq q] \wedge \left(\mathbf{1}[t < s] \vee \mathbf{1}[q - w < t] \vee (\mathbf{1}[t \in \mathcal{E}(q)] \wedge \mathbf{1}[\mathrm{rank}(t) \leq \tau_q]) \right) \tag{30}$$

This mask enforces the fixed budget $B = s + w + k$ once the sequence is longer than $s + w + k$. We verify through unit tests that this parallel training mask exactly matches step-by-step online cache eviction across dense warm-up, first saturation, and tie-breaking regimes.

C.2 Fenwick-Tree Cutoff Computation

The cutoff sequence τ_q can be computed in $O(S \log S)$ time and $O(S)$ space per head. First, sort tokens once by static priority and compute $\mathrm{rank}(t)$ for all t. Then scan query positions from left to right. When q increases by one, exactly one new token $t = q - w$ enters the eligible set. We insert its rank into a Fenwick tree over ranks. The cutoff τ_q is initialized to ∞ so that all eligible tokens are retained before saturation; once $|\mathcal{E}(q)| \geq k$, τ_q is updated to the smallest rank whose prefix count reaches k, i.e. the rank of the k-th highest-priority eligible token. Algorithm 2 summarizes the procedure.

Require:
static ranks $\mathrm{rank}(0), \ldots, \mathrm{rank}(S-1)$
budget (s, w, k)
initialize Fenwick tree T with zeros
$\tau_q \leftarrow \infty$ for all q ▷ retains all eligible tokens before saturation
for $q = 0, \ldots, S-1$ **do**
  $t \leftarrow q - w$
  if $t \geq s$ **then**
    insert $\mathrm{rank}(t)$ into T
  end if
  if $|\mathcal{E}(q)| \geq k$ **then**
    $\tau_q \leftarrow \mathrm{FindByPrefixCount}(T, k)$
  end if
end for
return $\tau_0, \ldots, \tau_{S-1}$

Algorithm 2: Running top-k cutoff computation for one KV head. A Fenwick tree maintains eligible long-range tokens ordered by their static rank. For each query position, the cutoff τ_q is initialized to ∞ (retaining all eligible tokens before saturation) and updated to the k-th rank once the long-range budget is reached.

The same cutoff computation routine is used for both mask construction and loss supervision: the sparse attention mask queries the student cutoff τ_q after inserting t_{new}, while the teacher boundary target queries the incumbent cutoff over previously eligible entries $\mathcal{E}(q) \setminus \{t_{\mathrm{new}}\} = \mathcal{E}(q-1)$ before insertion (Eq. (28)). Our GPU implementation parallelizes this procedure over batch elements and KV heads. The Fenwick tree is implemented in Triton; binary lifting retrieves the k-th inserted rank without materializing an $S \times S$ mask.

C.3 Use with FlexAttention

Given $\mathrm{rank}(t)$ and τ_q, the predicate in Eq. (30) can be evaluated inside a sparse attention kernel. We use FlexAttention with a mask function that checks whether a key is a sink, belongs to the recent window, or is an eligible long-range token whose rank is below the query-specific threshold. The mask is generated on the fly by the kernel, avoiding explicit storage of a dense boolean matrix.

D Efficient Future-Target Computation

This appendix derives the transposed-attention computation used to obtain the future-attention target in Eq. (3).

D.1 Dense Target Identity

Fix KV head h and query group g. Define the dense causal attention logit

$$\ell^{(h,g)}(d,t) = \langle \boldsymbol{q}_d^{(h,g)}, \boldsymbol{k}_t^{(h)} \rangle / \sqrt{d_k} \tag{31}$$

and the causal log-normalizer

$$\mathrm{LSE}_d^{(h,g)} = \log \sum_{t'=0}^{d} \exp\left(\ell^{(h,g)}(d,t')\right) \tag{32}$$

Then the causal attention probability is

$$p_{d\to t}^{(h,g)} = \exp\left(\ell^{(h,g)}(d,t) - \mathrm{LSE}_d^{(h,g)}\right) \qquad t \le d \tag{33}$$

The unnormalized future-attention mass for key t is

$$\bar{m}_t^{(h,g)} = \sum_{d=t+w}^{S-1} \exp\left(\ell^{(h,g)}(d,t) - \mathrm{LSE}_d^{(h,g)}\right) \tag{34}$$

Taking logs gives

$$\log \bar{m}_t^{(h,g)} = \mathrm{LSE}_{d:t+w\le d<S}\left[\ell^{(h,g)}(d,t) - \mathrm{LSE}_d^{(h,g)}\right] \tag{35}$$

Thus the target is a log-sum-exp over future queries for each fixed key.

D.2 Transposed Attention

Equation (35) can be evaluated by a second attention-like pass with swapped query and key roles. Define transposed inputs

$$\boldsymbol{q}'_t = \boldsymbol{k}_t^{(h)} \qquad \boldsymbol{k}'_d = \boldsymbol{q}_d^{(h,g)} \tag{36}$$

Their dot product gives the original attention logit $\ell^{(h,g)}(d,t)$. We then apply a score modifier that subtracts the original query normalizer,

$$\ell^{(h,g)}(d,t) \mapsto \ell^{(h,g)}(d,t) - \mathrm{LSE}_d^{(h,g)} \tag{37}$$

and a mask enforcing $d \ge t + w$. The auxiliary log-sum-exp returned for transposed query position t is exactly Eq. (35). Because the value output of this second pass is not used, the value tensor can be any tensor with a compatible shape. Although this pass computes $\mathcal{O}(S^2)$ query–key dot products over the sequence horizon S, streaming the log-sum-exp reduction directly within the fused attention kernel avoids materializing the $\mathcal{O}(S^2)$ attention matrix in global GPU memory, preserving an $\mathcal{O}(S)$ activation memory footprint.

D.3 Sparse Normalizer Approximation

If Eq. (32) is used, the transposed computation gives the exact dense future-attention target. To reduce training cost, we may reuse the log-normalizer returned by the sparse attention pass:

$$\widetilde{\mathrm{LSE}}_d^{(h,g)} = \log \sum_{t'=0}^{d} M_{d,t'} \exp\left(\ell^{(h,g)}(d,t')\right) \tag{38}$$

where M is the fixed-budget sparse mask. Replacing $\mathrm{LSE}_d^{(h,g)}$ with $\widetilde{\mathrm{LSE}}_d^{(h,g)}$ yields an approximate target. Formally, let $\rho_d = \sum_{j:M_{d,j}=1} p_{d\to j}$ be the retained dense-attention probability mass at query position d. Replacing the dense normalizer with the sparse student normalizer scales each query's attention probabilities by $\tilde{p}_{d\to t} = p_{d\to t}/\rho_d$, yielding the reweighted future-mass target:

$$\tilde{m}_t^{(h,g)} = \frac{1}{N_t} \sum_{d=t+w}^{S-1} \frac{p_{d\to t}^{(h,g)}}{\rho_d}. \tag{39}$$

Because $\rho_d \leq 1$ varies across query positions d, future queries are weighted inversely by their captured mass $1/\rho_d$. When the retained set captures most of the dense-attention mass, the query-dependent reweighting is small. In the evaluated 75% compression setting, dense and sparse normalization produce similar downstream performance. This comparison does not directly measure retained-mass or target-ranking agreement. This approximation is practical because it reuses quantities already produced by the forward sparse-attention call and avoids an additional dense causal pass. In settings where exact supervision is desired, the dense normalizer can be computed instead.

Empirical comparison: Dense vs. sparse LSE. To validate whether this approximation incurs an empirical performance penalty, we train matched QWEN3-4B models under identical conditions at 75% KV cache reduction (CR $= 75\%$), differing only in whether the future-attention target uses the exact dense causal LSE or the sparse student $\widetilde{\text{LSE}}$. As shown in Table 5, the downstream mathematical reasoning accuracy across all four benchmarks (evaluated using 4 rollouts per prompt, matching the ablation protocol in Appendix B.6) is virtually indistinguishable: the four-task averages differ by only 0.5 percentage points (0.4475 for dense vs. 0.4425 for sparse, computed directly as the arithmetic mean of the displayed two-decimal benchmark accuracies). These results show that in the evaluated 75% compression setting, dense and sparse normalization produce similar downstream performance, indicating that reusing the student's sparse normalizer provides an effective efficiency optimization that avoids the overhead of a separate dense-attention pass during training while preserving policy quality.

Table 5: **Ablation of dense vs. sparse query LSE normalizer.** We compare KVPOP on QWEN3-4B at 75% KV cache reduction (CR $= 75\%$) trained with the exact dense causal log-sum-exp (LSE) versus the student's sparse log-sum-exp ($\widetilde{\text{LSE}}$), evaluated with 4 rollouts per prompt. Averages are the arithmetic mean of the reported two-decimal benchmark scores.

	AIME		HMMT		Average
Target LSE Normalizer	2024	2025	2502	2511	Abs.
Dense (LSE)	.59	**.47**	.29	**.44**	**.4475**
Sparse ($\widetilde{\text{LSE}}$)	**.61**	.42	**.31**	.43	.4425

D.4 Implementation Steps

The target computation proceeds as follows:

1. Run the main attention pass and obtain per-query log-normalizers. These may be dense causal normalizers or sparse normalizers, depending on the chosen target variant.
2. Run a transposed attention pass with keys as queries and queries as keys.
3. In the transposed pass, subtract the original per-query log-normalizer as a score modifier and mask out positions with $d < t + w$.
4. Use the auxiliary log-sum-exp from the transposed pass to obtain $\log \bar{m}_t^{(h,g)}$ for all keys t.
5. Apply count normalization (Appendix B.2) to the per-group masses, incorporate ϵ, and aggregate across query groups according to Appendix B.1 to obtain $r_{h,t}^{\text{tgt}}$.

The transposed pass is used only while training the scorer. It is not required for autoregressive decoding.

E Scorer Architecture Details

This appendix describes the stateless and stateful scorer variants used with the same target and boundary loss.

E.1 Headwise Inputs and Read/Write Representations

For each KV head h and token t, we separate the representations used for the recurrent state update (write) from those used for the delayed query (read):

$$\boldsymbol{x}_{h,t}^{\text{write}} = [\boldsymbol{k}_{h,t}^{\text{preRoPE}}; \boldsymbol{v}_{h,t}], \qquad \boldsymbol{x}_{h,t}^{\text{read}} = [\boldsymbol{k}_{h,t}^{\text{RoPE}}; \boldsymbol{v}_{h,t}] \tag{40}$$

Recurrent write features and gates are computed from pre-RoPE keys and values available during the current attention layer computation. Delayed read queries are computed from the corresponding post-RoPE keys and values already stored in the cache. This separates the write and read representations without retaining a separate history of hidden states or un-rotated representations. Using cached $[\boldsymbol{k}^{\text{RoPE}}; \boldsymbol{v}]$ for delayed queries has two distinct advantages. First, it aligns the scorer with the KV heads used in grouped-query attention. Second, it enables delayed scoring without storing additional hidden states, because both post-RoPE keys and values are already maintained in the cache throughout the protected window until the token's eviction decision is made.

E.2 Stateless Linear and MLP Scorers

A stateless scorer predicts each token's score independently from its representation at insertion time:

$$\hat{r}_{h,t} = f_\theta^{(h)}(\boldsymbol{x}_{h,t}) \tag{41}$$

We consider a headwise linear layer and a small headwise MLP. These scorers add minimal overhead and are easy to parallelize during training and prefill. Their limitation is that they cannot incorporate context that arrives after token insertion.

E.3 mLSTM Scorer

Our memory-based scorer uses an mLSTM-style recurrent state for each KV head. Given the separate write and read inputs, small headwise projections produce features for the recurrent write and delayed read:

$$\tilde{\boldsymbol{k}}_{h,t} = \phi(W_k^{(h)} \boldsymbol{x}_{h,t}^{\text{write}}), \qquad \tilde{\boldsymbol{q}}_{h,t} = \phi(W_q^{(h)} \boldsymbol{x}_{h,t}^{\text{read}}), \qquad \tilde{\boldsymbol{v}}_{h,t} = \boldsymbol{v}_{h,t} \tag{42}$$

with $W_k^{(h)}, W_q^{(h)} \in \mathbb{R}^{(d_{qk}/2)\times(d_{qk}+d_v)}$. The headwise projections produce $d_{qk}/2$ features; the Hedgehog feature map (Zhang et al., 2024) $\phi : \mathbb{R}^{d_{qk}/2} \to \mathbb{R}^{d_{qk}}$ concatenates their positive and negative softmax features to obtain d_{qk}-dimensional read and write features, keeping the parameter overhead small. In our primary evaluated configuration, values $\tilde{\boldsymbol{v}}_{h,t} = \boldsymbol{v}_{h,t}$ are used directly from the cache without an additional learned projection matrix.

The recurrent state consists of a matrix memory $\boldsymbol{C}_{h,t}$ and a normalizer $\boldsymbol{z}_{h,t}$. With scalar input and forget gates $i_{h,t}$ and $f_{h,t}$ computed from the write representation $\boldsymbol{x}_{h,t}^{\text{write}}$, the update is

$$\boldsymbol{C}_{h,t} = f_{h,t}\boldsymbol{C}_{h,t-1} + i_{h,t}\tilde{\boldsymbol{k}}_{h,t} \otimes \tilde{\boldsymbol{v}}_{h,t} \tag{43}$$

$$\boldsymbol{z}_{h,t} = f_{h,t}\boldsymbol{z}_{h,t-1} + i_{h,t}\,\tilde{\boldsymbol{k}}_{h,t} \tag{44}$$

The gates are produced by two separate headwise linear projections of $\boldsymbol{x}_{h,t}^{\text{write}}$:

$$\begin{aligned} \alpha_{h,t}^{(i)} &= \boldsymbol{w}_i^{(h)\top} \boldsymbol{x}_{h,t}^{\text{write}} + b_i^{(h)}, \qquad \alpha_{h,t}^{(f)} = \boldsymbol{w}_f^{(h)\top} \boldsymbol{x}_{h,t}^{\text{write}} + b_f^{(h)}, \\ i_{h,t} &= \exp(\alpha_{h,t}^{(i)}), \qquad f_{h,t} = \sigma(\alpha_{h,t}^{(f)}) \end{aligned} \tag{45}$$

with optional softcapping of the preactivations for numerical stability. In the implementation, we use the numerically stabilized mLSTMformulation.

E.4 Pipelined Delayed Readout and Admission Timing

We distinguish scoring time from long-range cache admission time. A token t receives its retention score after processing position $t + d$ with scoring delay $d = w - 1$, while it is on its final step inside the protected recent window. The score is stored in the cache and retrieved at position $t + w$, when the token first exits the protected window and competes for the long-range budget. Thus, at admission

position $q = t + w$, the newly eligible token $u = q - w$ is ranked using the score precomputed from the recurrent state after position $q - 1 = u + (w - 1)$:

$$\boldsymbol{h}_{h,u} = \frac{\boldsymbol{C}_{h,q-1}^{\top} \tilde{\boldsymbol{q}}_{h,u}}{\tilde{\boldsymbol{q}}_{h,u}^{\top} \boldsymbol{z}_{h,q-1}} \tag{46}$$

The raw score is produced by

$$\hat{r}_{h,u} = \boldsymbol{a}_h^{\top} \operatorname{SiLU}(\boldsymbol{h}_{h,u}) + b_h. \tag{47}$$

For $w = 256$, token 744 is scored from the state after position 999 ($d = 255$) and competes for long-range admission at position 1000. At position 1000, the newly computed score is for token 745, stored to be used at position 1001. This pipelined readout is causal because all information in $\boldsymbol{C}_{h,q-1}$ is available prior to position q. It is forward-looking relative to token u, since the state summarizes $w - 1$ steps of accumulated context before u is considered for eviction.

E.5 Initialization and Practical Variants

We initialize the scorer so that the recurrent projections start close to simple identity-style mappings from the base attention keys. The query and key scorer projections are initialized from the key pathway, and the final score projection is zero-initialized so that the sparse policy starts from a neutral ranking before training. Gate biases are initialized to encourage stable memory updates early in training.

In our primary evaluated checkpoints (QWEN3-4B and QWEN3-8B at 4× and 8× compression), the scorer configuration uses:

- **Scoring delay:** $d = w - 1$ (255 steps for $w = 256$), pipelining the scoring step so scores are precomputed on the last protected step;
- **Feature representations:** pre-RoPE keys and values for recurrent writes and gates ($\boldsymbol{x}_{h,t}^{\text{write}}$); post-RoPE keys and values for delayed read queries ($\boldsymbol{x}_{h,t}^{\text{read}}$);
- **Feature map:** Hedgehog feature map ϕ for $\tilde{\boldsymbol{k}}$ and $\tilde{\boldsymbol{q}}$;
- **Values:** direct cached values ($\tilde{\boldsymbol{v}}_{h,t} = \boldsymbol{v}_{h,t}$) without an additional value projection (W_v);
- **Architectural options:** no skip connections from the read token to the final score and no output normalization layer.

All variants are trained with the same target and boundary loss, allowing us to isolate the effect of memory-based delayed scoring (see Appendix I).

F Experiment Details

Training details. All experiments were run on 8 H100 GPUs using PyTorch FSDP. We train for 2,000 steps with a global batch size of 128 packed sequences of length 16,384 via gradient accumulation, corresponding to a total budget of 4.2 billion tokens ($2{,}000 \times 128 \times 16{,}384 \approx 4.19$B). We use mixed precision (bfloat16 for model parameters and most compute; float32 for gradient all-gather/reductions), and gradient clipping to 1.0 for full finetuning. For numerical stability, we compute score decay and top-k ranking/selection in float32; all other activations remain in bfloat16.

To maximize GPU utilization, we pack multiple samples into a single sequence up to the maximum context length. We found that *preserving the attention mask across packed segments* (i.e., not resetting attention at packing boundaries) improved performance for our hybrid architecture, consistent with observations in prior work (Buitrago & Gu, 2025).

We train KVPOPby minimizing the KL Divergence to the teacher. For efficiency we compute the KL term over the top-256 teacher logits. This truncation substantially reduces the cost of teacher supervision and suggests a variant where teacher logits could be precomputed offline, avoiding loading the teacher model during training. We leave this optimization for future work.

We evaluate all models with the same decoding configuration to ensure that performance differences reflect the retention policy rather than sampling variability. We use nucleus sampling with temperature 0.6, top-p 0.95, top-k 20, no repetition penalty, and a zero-shot prompt format.

Table 6: Training and Inference Hyperparameters.

Training General	
Token budget	4.2B (2,000 steps)
Context size	16384
Batch size	128
Weight decay	0.0001
Optimizer	AdamW
Adam Betas	(0.9, 0.95)
Trainable Parameters	all
Sequence packing	true
Learning rate (existing params)	Constant LR 8e−5
Learning rate (new params)	100 Steps Warmup + Cosine Schedule (1e−3 to 8e−5)
KL temperature	1
KL reverse	false
KL Top-k	256
Training KVPOP	
Decay step size	1
Pairwise loss temperature	1.0
Pairwise loss margin weighting	true ($\tau_\omega = 1.0, \omega_{\min} = 0.2$)
Keep/drop balancing	true
Keep/drop clipping bounds ($[c_{\min}, c_{\max}]$)	$[0.5, 2.0]$
Target stability constant (ϵ)	10^{-6}
Sliding window (w)	256
Scoring delay ($d = w - 1$)	255
Sink tokens (s)	4
Recurrent write features ($\boldsymbol{x}^{\text{write}}$)	Pre-RoPE keys and values
Delayed read queries ($\boldsymbol{x}^{\text{read}}$)	Post-RoPE keys and values
Recurrent feature map (ϕ)	Hedgehog
Value projection (W_v)	None (direct cached values)
Top-k budget (stateless / MLP)	1788 ($B = 2048$) & 3836 ($B = 4096$)
Top-k budget (stateful / mLSTM)	1724 ($B = 2048$) & 3772 ($B = 4096$)
mLSTM state memory per head	64 KV tokens (32 KB)
Total persistent memory per head	2048 & 4096 token-equiv.
Decay init	0.9999
Decay min	0.999
Decay max	0.999999
Inference	
Temperature	0.6
Top-p	0.95
Top-k	20
Min-p	0.0
Repetition penalty	1.0
Number of shots	0

Table 6 summarizes the hyperparameters used throughout our training and evaluation pipeline.

F.1 Cache Budget and Persistent Memory Accounting

All evaluations enforce persistent memory budgets defined relative to a reference context length of $S = 16{,}384$. The two compression settings, nominally labeled 75% and 88%, correspond to total capacities of $B = 4{,}096$ tokens ($1 - 4096/16384 = 75.0\%$) and $B = 2{,}048$ tokens ($1 - 2048/16384 = 87.5\% \approx 88\%$).

Table 7: **Cache budget and persistent memory accounting.** For KVPOP and StreamingLLM+, budgets are strictly and uniformly enforced *per KV head* (2.0 MB at 75% compression, 1.0 MB at 88% compression). For DMS, values indicate the *model-wide average* per head matching the overall 75.0% and 87.5% target compression ratios across the model, with capacity distributed heterogeneously across heads via dynamic gates (Appendix H). One KV token occupies 512 bytes ($d_k = d_v = 128$, `bfloat16`). The MLSTM state matrix $\boldsymbol{C} \in \mathbb{R}^{128\times128}$ occupies 32 KB (64 token-equivalents).

Setting	Method	Sinks (s)	Window (w)	Top-k (k)	KV Tokens	Scorer State	Total Memory	Nominal CR
$B = 4096$ (75%)	StreamingLLM+	4	4092	–	4096	0 KB (0 tok)	2.0 MB (4096 tok)	75.0%
	DMS	–	–	–	4096*	0 KB (0 tok)	2.0 MB* (4096 tok)	75.0%
	KVPOP$_{\text{mlp}}$	4	256	3836	4096	0 KB (0 tok)	2.0 MB (4096 tok)	75.0%
	KVPOP$_{\text{MLSTM}}$	4	256	3772	4032	32 KB (64 tok)	2.0 MB (4096 tok)	75.0%
$B = 2048$ (88%)	StreamingLLM+	4	2044	–	2048	0 KB (0 tok)	1.0 MB (2048 tok)	87.5%
	DMS	–	–	–	2048*	0 KB (0 tok)	1.0 MB* (2048 tok)	87.5%
	KVPOP$_{\text{mlp}}$	4	256	1788	2048	0 KB (0 tok)	1.0 MB (2048 tok)	87.5%
	KVPOP$_{\text{MLSTM}}$	4	256	1724	1984	32 KB (64 tok)	1.0 MB (2048 tok)	87.5%

*Model-wide average per head (heterogeneous across heads); matches total aggregate KV cache memory across the model.

For KVPOP and StreamingLLM+, this budget is enforced *strictly and uniformly per KV head*. StreamingLLM+ retains $s = 4$ sink tokens (2 KB) and a sliding window of $w = B - 4$ recent tokens ($w = 4{,}092$ at 75% and $w = 2{,}044$ at 88%), with no long-range top-k cache ($k = 0$). KVPOP retains $s = 4$ sink tokens and a local protected window of $w = 256$ recent tokens (128 KB), allocating the remaining budget $k = B - (s + w)$ to the long-range cache ($k = 3{,}836$ at 75% and $k = 1{,}788$ at 88% for stateless models). In contrast, DMS (Łańcucki et al., 2025) does not enforce a uniform per-head budget. Instead, DMS is trained to meet an *overall model-wide compression target* (75.0% or 87.5% total sparsity across all heads and layers). Its dynamic eviction gates allocate capacity heterogeneously across heads and layers (as analyzed in Appendix H and Figure 8)—with certain heads retaining nearly dense caches while others collapse to local windows. Thus, for DMS, the 4,096 and 2,048 token figures represent the *model-wide average* per head matching the total aggregate cache footprint of KVPOP at the reference sequence length, rather than an identical per-head cap.

For QWEN3-4B and QWEN3-8B, the key and value head dimensions are $d_k = d_v = 128$ in 16-bit precision (`bfloat16`). A single cached token therefore requires $2 \times 128 \times 2 = 512$ bytes per KV head.

Accounting for stateful scorer memory. While the stateless scorer (KVPOP$_{\text{mlp}}$) requires no persistent recurrent state (0 bytes), the stateful MLSTM scorer maintains a recurrent memory matrix $\boldsymbol{C} \in \mathbb{R}^{128\times128}$ and normalizer vector $\boldsymbol{z} \in \mathbb{R}^{128}$ per KV head in 16-bit precision. The memory matrix requires $128 \times 128 \times 2 = 32{,}768$ bytes (32 KB), which is exactly equivalent to $32{,}768/512 = 64$ KV tokens. To ensure an exact match in persistent memory between stateless and stateful variants, we reduce the top-k budget of KVPOP$_{\text{MLSTM}}$ by exactly 64 tokens:

- **75% compression ($B = 4096$):** Stateless models retain $k = 3836$ long-range tokens ($s + w + k = 4096$ total KV entries, 2.0 MB per head). The stateful model retains $k = 3772$ long-range tokens ($s + w + k = 4032$ total KV entries, 1.969 MB per head); adding the 32 KB recurrent state yields exactly 2.0 MB (4096 token equivalents).
- **88% compression ($B = 2048$):** Stateless models retain $k = 1788$ long-range tokens ($s + w + k = 2048$ total KV entries, 1.0 MB per head). The stateful model retains $k = 1724$ long-range tokens ($s + w + k = 1984$ total KV entries, 0.969 MB per head); adding the 32 KB recurrent state yields exactly 1.0 MB (2048 token equivalents).

Table 7 summarizes the detailed budget and memory breakdown across evaluated methods.

F.2 Statistical Significance Testing

To evaluate whether the downstream gains of KVPOP over baseline eviction policies are statistically significant, we perform paired statistical tests across all 120 mathematical reasoning problems (30 problems each for AIME 2024, AIME 2025, HMMT 2502, and HMMT 2511). For every problem, pass@1 accuracy is estimated across 16 stochastic rollouts, which substantially reduces evaluation variance compared to small sample sizes.

We compare methods across all 120 mathematical reasoning problems using two-sided exact paired sign-flip tests on the problem-level success rates (aggregated over 16 rollouts per problem). Within each model size and compression setting, the three pairwise comparisons (KVPOP$_{\text{mlp}}$ vs. DMS, KVPOP vs. DMS, and KVPOP$_{\text{mlp}}$ vs. KVPOP) form a separate Holm–Bonferroni correction family, and we report Holm-adjusted p-values. In addition, we compute 95% confidence intervals via 100,000 paired, benchmark-stratified bootstrap resamples over problems.

Table 8 presents the statistical comparisons across settings, displaying the pairwise comparisons that achieve statistical significance alongside the representative non-significant 4B 75% setting (where unsubscripted KVPOP refers to the stateful KVPOP$_{\text{MLSTM}}$ variant). At 75% KV cache compression on QWEN3-4B, no pairwise difference among the evaluated policies is statistically significant ($p \geq 0.05$). However, under more aggressive 88% cache compression on QWEN3-4B, KVPOP significantly outperforms DMS by 3.33 percentage points (95% CI $[+1.25, +5.52]$, $p = 0.0092$). On QWEN3-8B, KVPOP achieves statistically significant gains over DMS across both evaluated budgets: by 2.76 percentage points at 75% compression (95% CI $[+0.83, +4.69]$, $p = 0.0217$) and by 7.03 percentage points at 88% compression (95% CI $[+4.74, +9.48]$, $p = 2.51 \times 10^{-8}$). Furthermore, the stateless KVPOP$_{\text{mlp}}$ variant also significantly outperforms DMS at 88% compression on QWEN3-8B by 5.89 percentage points (95% CI $[+3.12, +8.80]$, $p = 0.00016$). These results substantiate that KVPOP's performance advantage over learned baselines is statistically reliable, with the clearest and strongest gains emerging under tighter cache constraints.

Direct comparison between stateful KVPOP$_{\text{MLSTM}}$ and stateless KVPOP$_{\text{mlp}}$ reveals that the lightweight MLP scorer is already a remarkably strong baseline: across mathematical reasoning benchmarks, stateful MLSTM yields modest differences ranging from 0.0 to +2.0 percentage points, and none of the direct pairwise comparisons between KVPOP$_{\text{MLSTM}}$ and KVPOP$_{\text{mlp}}$ reach statistical significance ($p \geq 0.05$ after Holm–Bonferroni correction within each family). For this reason, Table 8 reports the statistically significant pairwise differences and omits the non-significant stateful-vs-MLP pairs. On out-of-domain evaluation (Table 3), KVPOP$_{\text{mlp}}$ similarly matches or slightly exceeds KVPOP$_{\text{MLSTM}}$. This indicates that the complete boundary-supervised eviction recipe provides the primary performance advantage over baselines already with stateless scoring, with the recurrent MLSTM scorer providing a setting-dependent incremental benefit for long-horizon reasoning where evidence accumulates over time, rather than an unconditional improvement across all settings.

Table 8: **Statistical significance testing on mathematical reasoning benchmarks.** Two-sided exact paired sign-flip tests on problem-level pass@1 rates (16 rollouts per problem across AIME and HMMT) with 95% bootstrap confidence intervals and Holm–Bonferroni-adjusted p-values. Unsubscripted KVPOP denotes KVPOP$_{\text{MLSTM}}$.

Model	CR	Comparison	Δ (% pts)	95% Bootstrap CI	Holm-adj. p
QWEN3-4B	75%	KVPOP vs. DMS	+1.00	[-1.12, +3.15]	≥ 0.05 (n.s.)
	88%	KVPOP vs. DMS	3.33	[+1.25, +5.52]	0.0092
QWEN3-8B	75%	KVPOP vs. DMS	2.76	[+0.83, +4.69]	0.0217
	88%	KVPOP vs. DMS	7.03	[+4.74, +9.48]	2.51×10^{-8}
	88%	KVPOP$_{\text{mlp}}$ vs. DMS	5.89	[+3.12, +8.80]	0.00016

F.3 LONG-CONTEXT RETRIEVAL: RULER SINGLE-NEEDLE TASKS

To evaluate retention of factual content across long prefill sequences where eviction occurs before query time, we evaluate QWEN3-4B on the three single-needle retrieval tasks (NIAH1, NIAH2, and NIAH3) from the RULER benchmark (Hsieh et al., 2024) at context lengths of 4k and 16k tokens under both 75% and 88% KV cache compression. To maintain identical compression ratios across context lengths, the per-head persistent budget B is scaled proportionally to sequence length: at 4k context, $B = 1{,}024$ (75% compression, $k = 700$ for MLSTM) and $B = 512$ (88% compression, $k = 188$); at 16k context, $B = 4{,}096$ ($k = 3772$) and $B = 2{,}048$ ($k = 1724$). As shown in Table 9, KVPOP retains perfect retrieval (1.00 average) across all three tasks at 4k context length under both compression ratios, demonstrating that even when 75% or 88% of cache tokens are evicted, essential factual anchors are reliably preserved. At 16k context length, KVPOP maintains strong retrieval performance, achieving an average accuracy of 0.95 at 75% compression (degrading slightly on

NIAH3 to 0.85) and 0.80 at 88% compression (degrading on NIAH2 to 0.87 and NIAH3 to 0.54). The learned baseline DMS achieves perfect retrieval (1.00) across all settings. This difference is consistent with the distinct allocation dynamics of the two methods: as analyzed in Appendix H and Figure 8, DMS's data-dependent eviction gates collapse to full dense attention in a subset of attention heads, which is exceptionally well suited for single-needle retrieval because retaining all tokens in even a single head preserves the needle. In contrast, KVPOP enforces a strictly uniform per-head budget ($s + w + k$) across every KV head to guarantee homogeneous execution and constant per-head memory throughout generation, while still preserving the vast majority of retrieval accuracy across long contexts.

Table 9: **RULER single-needle retrieval on QWEN3-4B.** Accuracy on NIAH1, NIAH2, and NIAH3 across 4k context (total persistent budget $B = 1{,}024$ for 75%, $B = 512$ for 88%) and 16k context ($B = 4{,}096$ for 75%, $B = 2{,}048$ for 88%).

		4k Context Length				16k Context Length			
CR	Method	NIAH1	NIAH2	NIAH3	Avg.	NIAH1	NIAH2	NIAH3	Avg.
Teacher		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
75%	DMS	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
	KVPOP	1.00	1.00	1.00	1.00	1.00	1.00	.85	.95
88%	DMS	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
	KVPOP	1.00	1.00	1.00	1.00	1.00	.87	.54	.80

F.4 INFERENCE EFFICIENCY BENCHMARKING SETUP

Hardware and software configuration. All inference benchmarks in Section 4 and Figure 4 were measured on a single NVIDIA H100 80GB SXM5 GPU using PyTorch 2.10 in `bfloat16` precision. Attention operations utilize the FlashAttention backend, and graph compilation is enabled using `torch.compile`.

Benchmarking protocol. Measurements evaluate end-to-end autoregressive decoding at batch size 1 initiated from a fixed prompt length of $L_{\text{prompt}} = 1{,}024$ tokens. Autoregressive decoding proceeds up to total generation lengths $L \in \{16{,}384, 32{,}768, 65{,}536, 128{,}000\}$ tokens. At each evaluated generation length, the cache budget is matched to maintain a constant 88% compression ratio ($B = L/8$ tokens per head, matching the $8\times$ compression target of DMS (Łańcucki et al., 2025); we use the official checkpoint and cache implementation from nvidia/Qwen3-8B-DMS-8x, matching scorer parameters with KVPOP as noted in Section 4). For KVPOP, we evaluate the stateful $\text{KVPOP}_{\text{MLSTM}}$ variant with $w = 256$ window tokens and $s = 4$ sinks, allocating the remaining budget $k = B - (s + w) - 64$ to the long-range cache while accounting for the 64-token-equivalent persistent MLSTM memory per head (Table 7). Each configuration is executed for five steady-state repetitions, excluding initial compilation and CUDA warmup iterations, and we report the mean end-to-end latency and peak allocated VRAM.

Component-level memory breakdown. Table 10 reports the concrete breakdown between model weights, uncompressed dense KV cache, compressed KV cache, and scorer state for QWEN3-8B across evaluated sequence lengths. Base model weights in `bfloat16` occupy ≈ 15.5 GB. For dense attention, the uncompressed KV cache scales linearly at 512 bytes per token across 36 layers and 8 KV heads (≈ 0.147 MB per token across all layers), requiring 2.4 GB at 16k tokens and rising to 18.8 GB at 128k tokens, driving peak allocated VRAM to 35.8 GB. Under 88% compression ($8\times$ reduction), the cached tokens occupy only $1/8$-th of this footprint (≈ 0.30 GB at 16k tokens and ≈ 2.35 GB at 128k tokens). Persistent stateful MLSTM memory across all 36 layers and 8 heads adds $36 \times 8 \times 32$ KB ≈ 9.2 MB. Consequently, peak allocated VRAM for KVPOP scales modestly from 15.8 GB at 16k tokens to 18.8 GB at 128k tokens, maintaining an ≈ 17 GB peak VRAM margin over dense attention at long horizons.

Table 10: **Memory breakdown for QWEN3-8B across generation lengths at 88% compression.** Base model weights occupy ≈ 15.5 GB in `bfloat16`. For sparse methods, the persistent cache budget is matched to enforce constant 88% compression ($B = L/8$) at each length. Stateful KVPOP$_{\text{MLSTM}}$ maintains 9.2 MB of persistent recurrent state across all 36 layers and 8 heads ($36 \times 8 \times 32$ KB). Total peak VRAM is measured on a single NVIDIA H100 GPU (batch size 1, prompt length 1,024, FlashAttention, `torch.compile`).

Generation Length	Dense Attention		KVPOP$_{\text{MLSTM}}$ (88% Compression)			
(L)	KV Cache	Peak VRAM	Budget (B)	KV Cache	Scorer State	Peak VRAM
16k (16,384)	2.4 GB	18.0 GB	2,048	0.30 GB	9.2 MB	15.8 GB
32k (32,768)	4.8 GB	20.5 GB	4,096	0.60 GB	9.2 MB	16.1 GB
64k (65,536)	9.7 GB	25.5 GB	8,192	1.20 GB	9.2 MB	16.8 GB
128k (128,000)	18.8 GB	35.8 GB	16,000	2.35 GB	9.2 MB	18.8 GB

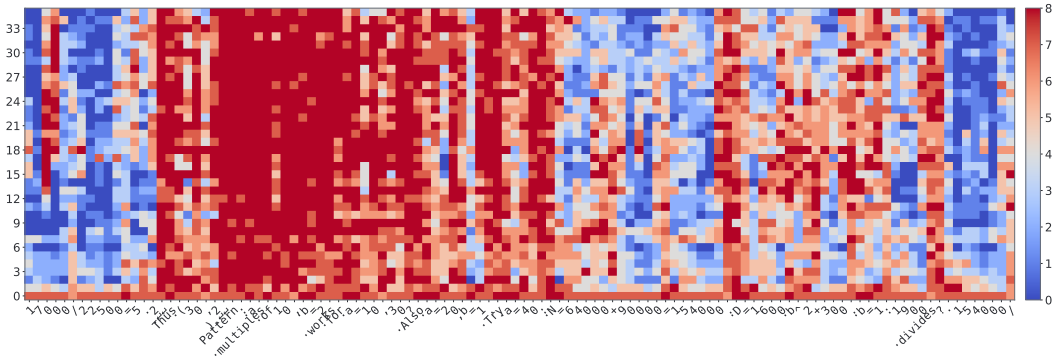


Figure 6: **Eviction patterns in a mathematical reasoning trace.** We visualize the last 112 tokens of a randomly sampled sequence. Rows denote layers, columns denote token strings, and color indicates how many attention heads in a layer retained the corresponding token. KVPOP tends to evict purely numeric tokens more often, while retaining reasoning-structural tokens.

G KVPOP SPARSITY PATTERNS

Eviction pattern visualization. Figure 6 visualizes token retention over the final 112 evictable tokens of a randomly sampled mathematical reasoning sequence. Rows are transformer layers, columns are token strings, and color encodes how many attention heads retain each token. KVPOP does not retain tokens uniformly. Purely numeric tokens are dropped more often, while tokens that organize or advance the reasoning, such as discourse markers (*Thus*), operation words (*multiplies*), and symbolic tokens (=), are retained by more heads and across more layers. The first layer is a notable exception, retaining nearly all tokens across heads in the displayed 112-token crop, consistent with prior observations that early layers perform broader processing before higher layers specialize (Tenney et al., 2019). This provides a qualitative observation consistent with content-dependent retention beyond simple recency, preferentially keeping tokens useful for interpreting the evolving solution. Additional examples are in Figure 9. As a complementary comparison, Appendix H analyzes the sparsity patterns learned by DMS, showing that its dynamic gates concentrate long-range capacity in a small subset of heads.

Recovering oracle eviction decisions. In Figure 7 we plot the top-k retention recall distribution over heads across all layers of QWEN3-4B at 75% KV cache compression. For a given head, the metric measures what fraction of the future-attention retention oracle's top-k tokens (derived from the target r^{tgt}) are also kept by the learned KVPOP scorer under the same cache budget. Most layer-wise distributions remain in a high range, with a global mean recall of 81%, indicating high agreement with the retention oracle policy.

Figure 9 shows additional qualitative examples of the learned token retention patterns. Across sequences, KVPOPdoes not simply preserve the most recent tokens or apply a uniform sparsity pattern. Instead, it often retains contiguous blocks of text that appear to carry the structure of the

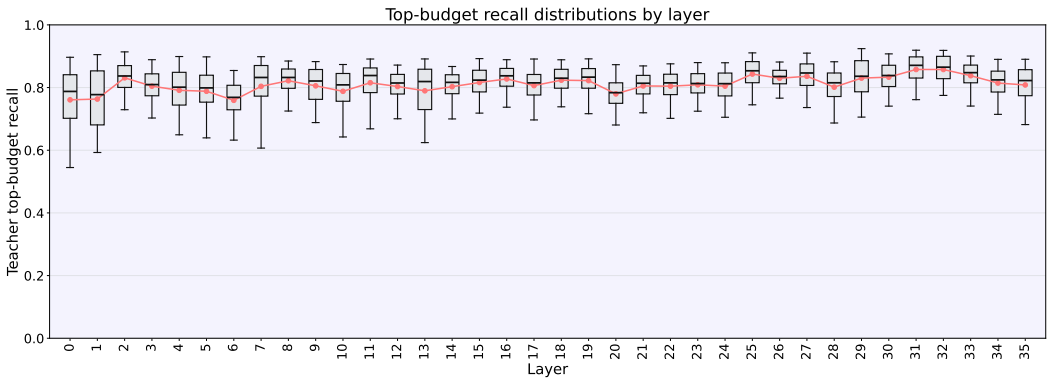


Figure 7: **Alignment with the future-attention oracle.** We plot the top-k retention recall distribution over heads for all layers of QWEN3-4B at 75% KV-cache compression. The metric measures what fraction of the future-attention retention oracle's top-k tokens are also kept by the learned KVPOPscorer under the same cache budget. Higher values indicate better agreement with the oracle retention policy.

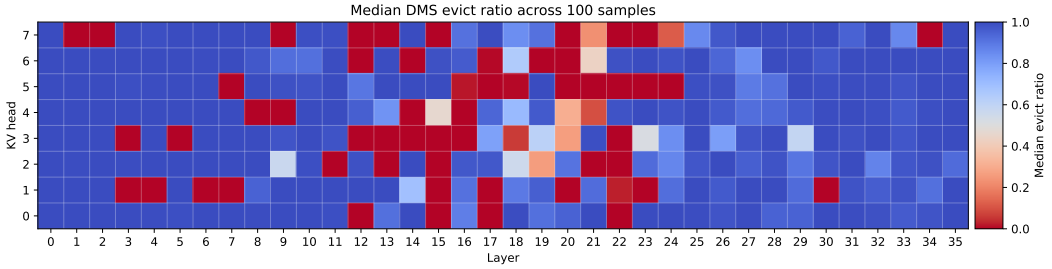


Figure 8: **DMS evict ratio.** We compute the median eviction ratio over 100 randomly sampled sequences for QWEN3-4B with DMS at a 75% compression ratio. A value of 0 indicates dense/full attention, whereas a value of 1 indicates that attention is restricted to the local sliding window. DMS exhibits highly heterogeneous eviction patterns: some heads use the full long-range token budget and behave close to dense-attention heads, while many heads—especially in early and late layers—collapse to sliding-window-only attention.

reasoning trace, while evicting many intermediate computation steps and purely numerical tokens once they have served their local role. This behavior is especially visible in later layers, where retained tokens tend to cluster around symbolic expressions, operation words, discourse markers, and other tokens that connect or summarize parts of the derivation. These examples provide qualitative observations consistent with a content-dependent eviction policy that preferentially preserves tokens useful for interpreting the evolving solution, rather than treating all operands and intermediate values equally.

H DMS SPARSITY PATTERNS

Dynamic Memory Sparsification (DMS) learns data-dependent eviction gates and therefore allows the effective cache allocation to vary across heads and layers (Łańcucki et al., 2025). To understand how a retrofit model uses this dynamic allocation in practice, we train DMS under the same 75% compression setting as KVPOPand analyze the resulting attention patterns for QWEN3-4B. Specifically, we compute the median eviction ratio over 100 randomly sampled sequences, where 0 indicates dense/full attention and 1 indicates that attention is restricted to the local sliding window.

Shown in Figure 8, DMS exhibits highly heterogeneous eviction patterns. A small subset of heads uses nearly the full long-range token budget and behaves close to dense attention, while many other heads—especially in early and late layers—collapse to sliding-window-only attention. This suggests

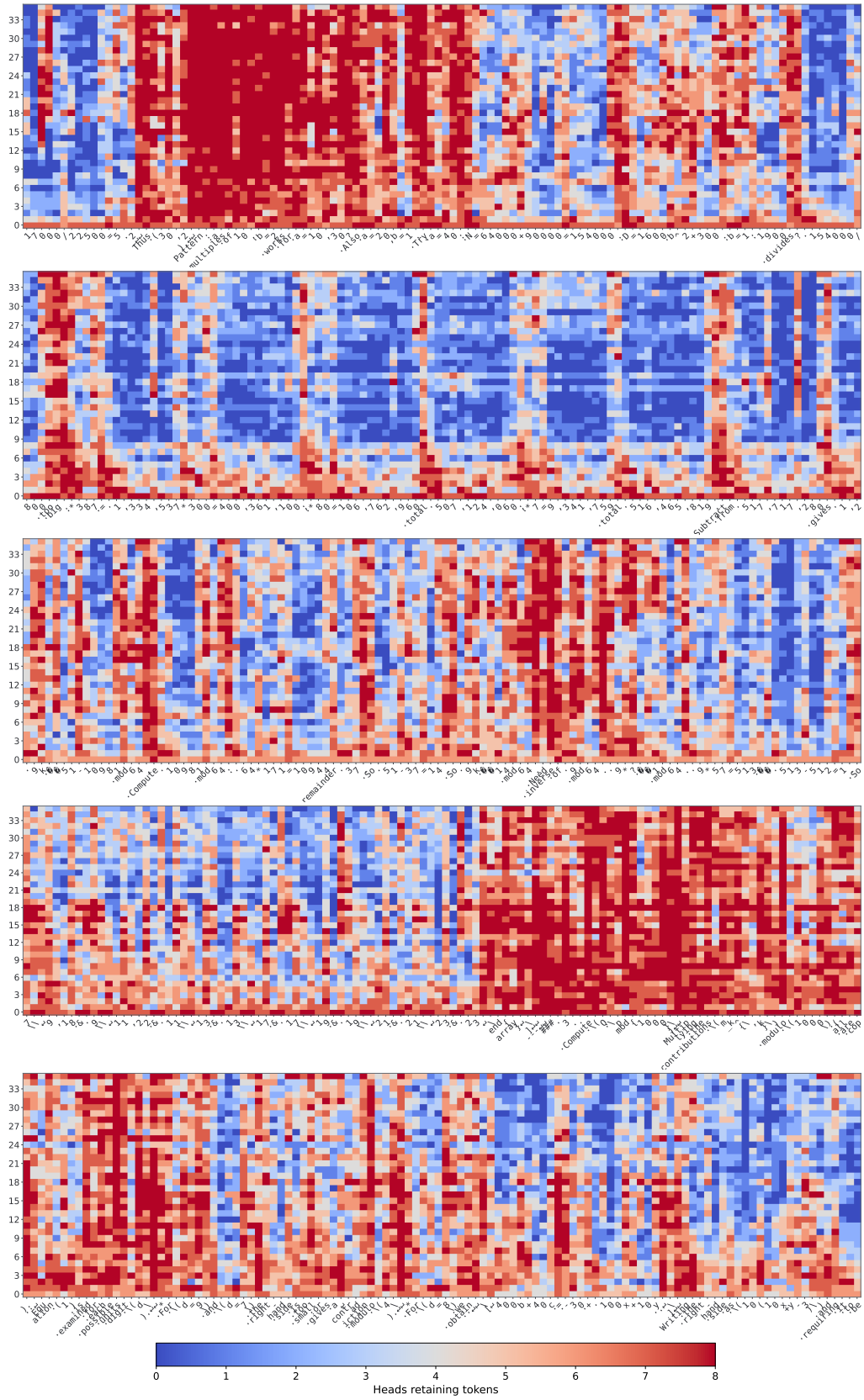


Figure 9: **KVPOPeviction patterns** (QWEN3-4B, CR=75%, final 112 tokens of five sampled sequences). Color shows how many heads retained each token (x-axis) per layer (y-axis).

a winner-takes-all allocation dynamic, where the available long-range budget is concentrated in a few heads rather than distributed evenly across the model. This behavior contrasts with KVPOP, where each head receives a fixed long-range token allocation by construction, yielding homogeneous and explicitly controlled sparse-attention budget.

I DELAYED SCORING IMPROVES STATEFUL EVICTION



Figure 10: **Effect of delayed scoring.** Next-token prediction accuracy on a held-out split of the retrofitting dataset with and without delayed scoring.

We ablate the effect of delayed scoring in Figure 10, comparing an MLSTM scorer with and without delayed readout. Here, *token accuracy* denotes next-token prediction accuracy (i.e., the fraction of ground-truth language model targets predicted correctly) evaluated on a held-out split of the Nemotron-Math v2 retrofitting dataset; it measures language modeling quality on held-out data under sparse attention rather than the accuracy of individual eviction decisions or downstream task scores. After 2,000 steps, delayed MLSTM scoring achieves a 0.2-point increase in token accuracy over immediate scoring.

Delayed readout allows a memory-based scorer to integrate additional context before the eviction decision: rather than committing when a token is first cached, delayed scoring leverages $w - 1$ steps of accumulated near-future evidence while the token remains in the protected window before competing for long-range admission.

J KVPOPPSEUDOCODE

The pseudocode summarizes the KVPOPforward pass used for training: each KV entry receives a learned retention score, and sparse attention keeps sinks, the protected recent window, and the highest-ranked long-range entries under a running prefix top-k rule. During training, the same sparse attention pass also provides the log-normalizers used to estimate each key's future attention mass by a transposed FLEXATTENTION call. This self-targeted signal trains the scorer to preserve keys that continue to receive probability mass after they leave the protected window.

Algorithm 3 KVPOP in PyTorch-style pseudocode.

```
def kvpop_attention(Q_rope, K_pre, K_rope, V, scorer, cfg, training):
    # Score each KV entry from detached inputs (no grad to network). MLP scorer and
    # mLSTM writes/gates use pre-RoPE K_pre; the delayed mLSTM read uses post-RoPE
    # K_rope with the state after t + w - 1 (d = w - 1), before t is eligible at t + w.
    r = scorer(K_pre.detach(), K_rope.detach(), V.detach(), delay=cfg.window - 1)

    # Prefix top-k over decayed KV scores.
    rank, cutoff = running_prefix_topk(
        r,
        sinks=cfg.sinks,
        window=cfg.window,
        topk=cfg.topk,
        decay=cfg.decay,
        decay_step=cfg.decay_step,
    )

    def sparse_mask(b, qh, q, t):
        kh = kv_head(qh)

        sink = t < cfg.sinks
        recent = q - cfg.window < t <= q
        eligible = cfg.sinks <= t <= q - cfg.window
        selected = rank[b, kh, t] <= cutoff[b, kh, q]

        return t <= q and (sink or recent or (eligible and selected))

    # Sparse student attention.
    Y, sparse_lse = flex_attention(
        Q_rope, K_rope, V,
        block_mask=create_block_mask(sparse_mask),
        return_lse=True,
    )

    if not training:
        return Y

    # Reuse the sparse-attention denominators to estimate each key's
    # future attention mass under the current sparse policy.
    def future_mask(b, qh, t, q):
        return q >= t + cfg.window

    def logprob_score(score, b, qh, t, q):
        return score - sparse_lse[b, qh, q]

    _, log_mass = flex_attention(
        repeat_kv_heads(K_rope), # original keys become queries
        Q_rope,                  # original queries become keys
        zeros_like(Q_rope),
        score_mod=logprob_score,
        block_mask=create_block_mask(future_mask),
        return_lse=True,
    )

    target = log_mass[..., :-cfg.window]
    target = target - log_num_future_queries_per_key(cfg.window)
    target = logaddexp(target, log(cfg.target_eps))  # log(m + eps), eps = 1e-6
    target = aggregate_query_heads(target, mode=cfg.group_agg)
    target = target.detach()  # target pathway does not backpropagate into
     representations

    loss = prefix_topk_boundary_loss(r, target, cfg)

    return Y, loss
```