MERGE THEN DROP, ADAPTIVE KV CACHE COMPACTION FOR LONG-CONTEXT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

As Large Language Models (LLMs) generate increasingly long reasoning traces, the Key-Value (KV) cache grows linearly and becomes a memory bottleneck. Learned KV Cache compactors typically assign the same budget to every layer and head, even though heads differ in how much context they need. We propose an online KV cache compactor that jointly learns the compressed representation and its per-head budget. The compactor produces an overcomplete set of merged candidate latents, from which differentiable Gumbel-Sigmoid gates decide which to retain under a global memory budget. Although gate allocation is fixed at inference, the compactor learns to route the relevant reasoning context into the surviving latents. Because backpropagating through the full sequential objective incurs an intractable memory cost for long contexts, we introduce an unbiased dual-sampling gradient estimator that retains only one compactor graph per trajectory. We evaluate on Qwen3-1.7B and 8B at compression ratios up to 16×. On 8B, our method lies on the accuracy-memory and accuracy-latency Pareto frontier for math reasoning, improves AIME Avg@4 by up to 18.3 points over a uniform-budget compactor and matches a state-of-the-art learned eviction baseline at about 350 MiB of peak KV cache.

1 INTRODUCTION

The shift towards test-time scaling (Muennighoff et al., 2025; Ke et al., 2025) and agentic workflows (Erdogan et al., 2025; Team et al., 2026) has made long Chain-of-Thought (CoT) reasoning standard practice for complex tasks. However, the standard transformer architecture dictates a linear growth of the KV cache. This imposes severe memory bottlenecks and latency degradation during extended generation trajectories, especially in constrained compute settings (Liu et al., 2025). This bottleneck necessitates efficient KV cache management (DeepSeek-AI, 2024), with online compaction emerging as a highly promising direction. Online KV cache compaction aims to dynamically compress context history as new tokens are generated without stalling decoding. While current online approaches primarily rely on heuristic token-dropping (Zhang et al., 2023; Li et al., 2024; Oren et al., 2024; Łańcucki et al., 2025) or token-merging (Nawrot et al., 2024), directly learning a latent KV cache transformation generalizes both approaches and enables a strictly more expressive paradigm.

Existing learnable soft KV compactors (O'Neill et al., 2026; Eyuboglu et al., 2026; Zweiger et al., 2026) are designed for the offline compression of static documents. They evaluate the entire context globally by relying on generated self-study question and allocates uniform compression ratios across all layers and heads, making them computationally expensive and suboptimal for continuous online generation. Conversely, recent works that train models to compress their own reasoning outputs during generation (Kontonis et al., 2026; Yan et al., 2025; Yang et al., 2026) rely on text-based summarization rather than latent cache transformation. These methods incur massive annotation and training costs, severe latency overheads by repeatedly generating and prefilling summaries, and require full-model retraining, risking catastrophic forgetting on general tasks.

Building on the observation that learned key representations act as stable, highly efficient routers (Diaz, 2025), we introduce an online compactor that combines high-fidelity token merging with learnable token eviction. Using a STILL-based architecture (O'Neill et al., 2026), we generate an overcomplete set of candidate latents that exceeds the target memory budget to preserve routing fidelity. We then apply a differentiable gating mechanism (Bengio et al., 2013; Jang et al., 2017)

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.

to drop less important tokens. While existing eviction methods simply select which original token KV entries to retain, our approach jointly optimizes the compactor and gates. This allows the model to actively merge features and heterogeneously allocate latent budgets across different layers and attention heads (Feng et al., 2025; Yang et al., 2024), steadily transitioning from a dense merged representation to a sparse, budget-constrained cache. Although the gating mask is static during inference, the compactor's cross-attention remains highly dynamic as it jointly learns how to route features specifically into the latent slots that survive the budget constraint. Through our unbiased sampled training strategy, we are able to reduce the memory cost of our sequential training objective and achieve stable training throughout.

We validate our method on math and coding reasoning benchmarks, including AIME 26, HMMT-Feb 26, and LiveCodeBench v6 (Jain et al., 2024). Our method is consistently at the Pareto frontier across peak KV memory, memory reads, and latency on mathematical reasoning. At high compression ratios, our method prevents the degradation seen in static compactors, yielding absolute accuracy improvements of +18.3 and +15.8 points on AIME at 12x and 16x compression, respectively. Furthermore, when constrained to a physical KV-cache memory footprint of $\sim$350 MiB, our compactor operates at a stricter 12x target yet matches the performance of state-of-the-art 8x learnable eviction baselines on AIME (61.7 vs. 62.5) and HMMT (34.8 vs. 34.1), while maintaining a strong generalization floor on out-of-distribution coding tasks.

Our main contributions are threefold:

1. We formulate online KV cache compaction as joint optimization of overcomplete latent representation and discrete per-head budget allocation, as well as a memory-efficient sequential sampled training procedure.
2. We demonstrate that our objective, training method, and learned routing architecture remain robust under repeated online compactions, achieving higher accuracy than static STILL on long mathematical CoT tasks.
3. We provide a fully functional systems implementation, featuring a head-specific paged KV cache and a custom Triton decode-attention kernel that actualizes physical memory savings with negligible fragmentation.

2 Related work

Soft token compression. Soft compression projects context into newly synthesized representations. Early methods compress text chunks into summary vectors: AutoCompressor (Chevalier et al., 2023) generates soft prompts for subsequent segments, while GistToken (Mu et al., 2023) condenses instructions into fixed-length prefixes. Other methods merge redundant tokens; ToMe (Bolya et al., 2023) utilizes bipartite matching, whereas DMC (Nawrot et al., 2024) retrofits the LLM to learn an online token-merging policy with variable compression rates. More recently, methods compress directly into the KV space. Cartridges (Eyuboglu et al., 2026) optimizes shareable compact caches via distillation on questions generated from self-study, while Attention Matching (Zweiger et al., 2026) bypasses this training overhead by numerically matching attention weights. STILL (O'Neill et al., 2026) accelerates this by training a lightweight compactor to synthesize a compact cache in a single forward pass. Our work builds upon STILL, adapting it for repeated online compaction and jointly learn to allocate token budgets across layers and route the representation into the surviving latent slots and heads.

Reasoning state compression. Recent methods address the context expansion of long reasoning by teaching models to restate their intermediate states as textual summaries. Approaches like InftyThink (Yan et al., 2025) and Accordion-Thinking (Yang et al., 2026) explicitly train the model to generate periodic summaries, discarding prior detailed reasoning to maintain a bounded context profile. Similarly, Memento (Kontonis et al., 2026) and LightThinker (Zhang et al., 2025) compress semantic blocks into short text mementos or condensed gist tokens using specialized attention masks, while Markovian Thinker (Aghajohari et al., 2026) periodically resets the context to force the policy to carry over essential state via a textual suffix. While effective at reducing peak context, these methods rely on generating additional summary tokens and require extensive full-model retraining that alters the underlying reasoning format. In contrast, our method keeps this chunk-wise schedule but compresses completed chunks directly into expressive latent KV states. By circumventing the need to re-verbalize

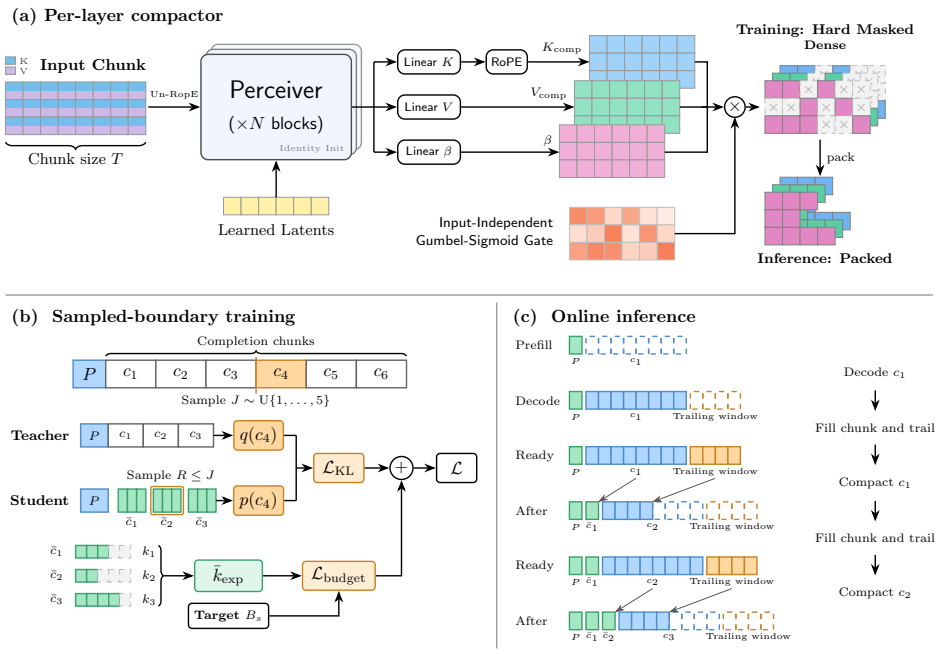

Figure 1: **Overview. (a) Architecture.** Layer-wise compactor compresses KV chunks into compressed keys, values, and β, and jointly learns eviction via a Gumbel-Sigmoid gate. **(b) Training.** Gradients flow through chunk R within a sampled boundary J. The student matches the teacher's distribution through $\mathcal{L}_{\text{KL}}$, while $\mathcal{L}_{\text{budget}}$ guides the average retained budget $\bar{k}_{\text{exp}}$ to match the target B_s. **(c) Inference.** Inference alternates with decoding and compaction. Each completed chunk c_i is replaced by its compact representation $\bar{c}_i$ and packed after the previously compacted chunks

the retained memory, we preserve the model's original continuous reasoning trajectory and avoid the massive annotation and training costs associated with explicit summarization.

3 Methodology

Prior work demonstrates that optimal KV cache compression ratios are highly heterogeneous; certain layers and attention heads process significantly more critical information than others (Yang et al., 2024; Feng et al., 2025; Zweiger et al., 2026). Consequently, enforcing a strictly uniform compression budget across the entire model is inherently suboptimal. This motivates a learned routing and pruning approach to context compaction, which jointly optimizes both the latent representations and the per-head budget allocation.

Following previous work (Inhar et al., 2026), we build upon STILL (O'Neill et al., 2026), which utilizes a lightweight Perceiver architecture (Jaegle et al., 2021) to produce a compacted KV cache (C_k, C_v) and a per-token attention bias β that modifies the standard attention computation to $\text{softmax}(QK^\top + \beta)V$ in a single forward pass. Rather than bottlenecking the Perceiver to a fixed, restrictive capacity, we configure it to output a larger candidate set of T_{init} latent tokens. By equipping these candidates with a simple learnable gating mechanism (Figure 1), we create an end-to-end learnable compactor that effectively functions as a hybrid between cross-attention token merging and discrete gates token dropping.

3.1 Problem Definition

Let π_θ be a pretrained autoregressive LLM with L layers and H KV heads. For a reasoning trajectory X, π_θ maintains a Key-Value (KV) cache $\mathcal{K}_{\text{full}} = \{(K^{(l,h)}, V^{(l,h)})\}_{l=1,h=1}^{L,H}$ whose size grows linearly with the sequence length. Online compaction compresses newly generated cache element either continuously or in chunks. Chunk based online compaction partitions this cache into consecutively growing chunks $\mathcal{K}_1, \ldots, \mathcal{K}_M$. Each newly completed chunk is compressed and appended to the previous compressed cache as generation proceeds.

Static compactors such as STILL (O'Neill et al., 2026) apply a layer-wise transformation $f^l_{\phi_l} : \mathbb{R}^{T\times H\times d\times 2} \to \mathbb{R}^{t\times H\times d\times 2}$ to a chunk of length T to compress it into exactly $t \ll T$ latent KV pairs per head, which imposes the same compression ratio on every layer and head, despite their differing importance to the model.

We instead formulate online compaction as joint optimization over representation learning and budget allocation. For each layer l, a compactor first maps chunk c to a high-capacity set of T_{init} candidate latent KV pairs, $g^l_{\phi_l} : \mathbb{R}^{T\times H\times d\times 2} \to \mathbb{R}^{T_{\text{init}}\times H\times d\times 2}$. Each candidate is then associated with a binary gate $m^{(l,h)}_{\alpha_l}(\mathcal{K}^{(l)}_c) \in \{0,1\}^{T_{\text{init}}}$, where α_l denotes the gating parameters for layer l. This enables different learned compression ratios between layers and heads while sharing a global cache budget.

For chunk c, layer l, and head h, the compressed chunk $\widehat{\mathcal{K}}^{(l,h)}_c$ and physically retained tokens $k_{c,l,h}$ can be written as

$$\widehat{\mathcal{K}}^{(l,h)}_c = \left\{ \left[g^l_{\phi_l}(\mathcal{K}^{(l)}_c) \right]^{(h)}_r \;:\; m^{(l,h)}_{c,r} = 1 \right\}, \qquad k_{c,l,h} = \sum_{r=1}^{T_{\text{init}}} m^{(l,h)}_{c,r}.$$

Let $p(i)$ denote the chunk containing token position i. During online inference, all chunks before $p(i)$ have been compressed, while the preceding tokens within the current chunk remain uncompressed. The cache used to predict token i can be written as

$$\mathcal{K}_{\text{online},<i} = \left(\bigoplus_{c=1}^{p(i)-1} \widehat{\mathcal{K}}_c \right) \oplus \mathcal{K}_{p(i),<i},$$

where $\oplus$ denotes concatenation. And for a trajectory containing M chunks, we can define the average retained budget per head and chunk as

$$\bar{k}(X) = \frac{1}{MLH} \sum_{c=1}^{M} \sum_{l=1}^{L} \sum_{h=1}^{H} k_{c,l,h}.$$

Our objective is to jointly learn both the compressed representations and their allocation across heads. With ϕ denoting the compactor parameters and α the gating parameters, we optimize compressed KV representations that best preserve the original model under a global average memory budget:

$$\begin{aligned} \min_{\phi,\alpha} \quad & \mathbb{E}_{X\sim\mathcal{D}} \left[\sum_i \mathcal{D}_{\text{KL}}\big(\pi_\theta(\cdot \mid \mathcal{K}_{\text{full},<i}) \,\|\, \pi_\theta(\cdot \mid \mathcal{K}_{\text{online},<i})\big) \right] \\ \text{s.t.} \quad & \mathbb{E}_{X\sim\mathcal{D}} \left[\bar{k}(X) \right] \le B. \end{aligned}$$

3.2 Training

Objective. We freeze the pretrained model π_θ and optimize only the compactor and gating parameters through chunk-by-chunk autoregressive self-distillation. Suppose a reasoning trajectory contains M equally sized chunk, with $M-1$ valid target boundaries, where at boundary j the compactor has compacted the preceeding j chunks ($c_{\le j}$). The distillation objective $\mathcal{L}_{\text{KL}}$ minimizes the forward Kullback-Leibler (KL) divergence between the teacher (using the uncompressed cache) and the student (using the compacted prefix) over all boundaries:

$$\mathcal{L}_j = \mathcal{D}_{\text{KL}}\left(\pi_\theta(c_{j+1} \mid c_{\le j}) \,\|\, \pi_\theta(c_{j+1} \mid \widehat{c}_{\le j})\right), \qquad \mathcal{L}_{\text{KL}} = \frac{1}{M-1} \sum_{j=1}^{M} \mathcal{L}_j.$$

We utilize the straight-through Gumbel-Sigmoid trick to enable continuous relaxation of binary variables and make the discrete budget allocation differentiable. During training, for candidate r of chunk c, layer l, and head h we draw $u^{(l,h)}_{c,r} \sim \text{U}([0,1])$ and compute the Logistic noise[1]

$$\eta^{(l,h)}_{c,r} = \log u^{(l,h)}_{c,r} - \log\left(1 - u^{(l,h)}_{c,r}\right).$$

[1]The difference of two standard Gumbel variables forms a Logistic distribution.

We utilize an input independent gate logits per candidate latent position, $a_r^{(l,h)}$. The relaxed and hard mask values are then computed as

$$\widetilde{m}_{c,r}^{(l,h)} = \sigma\left(\frac{a_r^{(l,h)} + \eta_{c,r}^{(l,h)}}{\tau}\right), \qquad m_{c,r}^{(l,h)} = \mathbf{1}\left[\widetilde{m}_{c,r}^{(l,h)} > 0.5\right].$$

During training, we use the hard gate $m_{c,r}^{(l,h)}$ in the forward pass and the straight-through gradient of $\widetilde{m}_{c,r}^{(l,h)}$ in the backward pass. The expected number of retained tokens per head is [2]

$$\bar{k}_{\text{exp}} = \frac{1}{MLH} \sum_{c=1}^{M} \sum_{l=1}^{L} \sum_{h=1}^{H} \sum_{r=1}^{T_{\text{init}}} \sigma\left(a_r^{(l,h)}\right).$$

We apply Huber loss to guide the compactor towards a target budget B_s, giving $\mathcal{L}_{\text{budget}} = \text{Huber}(\bar{k}_{\text{exp}}, B_s)$. The final full objective is a weighted sum $\mathcal{L} = \mathcal{L}_{\text{KL}} + \lambda \mathcal{L}_{\text{budget}}$. After a short warmup, we linearly anneal the target budget B_s from the initial latent count T_{init} down to the desired inference budget B. Importantly, B controls the expected target rather than an exact inference token count. At inference, the realized budget may differ from B (Appendix H) as the noise is removed and candidates are simply retained whenever $a_{c,r}^{(l,h)} > 0$.

Chunk sampling. Naively unrolling this full objective $\mathcal{L}$ in a long sequence is fully intractable as it requires retaining multiple student-model and compactor autograd graphs simultaneously; the number of graphs grows linearly with sequence length, and training a compactor for a 1.7B model with batch size 1 on a 16k length sequence needs upwards of 100 GiB of memory.

Generalizing on Zweiger et al. (2026) mass-matched formulation, Appendix C.1 formulates an attention-level composability result for independently compressed chunks. For a fixed query, independently compressed chunks are composable under attention and the error bound does not depend on the number of compacted chunks (Inhar, 2026). While not an end-to-end guarantee, this demonstrates that individual chunk compression objective remain effective even after multiple chunks are concatenated. We can then optimize the compression error for each chunk rather than explicitly training for composability. This argument motivates optimizing our chunk-level compactor via sampling rather than fully unrolling gradients through every boundary simultaneously.

To efficiently estimate the gradient of the full objective, we first sample a single boundary $J \sim \text{U}(\{1, \ldots, M-1\})$. The expectation over J recovers the mean and thus sampling one boundary provides a strictly unbiased estimate of the distillation objective, i.e. $\mathbb{E}_J[\nabla_{\phi,\alpha} \mathcal{L}_J] = \nabla_{\phi,\alpha} \mathcal{L}_{\text{KL}}$.

However, predicting chunk c_{J+1} at the sampled boundary J still involves J compressed prefix chunks $(\widehat{c}_1, \ldots, \widehat{c}_J)$. Naive backpropagation would therefore still require retaining J separate compactor graphs. Let $\psi = (\phi, \alpha)$ denote the combined compactor parameters. By the chain rule, the gradient of the loss at boundary J is the sum of contributions through each prefix chunk b:

$$G_{J,b} = \left(\frac{\partial \widehat{c}_b}{\partial \psi}\right)^{\top} \nabla_{\widehat{c}_b} \mathcal{L}_J, \qquad \nabla_\psi \mathcal{L}_J = \sum_{b=1}^{J} G_{J,b}.$$

To resolve this memory bottleneck, we sample just one gradient-bearing prefix chunk $R \mid J \sim \text{Uniform}(\{1, \ldots, J\})$. We retain the compactor graph only for chunk R, and detach the graphs for all other compressed chunks. Scaling the sampled gradient contribution by J yields an unbiased estimate of the local boundary gradient: $\mathbb{E}_{R|J}\left[J \cdot G_{J,R}\right] = \sum_{b=1}^{J} G_{J,b} = \nabla_\psi \mathcal{L}_J$.

Combining the two sampling stages, the final gradient estimator is unbiased with respect to the full distillation objective:

$$\mathbb{E}_{J,R}\left[J \cdot G_{J,R}\right] = \nabla_\psi \mathcal{L}_{\text{KL}}.$$

And crucially, all compressed chunks remain present in the forward pass, preserving their interactions through the attention mechanism. This estimator is unbiased for the full gradient w.r.t. the compactor parameters. Although we do not formalize it separately, the same sampling argument applies to the straight-through pseudo-gradient for the gate parameters. This sampling requires only one student-model graph and one compactor graph in memory per trajectory. We analyze the extra variance introduced by this sampling method in Appendix C.2.

[2] The cumulative distribution function of the standard Logistic distribution is the sigmoid function

4 EXPERIMENTS, RESULTS AND DISCUSSIONS

4.1 EXPERIMENTAL SETUP

Data and setup. We evaluate Qwen3 1.7B and 8B (Yang et al., 2025), utilizing the modified canonical STILL architecture (Inhar, 2026). For training, we train on Open-R1 reasoning traces (Hugging Face, 2025), up to 16,384 tokens, using AdamW (Loshchilov & Hutter, 2019) and a batch size of 16 on A100 GPUs, linearly annealing the target token budget over the first 1500 steps. We train our model to compress 512-token chunks at 8x (64 Tok), 12x (43 Tok), and 16x (32 Tok) compression ratios (CR). To ensure a sufficiently expressive merging representation, we initialize the overcomplete latents to $T_{\text{init}} = 256$. During inference, we only compact the reasoning trajectories leaving the prompts and answer tokens uncompacted to mimic what a long multi turn system would look like. We compact the latest chunk once the chunk and trailing window are filled. Training uses FlexAttention (Dong et al., 2025) to support attention bias natively and custom Triton (Tillet et al., 2019) kernels for inference to realize the physical memory saving (details in Appendix D and E).

Baselines. We compare against uncompacted base LLMs and four compacted baselines: Ungated (Vanilla) STILL (O'Neill et al., 2026), Attention Matching (AM) (Zweiger et al., 2026), TOVA (Oren et al., 2024), and DMS (Łańcucki et al., 2025). Vanilla STILL serves as our uniform-capacity control; it utilizes our identical setup (backbone, training data/objective, inference policy & backend) but omitting the overcomplete latent initialization and learned budget gates, directly emitting B latents per KV head. For DMS, we retrofitted the model via the NVIDIA ModelOpt library to do 8x CR with 512 trailing window following the original specification of the paper. We set a maximum of 8x CR as their paper notes that higher CR will collapse the KL loss. For AM, we use an append-only schedule similar to Ours with repeat prefill and large 512 uncompacted tokens. For TOVA, we utilizes NVIDIA KVPress library and set the static state size as maximum tokens present in generation divided by compression ratio. We evaluate Vanilla STILL, TOVA, and AM at 8x, 12x, and 16x CR. All external baselines use reference implementations (Appendix F)

Goal and metrics. To evaluate the stability of compression on hyper-scaled reasoning trajectories, we set maximum token limits to 38k for math and 32k for coding, as evaluating on completions with length 16k or below would inadequately test long-context robustness. We use Dekoninck et al. (2026) AIME26, HMMT-Feb26, and LiveCodeBench v6 as our main evaluation datasets. We report Avg@4 to assess compression robustness across the model's broader completion distribution rather than a single trajectory. We additionally report efficiency metrics which include the average compaction time, generation throughput, accumulated memory reads, and Peak KV averaged across all samples. Peak KV measures the actual physical size of KV blocks actively occupied during generation. Because our method and DMS true CR differs from the nominal targets, as well as the how backend implementations (Ours, DMS, TOVA) pre-allocate memory pools akin to vLLM, Peak KV enables a fair, memory-matched comparison for the methods. Evaluation details and sampling parameters are detailed on Appendix E.

4.2 ONLINE COMPACTION

We evaluate online compaction across Qwen3-1.7B and 8B. We observe high variance and noisy result for smaller model under increased compression in all methods, and thus we focus our primary analysis on the 8B model to establish clear performance trends. Our full table and Pareto result can be seen in Appendix K (Table 11 and Figure 8).

As shown in Figure 2, our method is consistently at the Pareto frontier across peak KV memory, accumulated KV reads, and latency on the math benchmarks. Our method matches DMS in memory at lower per-step latency and outperforms the remaining baselines. Furthermore, we exceed DMS by 2.3 points in HMMT on the same 8x CR at a slight memory cost. We outperform the controlled Vanilla STILL baseline, where our learned routing yields absolute improvements of +18.3 and +15.8 points on AIME at 12x and 16x compression respectively (Table 13). This empirically measures the gain of replacing a uniform latent bottleneck with jointly learned candidate routing and allocation.

The structural advantages of this routing of overcomplete latents are directly visible in the learned head budgets (Appendix H), our gates learn to heavily concentrate token budgets in the middle KV heads while aggressively sparsifying the early and late heads. This learned allocation enables stricter compression while still maintaining better performance. In the 16x setting, the 8B models easily

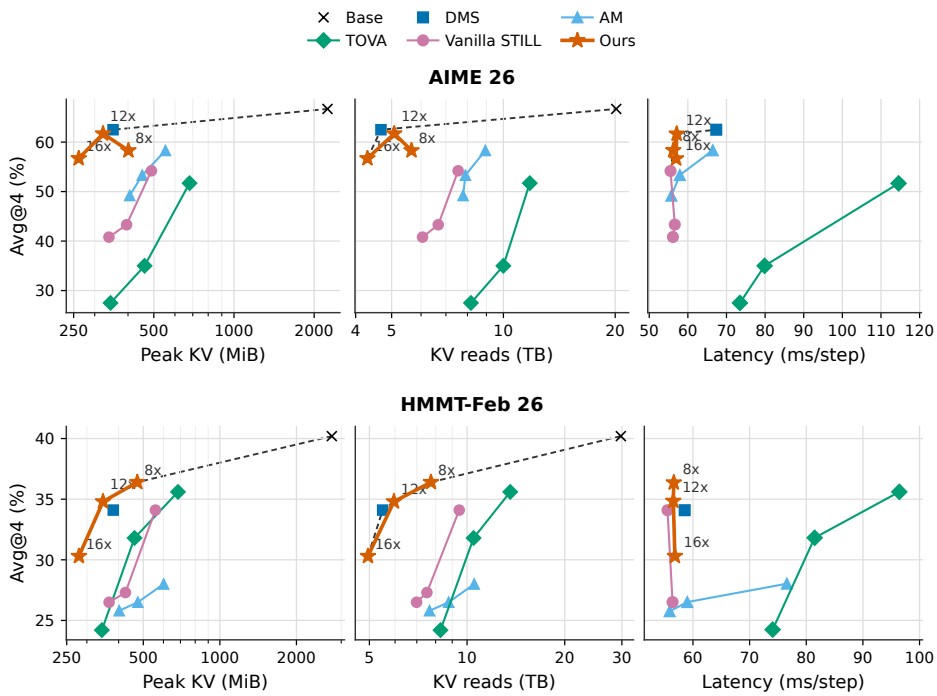

Figure 2: Memory metrics and per-step latency vs accuracy for Qwen3-8B for all methods. Upper left is the desired region. Dashed line represents the Pareto frontier. All metrics are averaged across the whole generations

drop well below the budget to and average of 23.7 retained tokens, while still maintaining better performance than Vanilla STILL that retain the full 32 tokens. This confirms that our jointly learned capacity allocation adaptively scales with the base model's inherent reasoning capabilities. We also show the generalization of the composability of our training method, achieving stable compression up to 38k tokens when only trained on data up to 16k in length.

		AIME 26			HMMT-Feb 26		
Method	**CR**	**Avg@4 ↑**	**Peak KV (MiB) ↓**	**Reads (TB) ↓**	**Avg@4 ↑**	**Peak KV (MiB) ↓**	**Reads (TB) ↓**
Base	–	66.7	2245	20.13	40.2	2755	29.78
DMS	8×	**62.5**	351	**4.68**	<u>34.1</u>	381	**5.49**
AM	16×	49.2	406	7.79	25.8	402	7.66
TOVA	16×	27.5	344	8.18	24.2	**344**	8.28
Vanilla STILL	16×	40.8	<u>339</u>	6.06	26.5	367	6.99
Ours	12×	<u>61.7</u>	**322**	<u>5.08</u>	**34.8**	<u>347</u>	<u>5.95</u>

Table 1: Matched-memory Qwen3-8B comparison on math. For each method, we select the operating point closest to roughly 350 MiB of measured peak KV. Bold and underline mark the best and second-best compressed methods. All metrics are the average across the whole generations

Peak KV varies with the same nominal CR because the total generation lengths for each methods differs. As detailed in Appendix I, we observe that soft-token compactors tends to generate longer reasoning than hard-eviction models. At 8x CR, our method generates 18% more tokens than DMS on AIME and 29% more on HMMT, and it reaches the generation cap more frequently. Our primary accuracy and efficiency results includes all completions. When evaluating on sequences that terminates on EOS, we observe a 7.4 points accuracy increase in AIME and 5.4 on HMMT. The observed pattern is consistent with delayed termination under soft compaction, and despite the disadvantage of the unfinished completions, our method still performs remarkably well.

Table 1 compares points near a matched peak KV payload of 350 MiB to match DMS 8x CR. Because we compact only reasoning tokens, our compactor operates at a stricter 12x compared to DMS. Despite this, we match DMS's performance on AIME (61.7 vs. 62.5) and on HMMT (34.8 vs. 34.1), while operating with lower peak memory. In contrast, Vanilla STILL completely collapses at this compression ratio (40.8 and 26.5, respectively), showing our method direct advantage on high CR.

Model	Base	DMS	AM	TOVA	Ours
Qwen3-1.7B↑	29.9	<u>26.3</u>	**30.6**	24.9	25.6
Qwen3-8B↑	47.4	45.1	**46.6**	39.6	<u>45.3</u>

Table 2: LCB v6 results at 8x CR

Method	AIME 26↑	HMMT-Feb 26 ↑
Vanilla STILL	18.01	18.02
DMS	14.85	<u>17.08</u>
AM	<u>15.05</u>	13.06
TOVA	8.73	10.37
Ours	**17.78**	**17.66**

Table 3: End-to-end generation throughput (Tok/s) for Qwen3-8B at 8x CR over all generations.

On LCB v6 (Table 2), AM achieves the highest retention as its doing exact numerical matches on the completion. Our method and DMS perform similarly but fall slightly behind AM. This follows from both our training data, as we both train on Open-R1, making code generation a slightly out-of-distribution task. Statistical analysis (Table 12) confirms that the performance gap between our method and DMS on LCB is indistinguishable at the 95% confidence level. Our method retains the same generalization floor as state-of-the-art eviction methods, while operating at the Pareto frontier on targeted reasoning tasks.

Latency and throughput. We plot the latency vs generated token of our method on 8x and 16x compression setting and compare it to the base model (Figure 3). We can see that our method can generate longer sequences without going out of memory with a relatively flat latency. Vertical lines represent extra latency every 512 tokens due to the compaction, which includes gather, compactor forward, evict, and store of the chunks. Our compaction overhead in a 64 batch size setting averaged well below 2 seconds and only happens every 512 tokens, which are negligible for overall throughput. Next, we compare the end-to-end generation throughput of our method against the rest of the baselines on real benchmark completions on the same machine, GPU, and batch size. We define throughput as the total generated token divided by total generation time (sample-wise) to take into account the different overhead present in each methods excluding prefill. We find that Vanilla STILL achieves the highest throughput over the two benchmarks, which is expected due to the low overhead of a single forward pass per compaction. Our method is slightly behind (∼0.3 Tok/s slower) due to the gate and packing overhead, as well as the inefficiency of the scheduling of our custom inference kernel. However, our throughput is clearly faster compared to the rest of the baselines, with the closest being ∼2.7 and 0.6 Tok/s slower in AIME and HMMT respectively. This validates our systems implementation and shows that the compaction and packing overhead remains small relative to decoding.

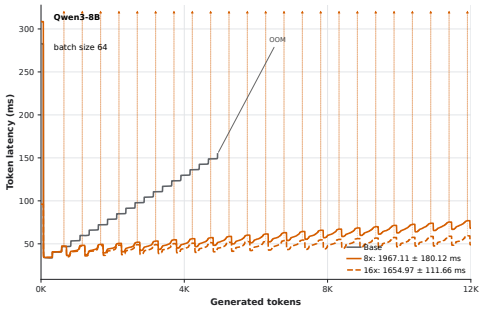

Figure 3: Latency benchmark for base vs Ours 8x and 16x. Number in legend refers to average latency directly after compaction. Vertical lines are truncated

4.3 Ablation

Soft masking and input-dependent gates. We justify our gating architecture by ablating against alternative design and training choices. In particular use static gating and we completely mask off dropped tokens during training (hard masking) to mimic the inference setting. Alternatively, we can equip each KV head with an input-dependent gate $\ell_{b,h,i} = \mathbf{w}_h^\top \mathbf{z}_{b,h,i} + b_h$ to control the eviction and use soft masking based on the logits during training so that dropped tokens can still contribute to $\mathcal{L}_{\text{KL}}$ (Łańcucki et al., 2025; Zhou et al., 2026). We train a Qwen3-1.7B compactor on the four combinations of these settings and find that Static-Hard performs the best, while the rest completely collapsed during training. Specifically, training with Static-Soft degrades inference performance

because the gradient from the KV-loss overpowers the gradient from the budget loss resulting in similar gate scores for the majority of tokens which collapses the number of tokens kept during inference to only ∼5. Furthermore, we found that input-dependent gates cannot reliably learn which tokens to attend to, which collapses the KL Loss during the dropping part of training. Inference result on all ablated methods can be seen on Table 8.

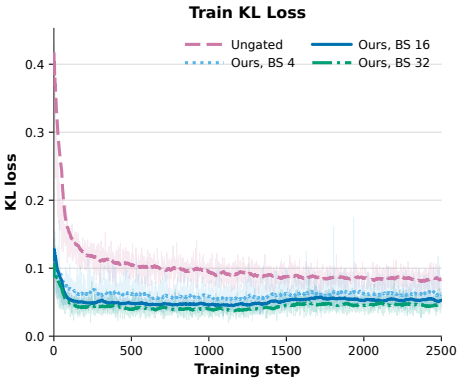

Figure 4: Optimization behaviour of Vanilla STILL control compared to Ours

Training stability of learned allocation. We analyze the optimization behaviour of our learned overcomplete initialization (Figure 4). We train a compactor for Qwen3-1.7B 8x CR with Vanilla STILL and Ours with 3 different batch sizes. Under the same training setup, Ours consistently attains lower $\mathcal{L}_{\mathrm{KL}}$ during training than Vanilla STILL. Furthermore, we can also see that Ours training loss remains stable during the token dropping part of training, only increasing slightly during the tail end of the 1500 step anneal schedule. Our sampled training produces high variance at batch size 4 with sub-par convergence, though still lower than Vanilla STILL, compared to batch sizes 16 and 32 which converge better. This follows from our sampling strategy, which was designed to enable a large batch size at the cost of extra added single lane variance. The gains from using this strategy outweigh the disadvantage of increased variance, as we are able to both fit large batch sizes and get a faster convergence due to lower time per training step.

Initialization and inference schedule. To determine optimal overcomplete initialization number, we sweep candidate capacities $T_{\mathrm{init}} \in \{64, 128, 192, 256\}$ for a Qwen3 1.7B compactor in 32-token target expected budget. We found that the performance is upper bounded by the number of latents generated as degradation due to token dropping in training is minimal. Performance plateaus above 4 times the desired capacity, but remains stable and able to prune excess tokens regardless of starting size (Table 9). We default every run to $T_{\mathrm{init}} = 256$ to maximize initial routing fidelity and standardize the candidate capacities across different CR. During online inference, delaying compaction for the most recently completed chunks improves performance as adjacent tokens rely heavily on local attention. However, because our compactor is trained using the immediate next chunk, using large uncompressed trailing windows ($\geq$ chunk size) during inference pushes the generation to be out-of-distribution. We evaluate a trained Qwen3 1.7B compactor on MATH-500 level 5 (Hendrycks et al., 2021) as a validation set, and sweep the minimum trailing window. As shown in Table 10, a 128-token trailing window achieves the highest Avg@4 with negligible impact on peak memory. We use these configuration throughout our experiments for both our method and Vanilla STILL.

5 Conclusion

In this work, we introduced an online KV cache compaction that jointly learns high-fidelity token merging with learnable token dropping. By extending STILL with a differentiable Gumbel-Sigmoid gating mechanism, our approach jointly learns compression with budgets allocation across different layers and heads under a global memory constraint. We propose a sequential objective to train this compactor by distillation and an unbiased dual-sampling strategy that reduces training memory footprint by bounding the number of retained computation graph. Extensive evaluations on Qwen3 model family demonstrate that our method consistently is at the Pareto frontier across peak memory, memory reads, and latency on mathematical reasoning tasks. Crucially, our jointly learned capacity allocation successfully preserves model behavior under high compression ratios (up to 16x), significantly outperforming static uniform compactors and matches state-of-the-art learnable eviction baselines at a matched physical memory footprint while having lower per-step latency. We also propose a paged attention backend implementation with a custom Triton decode-attention kernel that realizes physical memory savings while still maintaining better generation throughput than other baselines. These results suggest feasibility of adoption and significant potential for scaling to broader and more complex tasks in the future.

AI USE STATEMENT

We use AI to help with editing and restructuring of the paper, implementing and debugging codes for our method, and preparing and launching computational experiments. We did not use generative AI to generate datasets, formulate mathematical claims or proofs, determine research hypotheses and interpret our results. In particular, we use AI to help clarify our writing by restructuring and/or appending clarifying sentences, especially in Section 3 and Appendix. Furthermore, we place emphasis in using as little AI influence as possible in the main body of the manuscript to ensure readability (used mainly in methodology and result section), while allowing more freedom in editing the Appendix which benefits from extra verboseness. AI help in writing codes for our experiments were done with standard practice AI coding tools, with human-in-the-loop present in all steps to check and verify the generated codes. We note that while AI is used to help launch large compute jobs, all experiments are human supervised to ensure correctness and fairness, with logs from every experiments still stored. Finally, we use AI to generate additional feedback to help us iterate on the method and the manuscript itself. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The minimal implementation to understand and reproduce the result for our method, which consist of the compactor architecture, gating, model, inference backend, evaluation script, and the training script is attached as supplementary material for the submission. We plan to open source the codes upon further development.

REFERENCES

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. The markovian thinker: Architecture-agnostic linear scaling of reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3As6AQ9ELI.

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.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=JroZRaRw7Eu.

Zefan Cai, Wen Xiao, Hanshi Sun, cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. R-kv: Redundancy-aware kv cache compression for reasoning models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 60980–61005. Curran Associates, Inc., 2025. doi: 10.52202/085713-2038. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/57e0358aed314784cad14a26e1eba642-Paper-Conference.pdf.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. doi: 10.52202/068431-1189. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. doi: 10.48550/arXiv.2405.04434.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

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

Mauri Diaz. Learned structure in CARTRIDGES: Keys as shareable routers in self-studied representations. In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=8pWoQFW7pY.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flexattention: A programming model for generating fused attention variants. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/61a9278dfef5f871b5e472389f8d6fa1-Paper-Conference.pdf.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 15419–15462. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/erdogan25a.html.

Sabri Eyuboglu, Ryan Saul Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Ruoyu Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=0k5w8O0SNg.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 113152–113188. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a40ff56daab9f4808b1e18350c8a11ce-Paper-Conference.pdf.

Lukas Hauzenberger, Niklas Schmidinger, Anamaria-Roberta Hartl, David Stap, Thomas Schmied, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. Kvpop – key-value cache compression with predictive online pruning, 2026. URL https://arxiv.org/abs/2607.05061.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.

Izaaz Inhar. Agentic memory should localize compression. In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*, 2026. URL https://openreview.net/forum?id=ztmwHisqJ4.

Izaaz Inhar, Faeyza Rishad Ardi, Muhammad Reza Qorib, Alham Fikri Aji, and Farid Adilazuarda. Beyond merging and dropping: Adaptive online KV cache compaction. In *COLM Workshop on Context Beyond the Window: Persistent Knowledge in Language Models*, 2026. URL https://openreview.net/forum?id=rcR66RU5u1.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4651–4664. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jaegle21a.html.

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. *arXiv preprint arXiv:2403.07974*, 2024.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, PeiFeng Wang, silvio savarese, Caiming Xiong, and Shafiq Joty. A survey of frontiers in LLM reasoning: Inference scaling, learning to reason, and agentic systems. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=SlsZZ25InC. Survey Certification.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. Memento: Teaching llms to manage their own context, 2026. URL https://arxiv.org/abs/2604.09852.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. 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, 2023. Association for Computing Machinery. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Maria Ponti. Inference-time hyper-scaling with kv cache compression. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 9365–9397. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/0d781fa5f639bf2caf728a68e9678362-Paper-Conference.pdf.

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 A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970. Curran Associates, Inc., 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/28ab418242603e0f7323e54185d19bde-Paper-Conference.pdf.

Sicong Liu, Weiye Wu, Xiangrui Xu, Teng Li, Bowen Pang, Bin Guo, and Zhiwen Yu. Adaptive and resource-efficient agentic ai systems for mobile and embedded devices: A survey, 2025. URL https://arxiv.org/abs/2510.00078.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3d77c6dcc7f143aa2154e7f4d5e22d68-Paper-Conference.pdf.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 37396–37412. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/nawrot24a.html.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized kv cache compaction in a single forward pass, 2026. URL https://arxiv.org/abs/2606.07878.

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 Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men,

Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

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.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models. *CoRR*, abs/2503.06692, March 2025. URL https://doi.org/10.48550/arXiv.2503.06692.

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. URL https://arxiv.org/abs/2505.09388.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV cache compression for high-throughput LLM inference. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3258–3270, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.195. URL https://aclanthology.org/2024.findings-acl.195/.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Wenlei Shi, Yiwei Wang, Xiaodan Liang, and Jing Tang. Accordion-thinking: Self-regulated step summaries for efficient and readable llm reasoning, 2026. URL https://arxiv.org/abs/2602.03249.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. LightThinker: Thinking step-by-step compression. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13307–13328, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.673. URL https://aclanthology.org/2025.emnlp-main.673/.

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. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Enshuai Zhou, Yifan Hao, Chao Wang, Rui Zhang, Di Huang, Jiaming Guo, Xing Hu, Zidong Du, Qi Guo, and Yunji Chen. Lkv: End-to-end learning of head-wise budgets and token selection for llm kv cache eviction, 2026. URL https://arxiv.org/abs/2605.06676.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via Attention Matching, 2026. URL https://arxiv.org/abs/2602.16284.

A Further related works

Hard token compression. Hard token compression decreases memory usage by retaining a subset of the original tokens. Heuristic-based methods use fixed rules to estimate token importance for eviction: StreamingLLM (Xiao et al., 2024) retains initial sinks and the most recent tokens, H2O (Zhang et al., 2023) and TOVA (Oren et al., 2024) rely on attention scores, and SnapKV (Li et al., 2024) selects clustered high-attention positions. PyramidKV (Yang et al., 2024) and AdaKV (Feng et al., 2025) dynamically distribute this budget across layers and heads, while R-KV (Cai et al., 2025) tailors eviction specifically for long reasoning trajectories. Expected Attention (Devoto et al., 2025) instead estimates in closed form how future queries will behave, enabling training-free token pruning. Learnable methods fine-tune models to learn their own eviction policies, LookaheadKV (Ahn et al., 2026) predicts future context importance while DMS (Łańcucki et al., 2025) retrofits LLMs to delay eviction and implicitly preserve discarded information. LKV (Zhou et al., 2026) and KVpop (Hauzenberger et al., 2026) instead learn which original KV entries to retain akin to DMS. Despite their efficiency, these methods remain restricted to a strict subset of the original tokens. We note that LKV and KVpop were released during the development of this work and after our method and evaluation design had been established. We cite them for completeness but retain DMS as the primary experimental baseline.

Adaptive architecture and discretization. Our method's transition from dense, high-fidelity tokens to sparse representations relies on differentiable discrete gating. This builds on the Straight-Through Estimator (Bengio et al., 2013), which bypasses the zero-gradient problem of step functions, and Gumbel-Softmax (Jang et al., 2017), which provides a differentiable relaxation of categorical distributions. Our per-head budget allocation also draws from adaptive compute architectures: just as L_0 Regularization (Louizos et al., 2018) learns sparse networks using stochastic gates, Mixture-of-Depths (Raposo et al., 2024) dynamically allocates compute via capacity-constrained top-k routing.

We adopt these principles, using differentiable binary gates to learn how many compact tokens to retain per head while enforcing a global cache budget.

B LIMITATION AND FUTURE WORKS

Limitations. While our approach achieves robust hyper-scaled compression, our empirical validation is currently limited to the Qwen3 model family. Furthermore, due to the way on how KV compression is inherently done, memory and compute savings will decrease when applied to hybrid models that incorporate linear attention. The main weakness of our method is delayed EOS, with Ours showing high percentage of unfinished completions. Lastly naive input-dependent gates fail to learn reliably under our current training paradigm. This combines with our training that exclusively train on reasoning tokens will likely result in degradation of performance for recall heavy tasks for our current model.

Future work. We identify several promising directions for future research. First, developing a mechanism to train input-dependent gates concurrently with the compactor would restore dynamic eviction without dispersing the training signal. Furthermore, because our compactor optimize completion generation, it naturally facilitates faster reinforcement learning (RL); a proposed pipeline for joint RL training is detailed in Appendix G. Additionally, this architecture could be extended beyond long reasoning trajectories to more general tasks, including exploring its viability as a persistent memory representation for continual learning. Ultimately, training the compactor jointly with the base LLM from scratch could enable native compaction, bypassing the need for post-training adaptation altogether.

C THEORETICAL ANALYSIS

C.1 ATTENTION LEVEL COMPOSABILITY OF INDEPENDENTLY COMPACTED CHUNKS

To analyze the error bound of compressed chunks, we first define the standard attention components for a given query q. Let $Z_c(q)$ denote the partition function of an uncompressed chunk c, and let $N_c(q)$ be its unnormalized value-weighted sum. We denote their corresponding approximations after compression as $\widehat{Z}_c(q)$ and $\widehat{N}_c(q)$.

Proposition 1. *Fix an attention head and query q, and partition the context into chunks $\mathcal{C}$. Suppose each compressed chunk $c \in \mathcal{C}$ satisfies*

$$|\widehat{Z}_c(q) - Z_c(q)| \le \epsilon_Z Z_c(q), \qquad \|\widehat{N}_c(q) - N_c(q)\| \le \epsilon_N Z_c(q),$$

where $\epsilon_Z < 1$, and the original attention output is bounded by V, i.e., $\|A(q)\| \le V$. Then

$$\|\widehat{A}(q) - A(q)\| \le \frac{\epsilon_N + V\epsilon_Z}{1 - \epsilon_Z}.$$

In particular, this bound is independent of the number of compressed chunks.

Proof. Fix an attention head of dimension d, a query $q \in \mathbb{R}^d$, and a collection of chunks $\mathcal{C}$. For a chunk $c = \{(k_i, v_i)\}_{i=1}^T$, define its partition function and value-weighted mass by

$$Z_c(q) = \sum_{i \in c} \exp\left(\frac{q^\top k_i}{\sqrt{d}}\right), \qquad N_c(q) = \sum_{i \in c} \exp\left(\frac{q^\top k_i}{\sqrt{d}}\right) v_i.$$

Let compression replace c by $\widehat{c} = \{(\widehat{k}_i, \widehat{v}_i, \beta_i)\}_{i=1}^t$, where β_i is the learned log-weight of compressed token i. Define

$$\widehat{Z}_c(q) = \sum_{i=1}^t \exp\left(\frac{q^\top \widehat{k}_i}{\sqrt{d}} + \beta_i\right), \qquad \widehat{N}_c(q) = \sum_{i=1}^t \exp\left(\frac{q^\top \widehat{k}_i}{\sqrt{d}} + \beta_i\right) \widehat{v}_i.$$

Assume that each compressed chunk satisfies

$$|\widehat{Z}_c(q) - Z_c(q)| \le \epsilon_Z Z_c(q), \qquad \|\widehat{N}_c(q) - N_c(q)\| \le \epsilon_N Z_c(q).$$

The exact and compressed global quantities are

$$Z = \sum_{c \in \mathcal{C}} Z_c(q), \qquad N = \sum_{c \in \mathcal{C}} N_c(q),$$
$$\widehat{Z} = \sum_{c \in \mathcal{C}} \widehat{Z}_c(q), \qquad \widehat{N} = \sum_{c \in \mathcal{C}} \widehat{N}_c(q),$$

and the corresponding attention outputs are

$$A(q) = \frac{N}{Z}, \qquad \widehat{A}(q) = \frac{\widehat{N}}{\widehat{Z}}.$$

By the triangle inequality,

$$|\widehat{Z} - Z| \le \epsilon_Z Z, \qquad \|\widehat{N} - N\| \le \epsilon_N Z.$$

Consequently, $\widehat{Z} \ge (1-\epsilon_Z)Z$. Moreover, the assumption $\|A(q)\| \le V$ gives $\|N\| \le VZ$. Therefore,

$$\begin{aligned} \|\widehat{A}(q) - A(q)\| &\le \frac{\|\widehat{N} - N\|}{\widehat{Z}} + \frac{\|N\||Z - \widehat{Z}|}{Z\widehat{Z}} \\ &\le \frac{\epsilon_N + V\epsilon_Z}{1 - \epsilon_Z}. \end{aligned}$$

The bound contains no dependence on $|\mathcal{C}|$, the number of compressed chunks. □

This proposition lays the attention-level argument underlying our chunk-wise training and inference schedule. It extends the mass matching view used by Attention Matching from one compacted block to a set of independently compacted blocks for a fixed attention head and query q. We do note that it does not bound errors induced by earlier compressed layers or autoregressive steps. We use it as motivation for the training decomposition and not an end-to-end theoretical guarantee.

C.2 Variance analysis of sampled training

Continuing from section 3, we have derived some gradients of the total loss w.r.t chunks. Let us denote a shorthand for these notation, let g_j be the exact gradient, using straight-through pseudo-gradient for the gate parameters, at target boundary j with n_j compressed prefix chunks $\mathcal{P}_j$, let g be the full distillation gradient, i.e.

$$g_j = \nabla_\psi \mathcal{L}_j = \sum_{b \in \mathcal{P}_j} G_{j,b}, \qquad g = \nabla_\psi \mathcal{L}_{\mathrm{KL}} = \frac{1}{M-1} \sum_{j=1}^{M-1} g_j.$$

Let $\widehat{g}$ be our dual-sampled estimator for boundary J and prefix chunk R, i.e.

$$\widehat{g} = n_J G_{J,R}.$$

Because $\widehat{g}$ is strictly unbiased, its total variance can be decomposed via the law of total variance. Conditioning on the sampled boundary J yields:

$$\mathbb{E}\left[\|\widehat{g} - g\|^2\right] = \mathbb{E}_J\left[\mathbb{E}_{R|J}\left[\|\widehat{g} - g_J\|^2\right]\right] + \mathbb{E}_J\left[\|g_J - g\|^2\right].$$

Expanding the inner expectation over the uniform choice of $R \in \mathcal{P}_j$ gives the decomposition:

$$\mathbb{E}\left[\|\widehat{g} - g\|^2\right] = \frac{1}{M-1} \sum_{j=1}^{M-1} \left(\frac{1}{n_j} \sum_{b \in \mathcal{P}_j} \|n_j G_{j,b} - g_j\|^2 \right) + \frac{1}{M-1} \sum_{j=1}^{M-1} \|g_j - g\|^2.$$

We can rearrange the first term to isolate the intra-boundary variance from the inter-boundary variance:

$$\mathbb{E}\big[\|\widehat{g}-g\|^2\big] = \frac{1}{M-1}\sum_{j=1}^{M-1} n_j \sum_{b\in\mathcal{P}_j}\left\|G_{j,b}-\frac{g_j}{n_j}\right\|^2 + \frac{1}{M-1}\sum_{j=1}^{M-1}\|g_j-g\|^2.$$

To analyze how this variance scales with the trajectory length, we can expand the squared norms and simplify. Expanding the inner term yields a more compact expression:

$$\begin{aligned}\mathbb{E}\big[\|\widehat{g}-g\|^2\big] &= \frac{1}{M-1}\sum_{j=1}^{M-1} n_j \sum_{b\in\mathcal{P}_j}\|G_{j,b}\|^2 - \frac{1}{M-1}\sum_{j=1}^{M-1}\|g_j\|^2 + \frac{1}{M-1}\sum_{j=1}^{M-1}\|g_j-g\|^2 \\ &= \frac{1}{M-1}\sum_{j=1}^{M-1} n_j \sum_{b\in\mathcal{P}_j}\|G_{j,b}\|^2 - \|g\|^2\end{aligned}$$

Assuming the gradient contribution of any individual chunk is bounded by $\|G_{j,b}\| \le C$, we can place an upper bound for this variance. Because boundary j contains $n_j = j$ prefix chunks, the worst-case variance scales quadratically with the total number of chunks M:

$$\begin{aligned}\mathbb{E}\big[\|\widehat{g}-g\|^2\big] &\le \frac{C^2}{M-1}\sum_{j=1}^{M-1} j^2. \\ &\le \frac{C^2 M(2M-1)}{6} = \mathcal{O}(M^2).\end{aligned}$$

This quadratic scaling naturally motivates theoretical variance reduction techniques, such as sampling $K > 1$ prefix chunks to form a multi-sample estimator:

$$\widehat{g}_K = \frac{n_J}{K}\sum_{k=1}^{K} G_{J,R_k}.$$

Assuming independent samples with replacement, this strictly scales down the intra boundary variance by K:

$$\mathbb{E}\big[\|\widehat{g}_K-g\|^2\big] = \frac{1}{K}\left(\frac{1}{M-1}\sum_{j=1}^{M-1} n_j \sum_{b\in\mathcal{P}_j}\left\|G_{j,b}-\frac{g_j}{n_j}\right\|^2\right) + \frac{1}{M-1}\sum_{j=1}^{M-1}\|g_j-g\|^2.$$

Alternatively, employing non-uniform importance sampling by drawing prefix chunk b with probability $p_{j,b}$ modifies the estimator and its conditional variance:

$$\widehat{g}_{\mathrm{IS}} = \frac{G_{J,R}}{p_{J,R}}, \qquad \mathbb{E}_{R|J=j}\big[\|\widehat{g}_{\mathrm{IS}}-g_j\|^2\big] = \sum_{b\in\mathcal{P}_j}\frac{\|G_{j,b}\|^2}{p_{j,b}} - \|g_j\|^2.$$

Minimizing this conditional variance yields the theoretically optimal sampling distribution:

$$p^{\star}_{j,b} = \frac{\|G_{j,b}\|}{\sum_{r\in\mathcal{P}_j}\|G_{j,r}\|}.$$

In practice, however, both of these approaches have their own tradeoffs (for example in the second approach, calculating the full gradient norm would defeat the memory saving purpose of sampling) and simply scaling up the global batch size directly reduces the variance of the sample mean proportionally. Because our single-chunk sampling strategy yields huge memory savings, we can accommodate large batch sizes required to bound the variance. This simplifies training while still providing well learnable signal to the model. We justify our choice empirically with the stability of our training runs, while noting that batch size above 32 yields little to no improvements.

D IMPLEMENTATION DETAILS

We detail the implementation of our compactor and attention backends below. Default configurations are provided in Tables 4 and 5.

STILL implementation. Because the original STILL (O'Neill et al., 2026) source code is not open sourced, we implement both our learned routing method and the static STILL baseline within a unified Qwen3 framework based on their paper. We freeze the base language model and attach an independent compactor to each transformer layer, operating independently across KV heads. For a given head with dimension d, the compactor concatenates unrotated keys and values into $[K; V] \in \mathbb{R}^{T \times 2d}$ and maps learned latent queries through two Perceiver blocks. Each block comprises cross-attention into $[K; V]$, latent self-attention, residual connections, and RMS normalization, followed by linear projections to output compact keys, values, and an attention bias β. To ensure training stability, we apply a near-identity initialization: the key and value projections select their respective halves of $[K; V]$, the first cross-attention output projection is an identity mapping, and all remaining projections are zero-initialized. During distillation, gradients flow through the frozen student model directly into the compacted representations, updating only the compactor parameters.

Gating and compression budget. The compactor initially synthesizes T_{init} latent candidates per layer and head. Our primary architecture employs static-hard gating, learning a single FP32 logit for each latent slot and applying a hard straight-through Gumbel-Sigmoid mask at a fixed temperature. For our ablation studies (Section 4.3), we also implement an input-dependent variant and a soft-mask variant. In the input-dependent variant, a content projection and bias are learned per layer and KV head and shared across latent slots, so the gate logit depends on the current latent state. In the soft-mask variant, all candidates remain dense during training and $\log \sigma((a + \epsilon)/\tau)$ is added to their attention bias, while a straight-through hard sample supplies the budget gradient. Its temperature is annealed linearly from 1.0 to 0.05. These variants are evaluated with the same deterministic hard threshold as the main method, but neither is used for the main results. The target budget is held at T_{init} during warmup and linearly annealed to the final budget B_s. Hard-selected entries are packed independently for every KV head at inference, so different heads may retain different numbers of latents even though the loss constrains their mean. This results in the final compression ratio to not be exact the specified budget.

Positional embeddings. Because the base Qwen3 model caches pre-rotated keys, we first invert the Rotary Position Embedding (RoPE) using the original absolute token positions before compaction. The Perceiver then applies its own positional encoding during its inner cross-attention. Tokens from the uncompacted sequence (source keys) retain their original positions, while the perceiver's own learned latent queries are assigned positions uniformly interpolated across the source chunk. Finally, the resulting compacted keys are re-rotated using the base model's RoPE at those latent positions. We track logical sequence positions with no reference to the current physical cache length, ensuring subsequent generated tokens receive correct absolute positional encodings.

Attention bias β. Each resulting compacted latent has a corresponding learned scalar bias β, which is added directly to its pre-softmax attention score. The projection is zero-initialized and bounded to prevent attention spikes early in training: $\beta = \beta_{\max} \tanh(\beta_{\text{raw}}/\beta_{\max})$, with $\beta_{\max} = 5$. While our implementation supports a mass-matching offset proportional to $\log(T/T_{\text{init}})$, we disable it for all reported experiments. Uncompressed tokens simply receive a zero bias.

FlexAttention implementation. We use PyTorch FlexAttention (Dong et al., 2025) to support differentiable attention modifications without materializing dense $Q \times K$ matrices. We define a custom block mask concatenating the uncompressed prompt, the previously compacted prefix chunks, and the (causal) target chunk. Compacted tokens are injected directly via FlexAttention's score modification function. This function maps queries to their corresponding KV heads, adds the attention bias β, and applies the straight-through gate mask. This natively fuses all masking and bias operations, allowing gradients to flow directly to the compact latents and gates.

Sampled-boundary distillation. Following the dual-sampling strategy in Section 3, we partition CoT trajectories into fixed-size source chunks and uniformly sample a single target boundary and a single prefix chunk, of which we only retain the compactor autograd graph for that specific chunk. We calculate $\mathcal{L}_{\text{KL}}$ as the forward KL divergence of the sampled chunk between teacher and student; we calculate $\mathcal{L}_{\text{budget}}$ as the average of the $\bar{k}_{\text{exp}}$ of each of the prefix chunks. We apply a

Hyperparameter	Value
Training data	OpenR1-Math-220k (correct traces only)
Max sequence length	16,384
Source chunk size (T)	512
Trailing window	0
Latent candidates (T_{init})	256
Target budgets (B_s)	$\{64, 43, 32\}$ per head (for 8x, 12x, 16x)
Perceiver depth	2 blocks
Perceiver attention heads	1
Perceiver RoPE base	10
Budget loss weight (λ)	0.1
Budget schedule	Constant 256 for 30 steps, linear anneal over 1,500 steps
Gate mask	Hard straight-through Gumbel-Sigmoid
Gate initialization	4.0
Gate temperature (τ)	0.5
Optimizer	AdamW
Learning rate schedule	30-step linear warmup, cosine decay
Learning rate (compactor)	5×10^{-5}
Learning rate (gate logits)	10^{-2}
Weight decay	10^{-4}
Gradient clipping	1.0
Effective batch size	16
Training steps	2,500
Random seed	42

Table 4: Training hyperparameters for the main experiments.

corresponding token-count importance weight on both $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{budget}}$. Our derivation in Section 3 is simplified to assume equal size chunks and omit this importance weights. However in practice, this is needed to fully recover an unbiased estimate of the full objective to account for a shorter final chunk. We optimize the token weighted forward KL divergence and the budget penalty, leaving zero uncompressed trailing tokens. We apply a corresponding token-count importance weight on both $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{budget}}$ to account for differing target chunk size.

Paged inference and custom Triton kernels. We use standard PyTorch's SDPA backend for prefill (defaulted to FlashAttention2 (Dao et al., 2022)), grouping prompts by sequence length to avoid dense padding masks. For decode, we implement a custom Triton paged-attention kernel over a unified memory pool of K, V, and β pages. Because hard gating produces variable cache lengths across heads, we maintain independent block tables and valid-length arrays for every $(B, \text{layer}, \text{KV head})$ tuple.

The decode kernel accesses these page tables directly. It maps query heads to KV heads, evaluates the biased attention scores, and accumulates softmax statistics and value-weighted sums. This takes into account the schedules for differing head lengths, bypassing longest-head padding entirely. For extended contexts, we partition head-specific caches into 16-page blocks and compile a packed work schedule. The primary attention kernel restricts its operation to valid $(B, \text{query head}, \text{partition})$ caches, tracking statistics for online-softmax. A secondary reduction kernel then merges these partition-level statistics. Because cache lengths grow monotonically, we retain this execution schedule in memory and rebuild it only when a head crosses a partition boundary.

At each compaction boundary, a fused Triton gather operation reconstructs the contiguous source chunk from the generation pages across all layers and heads. After the compactor compact the chunk, we release the original pages and execute a fused store to write back the retained latent representations. To minimize fragmentation, original prompt and consecutive compacted chunks share a packed prefix; a new chunk fills the remainder of the last occupied page before requesting a new allocation. This bounds fragmentation to at most one partial prefix page and one partial generation page per layer and head. We default to 16-entry pages to match standard paged attention (Kwon et al., 2023).

Hyperparameter	Value
Samples per problem	4
Sampling temperature	0.6
Top-p	0.95
Top-k	20
Min-p	0
Base seed	42 (same offset per problem multi sampling)
Eval Batch size	8
Max reasoning tokens	38,912 (math) / 32,768 (code)
Final-answer budget	32 (math) / 4,096 (code)
Source chunk size (T)	512
Trailing window	128 (Ours, Vanilla STILL), 512 (DMS, AM)
Latent candidates (T_{init})	256
Target budgets (B)	$\{64, 43, 32\}$ per head
Inference gate decision	Hard mask (logit > 0)
Paged-cache block size	16 entries

Table 5: Online generation and sampling hyperparameters.

E EVALUATION PROTOCOL

We evaluate all baselines using identical prompts, token budgets, sampling parameters (Table 5), and per-completion random seeds as our methods. To ensure fair comparison, all methods share a unified decoding harness with consistent batching and stopping criteria. We use the same answer parsing and matching method for all baselines, conditioned on answers in \boxed{ }. Cache occupancy is measured strictly from physically retained entries rather than allocated capacity and we record end-to-end generation time excluding prefill. All methods are evaluated using identical prompts. We manually construct the Qwen ChatML prompt and initialize the assistant response with an open `<think>` tag. The task-specific templates are shown below.

Mathematical reasoning: system message

```
Solve the problem by thinking step-by-step.
Enclose your complete reasoning process within <think> and </think>
    tags.
After thinking, provide the final answer within \boxed{}..
```

Mathematical reasoning: user-message template

```
{problem}
```

LiveCodeBench: system message

```
You are a helpful and harmless assistant. You are Qwen developed by
    Alibaba. You should think step-by-step.
```

LiveCodeBench: user-message template

```
You will be given a question (problem specification) and will
    generate a correct Python program that matches the specification
    and passes all tests.

Question: {question}
```

````
Read the inputs from stdin solve the problem and write the answer to
    stdout (do not directly test on the sample inputs). Enclose your
    code within delimiters as follows. Ensure that when the python
    program runs, it reads the inputs, runs the algorithm and writes
    output to STDOUT.
```python
# YOUR CODE HERE
```
````

F BASELINE IMPLEMENTATIONS

Controlled comparison with Vanilla STILL. We conduct experiments with Vanilla STILL to correctly measure the effect of our jointly learned overcomplete representations. We use the same Qwen3 model, same perceiver backbone architecture and implementation, training setup, online schedule, inference schedule, and inference systems backend. The main difference in Vanilla STILL and Ours is the number of candidate outputs per head; Vanilla STILL directly match the budget B, while Ours emits an overcomplete candidate $T_{init} = 256$ subject to control by the gates and budget loss. Therefore, the extra parameter of our method compared to Vanilla STILL is contained in the final candidate latent projection layer and the gates itself, making it suitable to be the direct baseline for our proposed changes of overcomplete latents.

Attention Matching (AM). We use the upstream Fast KV Compaction implementation (Zweiger et al., 2026) configured with `AM-HighestAttnKeys-basic` and `repeat-prefill`. To align with our online setting, we enforce an append-only schedule: generation is partitioned into C-token chunks, which are sequentially compressed into c tokens per layer and head. The prompt and the most recent trailing window remain uncompacted. AM generates reference queries online based strictly on the current physical cache. Keys are selected via RMS attention over these queries, while attention biases and values are fitted using least squares. We tensorize the key-selection and fitting operations across sequences, layers, and heads to support batched inference.

TOVA. We implement TOVA (Oren et al., 2024) via the NVIDIA `kvpress` library. Following DMS baseline implementation, we interpret $C \to c$ as a strict compression ratio $r = C/c$. We set a fixed global KV capacity $B_{\text{TOVA}} = \max(1, \lfloor (L_{\text{in}} + L_{\text{gen}})/r \rfloor)$, where L_{in} and L_{gen} denote the prompt length and maximum generation budget, respectively. During generation, TOVA performs dense prefill and decoding until this capacity is reached, after which it evicts tokens at every decoding step to maintain the target budget.

DMS. We train and evaluate DMS (Łańcucki et al., 2025) using the reference NVIDIA ModelOpt adapter-only recipe. The base LLM remains frozen while learned eviction policies are integrated into the query projections. Because compression ratio is defined by the trained checkpoint rather than a specified parameter, we fine-tune separate checkpoints for each target compression ratio on the Open-R1 dataset. At inference, the prompt is processed via dense prefill. After which we transition the paged KV cache to decode mode, activating continuous eviction while preserving the checkpoint's configured trailing window. DMS operates without a specified capacity constraint or chunk schedule, instead, memory allocation and eviction are natively managed by the ModelOpt backend.

G RL COMPATIBILITY

As a proof of concept, our KV compaction integrates with very little effort into existing RL pipelines by treating the compressed cache as the policy's recurrent state. During rollout, generation as we described, periodically compacting older KV chunks with compact latents, while recording the full completion. By recording the sampled tokens, log-probabilities, sampling parameters, and compaction boundaries, we reconstruct the exact state sequence during training (state here being the chunk generated after compaction). Reducing the peak KV payload directly enables longer reasoning traces and accommodates more concurrent rollout trajectories. Furthermore, this requires no modification to the underlying RL objective, as reference policy, advantage calculation, and loss are similar to non compacted sequences.

During replay, the sampled sequence is processed using identical cache behaviour and compaction boundaries. While we can set the compactor parameters to remain frozen, gradients must propagate through its operations to update the earlier language model states that produced the dense chunks. Consequently, compaction reduces total memory required during rollout, exact replay implementation still result in non-reduced activation memory and remains slower than optimized batched teacher forcing. However, this overhead is an engineering limitation rather than an inherent limitation. A custom backward-able Triton kernel could operate directly on variable-length caches across heads, avoiding padding and dense intermediate tensors. Combined with activation recomputation across compaction boundaries, such a kernel would extend our memory savings to the training phase while preserving exact recurrent gradients.

H LEARNED HEAD BUDGET

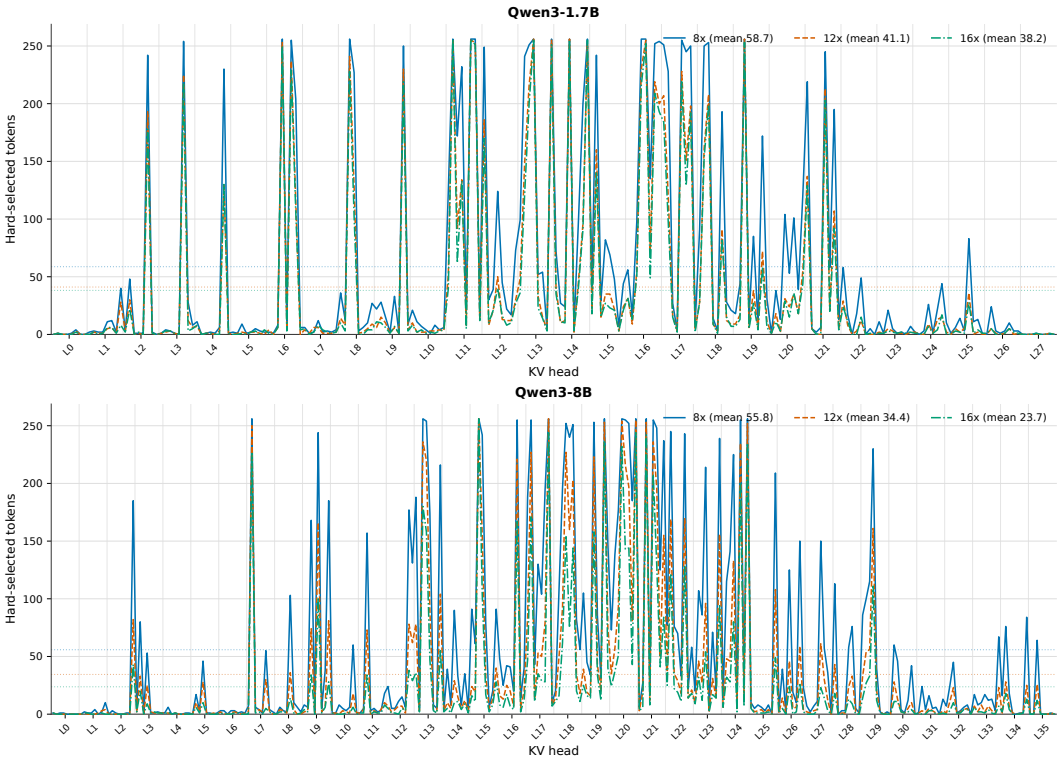

Figure 5: Learned head budget for all our trained compactor with 8x, 12x, and 16x compaction ratio

Due to the static nature of our gates, we can immediately plot the learned head budget based on the gating parameter (Figure 5). Across all models, the learned token budget is heavily concentrated in the middle KV heads, while early and late heads remain visibly sparse. Within each model, the head budget profiles have the exact same peaks and valleys across the 8x, 12x, and 16x compaction ratios, which aligns with the expected training behavior of dropping tokens. This implies that optimal head ratios are architecture and task specific rather than pure base model and parameter count, and it provides a strong case for aggressively pruning earlier heads during reasoning workloads.

The budget loss depends on the expected budget $\bar{k}_{\text{exp}}$, which is the sum of relaxed gate probabilities. Inference uses $\bar{k}_{\text{hard}}$, the number of logits above the deterministic threshold. These two may differ, and several gates with probabilities below 0.5 can still jointly contribute high fractional mass to $\bar{k}_{\text{exp}}$ during training but then are all removed by the deterministic thresholding. The effect of the contributions are controlled by the temperature variable, and we can expect the performance in inference to be similar to training due to modulation of the logits by gumbel noise in training.

The inference data also shows that larger models can operate on far fewer tokens (Table 6). Trained against target budgets of 64 (8x CR), 43 (12x CR), and 32 (16x CR) tokens, the 1.7B model struggles to reach the target at higher compression, exceeding the 16x budget with a mean of 38.2 tokens. Conversely, the 8B models easily surpass the constraints, dropping well below the 32-token budget and averages 23.7 tokens.

To explain this we plot the token behaviour during training (Figure 6). After the initial hard drop, we can see the number of inference tokens continues to decrease for the 8B model, while the smaller 1.7B model introduce a lot of uncertainties that makes it harder for the overall inference token to converge to a higher compression ratio. This suggests that increased model scale and capabilities tolerates much higher sparsity while maintaining better performance.

Model	Metric	8× (64)	12× (43)	16× (32)
Qwen3-1.7B	Actual Tokens	58.7	41.1	38.2
	Actual CR	8.7×	12.5×	13.4×
Qwen3-8B	Actual Tokens	55.8	34.4	23.7
	Actual CR	9.2×	14.9×	21.6×

Table 6: Actual retained-token budgets and corresponding actual compression ratios for our static-hard models.

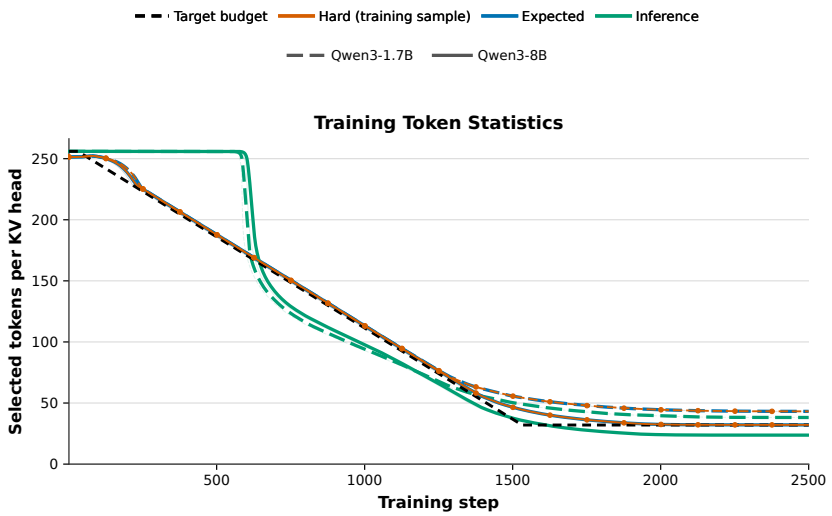

Figure 6: Training token statistics for Qwen3-1.7B and Qwen3-8B under a shared budget target of 32. Hard denotes the sampled straight-through training tokens, expected is the summed gate probability, and inference applies the deterministic gate threshold used at evaluation

I Generation length analysis

We analyze the generation results on both the AIME and HMMT benchmarks, specifically focusing on mean generation length and the rate of incomplete generations (Table 7 and Fig 7). We compare all methods using Qwen3-8B at 8× CR, and define a completion as unfinished if it reaches the generation cap without emitting EOS. This analysis is important as it explains why the Peak KV statistics differs wildly for different method with the same CR.

We can see that almost every compacted generation produces more token than the base model. Compared to DMS, Our method generates around 18% more tokens on average on AIME, and around 29% more tokens on average on HMMT. This still persist when we only take completion that reaches EOS, where Ours is still longer by 6% and 19% on AIME and HMMT respectively. However, we do note that shorter generation length does not correspond directly with accuracy, as TOVA performs worse than Ours while maintaining similar mean generation length and unfinished generation percentage, and DMS accuracy is below uncompressed base model with similar mean generation length and unfinished generation ratio. We do note that generation can still output their answer even if they terminates by reaching the generation cap instead of EOS, and thus qualifies as being correct.

This length discrepancy contributes to the higher Peak KV of our 8× compactor relative to DMS on AIME and coincides with lower all-completion accuracy, as our method reaches the generation cap on 9.2 percentage points more completions. For additional context, we report accuracy conditioned on EOS termination; because each method produces a different EOS-completed subset, these statistics are descriptive and do not directly measure relative overall capability. When our method emits an EOS token, its accuracy is 65.7% on AIME, compared with 58.3% across all completions, and 42.1% on HMMT, compared with 36.4% across all completions. For DMS, the corresponding differences are

smaller, with EOS-conditioned accuracy increasing from 62.5% to 64.7% on AIME and from 34.1% to 36.3% on HMMT-Feb. These results can't be interepreted directly, as every method terminates on a different subset of samples, and we cannot draw any comparison between different models.

The combination of longer EOS-terminated sequences, more cap-limited generations, and relatively high accuracy among terminated sequences is consistent with delayed termination under lossy merging. It suggests that our compactor for the most part preserves the complex latent reasoning states required to solve the problem, but occasionally struggles to end the reasoning step and trigger termination. We attribute this behavior partly to our training setup, in which the compactor rarely observes EOS-containing targets, unlike DMS. Despite the disadvantage of the unfinished completions, our method still overall performance is still strong; Ours are consistently at the pareto frontier, while matching the state-of-the-art in the memory matched setting.

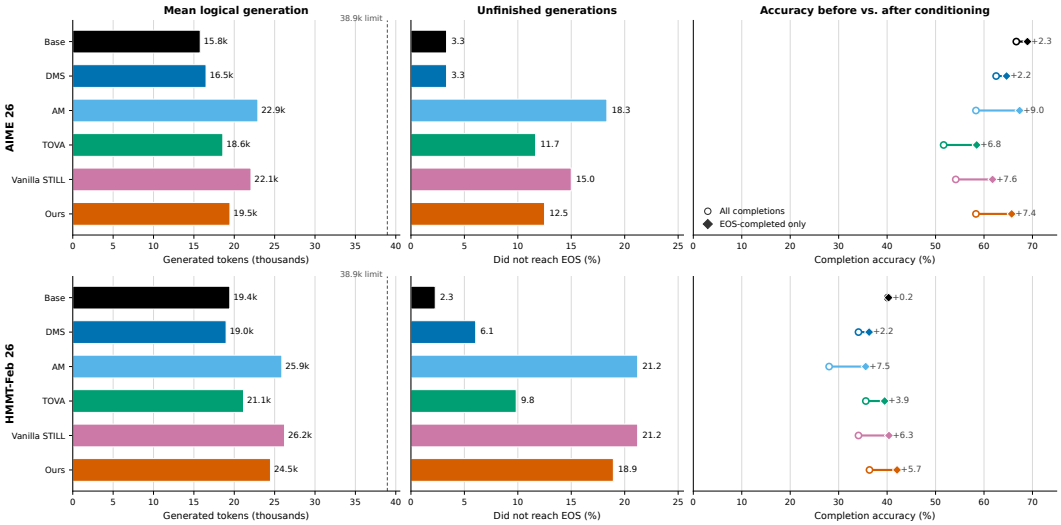

Figure 7: Completion length, unfinished generation rate, accuracy, and EOS-conditioned accuracy for Qwen3-8B at 8x CR

Dataset	Method	Mean tok.↓	Finished tok.↓	Unfinished↓	All acc.↑	Finished acc.↑
AIME 26	Base	15.8k	15.0k	3.3%	66.7	69.0
	DMS	16.5k	15.7k	3.3%	62.5	64.7
	AM	22.9k	19.3k	18.3%	58.3	67.3
	TOVA	18.6k	15.9k	11.7%	51.7	58.5
	Vanilla STILL	22.1k	19.1k	15.0%	54.2	61.8
	Ours	19.5k	16.7k	12.5%	58.3	65.7
HMMT–Feb 26	Base	19.4k	19.0k	2.3%	40.2	40.3
	DMS	19.0k	17.7k	6.1%	34.1	36.3
	AM	25.9k	22.4k	21.2%	28.0	35.6
	TOVA	21.1k	19.2k	9.8%	35.6	39.5
	Vanilla STILL	26.2k	22.8k	21.2%	34.1	40.4
	Ours	24.5k	21.1k	18.9%	36.4	42.1

Table 7: Completion length and truncation for Qwen3-8B at $8\times$ compression. Mean tok. averages the generated length over all completions. Finished tok. conditions on termination by EOS. Unfinished reports the percentage that reach the 38.9k generated-token cap without EOS. All acc. uses every completion, whereas Finished acc. conditions on EOS.

J ABLATION DATA

Method	AIME26↑	HMMT Feb 26↑
Static-Hard	**21.67**	**16.67**
Static-Soft	14.17	10.61
Input-Hard	9.17	4.55
Input-Soft	19.17	14.39

Table 8: Ablation for architecture and training choices. Proxy Qwen3-1.7B result on AIME 26 and HMMT-Feb 26. Static-Hard preserves the best performance during training and inference.

Num Overcomplete Latents	**Avg@4 ↑**	**95% Range**	**Peak KV**
64	56.7	[49.8, 63.4]	164 MiB
128	62.1	[55.0, 69.0]	157 MiB
192	**63.8**	[57.1, 70.3]	165 MiB
256	62.1	[55.0, 68.8]	176 MiB

Table 9: Candidate-capacity (T_{init}) ablation at a fixed 32-token expected budget on a Qwen3-1.7B compactor. Performance plateaus at initialization greater than 128

Trailing tokens	**Avg@4↑**	**Gain vs 0**	**Peak KV (MiB)**	**Mem Reads (GB)**
0	68.3	–	203.5	1891.3
16	68.8	+0.6 [-3.0, 3.9]	200.9	1909.6
32	66.4	-1.9 [-4.7, 0.9]	200.7	1798.8
64	69.4	+1.1 [-2.1, 4.3]	199.0	1764.9
128	**70.9**	+2.6 [-0.2, 5.6]	202.0	1776.2
256	70.7	+2.4 [-0.9, 5.8]	205.5	1727.9

Table 10: Trailing-token sweep proxy on validation dataset (MATH-500 level 5) to find the optimal min trail token parameter

K FULL RESULTS

Model	Method	CR	AIME 26 Avg@4↑	AIME 26 Peak KV (MiB) ↓	AIME 26 Mem Reads (GB) ↓	HMMT-Feb 26 Avg@4↑	HMMT-Feb 26 Peak KV (MiB) ↓	HMMT-Feb 26 Mem Reads (GB) ↓	LCB v6 Avg@4↑	LCB v6 Peak KV (MiB) ↓	LCB v6 Mem Reads (GB) ↓
Qwen3-1.7B	Base	–	35.8	1800.3	18970.4	25.0	1864.5	20045.0	29.9	1679.3	17570.9
	DMS	8×	31.7	270.4	4184.9	14.4	280.2	4341.1	26.3	267.3	3697.2
	AM	8×	27.5	377.3	5070.2	20.5	404.0	5977.6	30.6	378.3	4602.6
		12×	27.5	320.3	4890.2	18.9	313.7	4691.1	–	–	–
		16×	29.2	273.4	4276.7	17.4	276.2	4473.6	–	–	–
	TOVA	8×	25.8	529.2	8756.9	20.5	530.9	9219.7	24.9	476.1	9818.3
		12×	22.5	359.0	6560.9	18.9	359.1	6488.5	–	–	–
		16×	14.2	267.5	5265.9	13.6	267.3	5233.0	–	–	–
	Vanilla STILL	8×	20.0	397.2	6166.4	18.2	408.4	6452.5	25.0	435.8	6773.8
		12×	14.2	313.6	5188.0	13.6	313.0	5136.2	–	–	–
		16×	15.8	256.7	4256.3	13.6	252.8	4236.8	–	–	–
	Ours	8×	30.0	334.6	4672.4	18.9	346.1	4894.4	25.6	362.6	5075.7
		12×	25.0	262.5	3742.7	17.4	264.4	3767.8	–	–	–
		16×	24.2	259.1	3870.5	19.7	257.9	3708.2	–	–	–
Qwen3-8B	Base	–	66.7	2244.7	20126.5	40.2	2755.3	29781.1	47.4	2113.4	13619.3
	DMS	8×	62.5	351.4	4679.4	34.1	381.1	5493.5	45.1	374.3	4448.4
	AM	8×	58.3	552.9	8939.2	28.0	601.2	10520.2	46.6	497.0	6590.8
		12×	53.3	452.4	7900.5	26.5	475.8	8776.6	–	–	–
		16×	49.2	406.0	7787.6	25.8	402.0	7661.8	–	–	–
	TOVA	8×	51.7	681.0	11775.8	35.6	684.4	13583.3	39.6	604.1	11059.0
		12×	35.0	461.6	9996.4	31.8	461.8	10475.9	–	–	–
		16×	27.5	343.9	8175.4	24.2	343.6	8280.8	–	–	–
	Vanilla STILL	8×	54.2	489.8	7551.0	34.1	558.1	9458.6	43.0	534.7	8060.6
		12×	43.3	395.0	6675.5	27.3	425.3	7520.7	–	–	–
		16×	40.8	339.2	6055.7	26.5	366.9	6988.4	–	–	–
	Ours	8×	58.3	401.4	5655.7	36.4	473.6	7740.0	45.3	450.9	6401.3
		12×	61.7	322.4	5081.7	34.8	347.2	5952.3	–	–	–
		16×	56.7	261.2	4302.9	30.3	279.2	4948.9	–	–	–

Table 11: Full result of online KV cache compaction methods of both Qwen3 variants (1.7B, 8B) on long-context mathematical and coding reasoning tasks. We report average correctness (Avg@4), the peak size of the stored KV cache, and the KV bytes read by attention during decoding, both averaged per completion. LiveCodeBench was evaluated at 8× only.

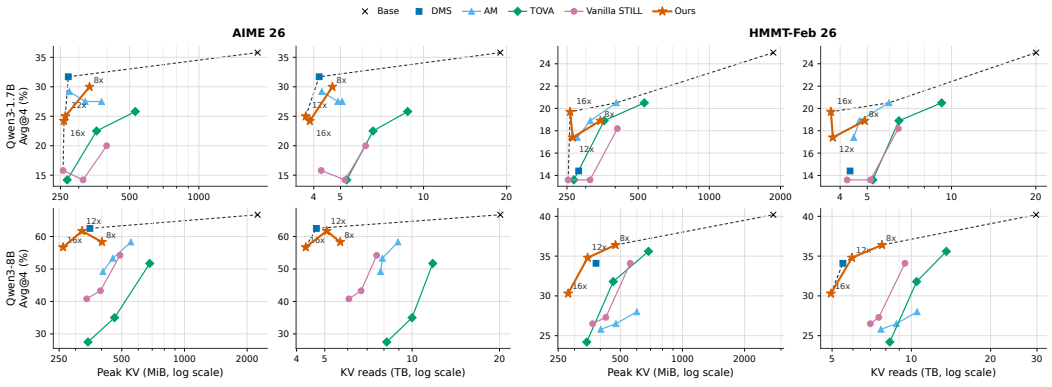

Figure 8: Pareto for All evaluated models

Model	DMS	Ours	Δ (Ours−DMS)	Paired 95% CI
Qwen3-1.7B	26.3	25.6	-0.7	[-3.0, +1.6]
Qwen3-8B	45.1	45.3	+0.1	[-2.3, +2.6]

Table 12: Paired comparison of Ours and DMS on LiveCodeBench v6 at $8\times$ compression. We bootstrap paired problem-level differences after averaging with 10,000 resamples. All intervals include zero, so the observed differences are not statistically distinguishable at the 95% level

		AIME 26			**HMMT-Feb 26**		
Model	**CR**	**Vanilla**	**Ours**	**Gain**	**Vanilla**	**Ours**	**Gain**
Qwen3-1.7B	$8\times$	20.0	30.0	**+10.0 [3.3, 17.5]**	18.2	18.9	+0.8 [-6.1, 7.6]
	$12\times$	14.2	25.0	**+10.8 [1.7, 20.8]**	13.6	17.4	+3.8 [0.0, 7.6]
	$16\times$	15.8	24.2	**+8.3 [1.7, 15.8]**	13.6	19.7	**+6.1 [1.5, 11.4]**
Qwen3-8B	$8\times$	54.2	58.3	**+4.2 [0.8, 8.3]**	34.1	36.4	+2.3 [-5.3, 9.8]
	$12\times$	43.3	61.7	**+18.3 [10.0, 28.3]**	27.3	34.8	**+7.6 [1.5, 14.4]**
	$16\times$	40.8	56.7	**+15.8 [4.2, 28.3]**	26.5	30.3	+3.8 [-1.5, 9.8]

Table 13: Uniform control (Vanilla) versus learned latent allocation (Ours) at the same target compression ratio. We bootstrap paired problem-level differences after averaging with 10,000 resamples with 90% confidence interval; The joint overcomplete latent and learned allocation helps in more aggressive compression ratio. On LCB v6 at 8x, the gates have no measurable effect, which is expected as due to the OOD nature of code benchmark

		AIME 26			**HMMT-Feb 26**		
Model	**Method**	$8\times$	$12\times$	$16\times$	$8\times$	$12\times$	$16\times$
1.7B	DMS	31.7	–	–	14.4	–	–
	AM	27.5	27.5	29.2	20.5	18.9	17.4
	TOVA	25.8	22.5	14.2	20.5	18.9	13.6
	Vanilla STILL	20.0	14.2	15.8	18.2	13.6	13.6
	Ours	30.0	25.0	24.2	18.9	17.4	19.7
8B	DMS	62.5	–	–	34.1	–	–
	AM	58.3	53.3	49.2	28.0	26.5	25.8
	TOVA	51.7	35.0	27.5	35.6	31.8	24.2
	Vanilla STILL	54.2	43.3	40.8	34.1	27.3	26.5
	Ours	58.3	61.7	56.7	36.4	34.8	30.3

Table 14: Accuracy against the target compression ratio per methods. TOVA and AM are training-free and run at each specified ratio; DMS, Vanilla STILL, and Ours are trained separately for each ratio. This shows degradation behaviour and cannot be interpreted directly as KV caches differs greatly even for the same compression ratio for each method

Model	Method	CR	Peak KV (MiB)	AIME 26	HMMT-Feb 26
1.7B	Vanilla STILL	16×	257	15.8	13.6
	Ours	16×	259	24.2	19.7
	Ours	12×	264	25.0	17.4
	TOVA	16×	268	14.2	13.6
	AM	16×	276	29.2	17.4
	DMS	8×	280	31.7	14.4
	Vanilla STILL	12×	314	14.2	13.6
	AM	12×	320	27.5	18.9
	Ours	8×	346	30.0	18.9
	TOVA	12×	359	22.5	18.9
	AM	8×	404	27.5	20.5
	Vanilla STILL	8×	408	20.0	18.2
	TOVA	8×	531	25.8	20.5
	Base	–	1864	35.8	25.0
8B	**Ours**	16×	279	56.7	30.3
	TOVA	16×	344	27.5	24.2
	Ours	12×	347	61.7	34.8
	Vanilla STILL	16×	367	40.8	26.5
	DMS	8×	381	62.5	34.1
	AM	16×	406	49.2	25.8
	Vanilla STILL	12×	425	43.3	27.3
	TOVA	12×	462	35.0	31.8
	Ours	8×	474	58.3	36.4
	AM	12×	476	53.3	26.5
	Vanilla STILL	8×	558	54.2	34.1
	AM	8×	601	58.3	28.0
	TOVA	8×	684	51.7	35.6
	Base	–	2755	66.7	40.2

Table 15: Results ordered by measured peak KV within each model size. Peak KV is the larger of the average per-completion peaks on AIME 26 and HMMT-Feb 26. This ordering shows that the same nominal compression ratio can produce substantially different peak KV. We show Ours are more memory efficient compared to static and learned baselines