MULTIPLE STAGES, MORE GAINS: PROACTIVE STAGED COMPRESSION FOR EFFICIENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The ever-growing Key-Value (KV) Cache generated during decoding becomes a bottleneck for large language models (LLMs) in long-context processing. Recent research has mainly focused on compressing the KV Cache through eviction and merging strategies, thereby enhancing long-context inference capability. However, most existing methods rely on aggressive compression at the memory limit, resulting in cumulative attention perturbation and model performance degradation. In this paper, we propose **PSC** (**P**roactive **S**taged **C**ompression), a compressor-agnostic scheduling framework that distributes KV Cache compression across multiple stages before the cache reaches its capacity limit. Our key insight is that decomposing compression into multiple stages with small removal ratios significantly reduces attention perturbation. From the cache capacity, prompt length and target retention ratio, we analytically derive the number of compression stages and the retained KV Cache size at each stage to obtain a closed-form two-phase compression schedule. Experimental results across various models and datasets show that incorporating PSC significantly enhances the performance of existing representative compression methods, achieving consistent accuracy improvement and accelerating inference by up to 1.17$\times$ compared to representative baselines.

1 INTRODUCTION

Large language models (LLMs) (Brown et al., 2020; Zhao et al., 2023; Godbole et al., 2025; Wang et al., 2024; Hannan et al., 2025; Zhang et al., 2025) demonstrate outstanding performance across various domains. When processing complex tasks, LLMs rely on the KV Cache to store the key-value pairs of all previous tokens (Rehg, 2024; Zhang et al., 2023; Behnam et al., 2025; Devoto et al., 2025), avoiding redundant computation and accelerating inference. However, the memory footprint of KV Cache grows linearly with the sequence length. In long-context scenarios, this can exceed hardware memory constraints, ultimately causing inference to halt or produce incomplete outputs.

To alleviate the memory overhead of the KV Cache, various compression techniques have been proposed. Two prevalent strategies are eviction-based (Liu et al., 2023; Zhang et al., 2023; Wang et al., 2025; Xiao et al., 2024; Yuan et al., 2024) and merging-based methods (Zhang et al., 2024; Wan et al., 2025). They reduce the memory footprint by discarding selected entries or combining them with retained representations. StreamingLLM (Xiao et al., 2024) retains initial attention sinks and recent tokens to support streaming inference. CaM (Zhang et al., 2024) and D2O (Wan et al., 2025) merge evictable entries into representative ones to preserve their information. Other approaches exploit redundancy across layers (Liu et al., 2024a) or learn adaptive memory compression (Nawrot et al., 2024). Token merging has also been studied in vision Transformers (Bolya et al., 2022). Recent KV Cache methods further explore compression at the level of individual layers and attention heads (Fu et al., 2025; Yang et al., 2024; Zhou et al., 2024; Park et al., 2025; Tian et al., 2026).

Despite their differences, prior studies generally adopt a reactive strategy that triggers compression only when the KV Cache reaches its capacity limit. We refer to this paradigm uniformly as **Capacity Triggered Compression (CTC)**. In long generation scenarios, this forces repeated large-scale compressions to accommodate incoming tokens. Consequently, even carefully designed compression schemes suffer from progressive degradation in generation quality.

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

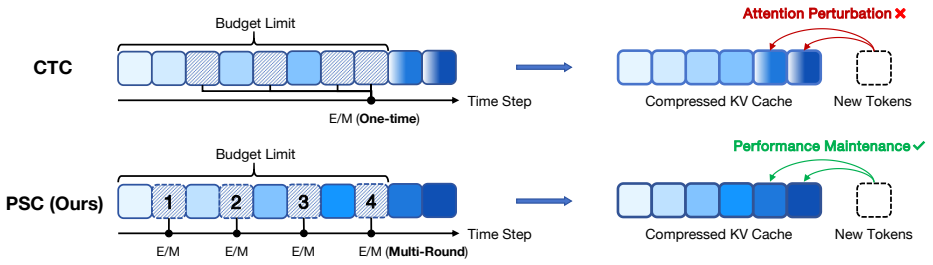

Figure 1: Comparison between CTC and PSC under the same compression ratio of 0.5. Dashed boxes indicate compressed KV Cache entries, where E/M denotes Eviction/Merging. CTC performs a single large-scale compression upon reaching the memory budget, whereas PSC distributes the compression across evenly spaced decoding steps, enabling multi-round selection of more important semantic information throughout generation.

To mitigate the performance degradation on model performance caused by CTC, we make the first attempt to explore an improved KV Cache compression mechanism from the perspective of macroscopic compression scheduling. Our analysis shows that **proactive and frequent compression** reduces attention score perturbation after compression compared to traditional methods. We therefore propose **Proactive Staged Compression (PSC)**, a model-agnostic scheduling framework that enables the model to dynamically filter and retain valid information more precisely during decoding. Figure 1 demonstrates that during inference, PSC decomposes conventional large-scale single compression into progressive and small compressions. This approach preserves critical semantic information more thoroughly and decelerates the rate at which the KV Cache reaches its memory budget, functioning as a significant scheduling optimization for traditional compression techniques as shown in Figure 2. PSC also seamlessly integrates with existing eviction- or merging-based methods to effectively enhance their performance.

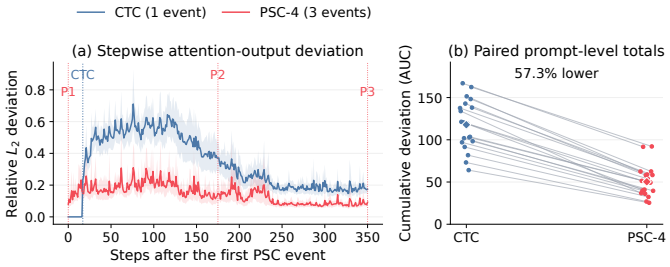

Figure 2: Attention perturbation under CTC and PSC with D2O on 20 GovReport prompts. (a) After CTC's single compression event, its attention-output perturbation is generally higher than PSC's over subsequent decoding steps. (b) PSC yields lower cumulative perturbation than CTC on all 20 prompts, with a 57.3% reduction in the mean.

Specifically, PSC partitions the generation process into two recurring phases delimited by cache saturation events and derives closed-form scheduling intervals from the cache capacity, the prefill length and the base retention ratio. Each compression round removes only a small fraction of the tokens that a conventional method would discard in a single pass, so the cumulative compression budget across all rounds remains equivalent to that of the traditional approach. The main contributions of this work are as follows:

- We identify compression scheduling as an overlooked yet consequential dimension of KV Cache compression design. Experimental analysis shows that, under an equivalent total compression budget, proactive and frequent compression incurs strictly smaller attention score perturbations than CTC, which motivates us to explore a novel KV Cache compression framework.
- We propose PSC, a model-agnostic scheduling framework that decomposes a conventional large-scale compression event into multiple lightweight fractional rounds at analytically derived intervals. PSC derives a two-phase schedule from the prompt length, cache capacity, and retention budget, without changing the underlying compressor's selection or merging mechanism. It can therefore be integrated with representative methods based on eviction or merging.

- Experimental results demonstrate that PSC improves task performance by up to 69.4% and inference throughput by up to 16.9% over the corresponding CTC baselines, highlighting the advantage of PSC in improving model quality while maintaining competitive inference efficiency.

2 Related Work

Eviction and Merging. Quantization reduces the KV Cache memory footprint by representing cached values at lower precision (Dong et al., 2024; Liu et al., 2024c; Hooper et al., 2024), while CacheGen compresses and streams cached representations to accelerate context loading (Liu et al., 2024b). Related work also studies weight quantization (Lin et al., 2024) and the effect of low-bit quantization on mathematical reasoning (Li et al., 2025). Eviction and merging instead reduce memory by shortening the effective sequence length. StreamingLLM (Xiao et al., 2024) retains initial attention sinks and recent tokens, whereas H2O (Zhang et al., 2023) retains a combination of recent tokens and heavy hitters identified from attention scores. FastGen (Ge et al., 2024) adapts cache retention to the attention patterns of individual heads. To mitigate information loss from eviction, CaM (Zhang et al., 2024) merges selected entries into retained representations. D2O (Wan et al., 2025) combines discriminative eviction and merging operations for efficient inference.

Multi-granularity Compression. Recent studies have explored more fine-grained compression strategies by exploiting heterogeneity across layers and attention heads (Wu et al., 2025a; Fu et al., 2025; Park et al., 2025; Tian et al., 2026; Wu et al., 2025b). Rather than applying uniform compression policies, these approaches adapt compression decisions to individual layers or heads according to their distinct distributional characteristics. PyramidInfer (Yang et al., 2024) observes that deeper layers contribute less to model inference and proposes a pyramid-shaped KV Cache, where more cache entries are retained in the shallow layers, while the retention ratio gradually decreases with depth. KeepKV (Tian et al., 2026) builds on layer-wise compression strategy of PyramidInfer and further proposes periodic lossless Compression to reduce information loss during cache merging. KeyDiff (Park et al., 2025) departs from reliance on attention score ranking, and instead compresses KV Cache purely based on similarity across different layers and heads.

However, the methods mentioned above focus primarily on reducing attention perturbation within a single compression operation. In practical scenarios, long-form text generation and multi-turn dialogues repeatedly drive the KV Cache to its capacity limit, necessitating successive rounds of compression (Rehg, 2024; Wan et al., 2025; Tian et al., 2026). Under the conventional reactive strategy of compressing only upon cache saturation, model performance degrades cumulatively with each compression pass, while inference speed deteriorates as sequence length grows. To address this, we propose PSC, a model-agnostic framework for multi-stage KV Cache compression that can be seamlessly integrated with any eviction or merging method, improving post-compression inference accuracy while preserving model throughput.

3 Methodology

3.1 Preliminary

KV Cache and Memory Budget. During auto-regressive decoding of LLMs, the KV Cache grows monotonically with the number of generated tokens. For a transformer model with h attention heads, ℓ layers, and hidden dimension d, the memory footprint of the KV Cache at decoding step t is:

$$\mathcal{M}(t) = 2 \cdot h \cdot \ell \cdot t \cdot d \cdot \text{sizeof}(\text{dtype}) \tag{1}$$

Since h, ℓ, and d are fixed constants for a given model, $\mathcal{M}(t)$ scales *linearly* with the sequence length t. Therefore, enforcing a memory budget is equivalent to enforcing a **maximum sequence length** L_m: once the number of cached token positions reaches L_m, compression must be triggered to prevent out-of-memory (OOM) failures (Zhang et al., 2024; Rehg, 2024). We adopt L_m as the sole budget parameter throughout this work.

Compression at Budget Saturation. Suppose the cache reaches capacity L_m at decoding step t. The current query $\mathbf{q}_t \in \mathbb{R}^d$ attends to all stored keys $\mathbf{K}_t \in \mathbb{R}^{L_m \times d}$ and values $\mathbf{V}_t \in \mathbb{R}^{L_m \times d}$ to

compute attention scores, which serve as the importance signal for selecting which KV entries to retain or discard. First, the attention score of each historical token is computed:

$$\alpha_i = \frac{\exp(\mathbf{q}_t \mathbf{k}_i^\top)}{\sum_{j=1}^{t} \exp(\mathbf{q}_t \mathbf{k}_j^\top)}. \tag{2}$$

Using a threshold τ, historical KV entries are partitioned into a *preservation set* $\mathcal{K}_m$ and an *eviction set* $\mathcal{K}_n$:

$$\begin{aligned} \mathcal{K}_m &= \{\, \mathbf{k}_i \mid \alpha_i \geq \tau \,\}, \\ \mathcal{K}_n &= \{\, \mathbf{k}_i \mid \alpha_i < \tau \,\}. \end{aligned} \tag{3}$$

For each evicted key $\mathbf{k}_n \in \mathcal{K}_n$, the most similar preserved key $\mathbf{k}_m \in \mathcal{K}_m$ (e.g., measured by cosine similarity) is selected for merging. The preserved entry is updated as:

$$\begin{aligned} \mathbf{k}_m &\leftarrow \lambda_{k,m}\, \mathbf{k}_m + \lambda_{k,n}\, \mathbf{k}_n, \\ \mathbf{v}_m &\leftarrow \lambda_{v,m}\, \mathbf{v}_m + \lambda_{v,n}\, \mathbf{v}_n, \end{aligned} \tag{4}$$

where $\lambda_{k,m}, \lambda_{k,n}$ control the merge strength for keys, and $\lambda_{v,m}, \lambda_{v,n}$ control the merge strength for values. The predominant paradigm for KV Cache management under a fixed memory budget follows a *Capacity Triggered Compression* (CTC) strategy: KV Cache grows freely during auto-regressive decoding until it reaches the maximum capacity L_m, whereupon a single compression event reduces it to RL_m $(0 < R < 1)$ tokens by compressing those with the lowest importance scores (Xiao et al., 2024; Zhang et al., 2024; 2023; Wan et al., 2025; Tian et al., 2026; Park et al., 2025). Despite their diversity in how tokens are scored for eviction or merging, compression fires exactly once per budget cycle, at the moment the cache saturates.

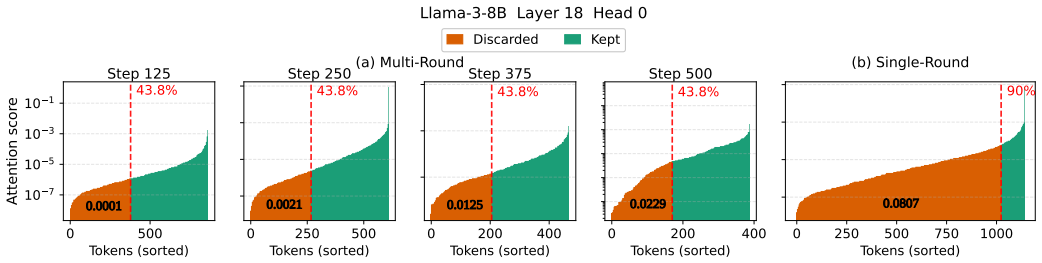

Figure 3: Under the same total compression budget, (a) distributing compression across multiple rounds incurs a cumulative attention loss of 0.0376, whereas (b) single-round aggressive pruning incurs 0.0807.

3.2 Motivation

Single-Snapshot Scoring Leads to Suboptimal Compression. Figure 3 demonstrates that deferring all compression decisions to the moment of cache saturation introduces a fundamental limitation. All tokens must be scored and compressed in a single pass from a frozen snapshot of the attention distribution, forcing the model to discard tokens to meet a fixed budget regardless of actual importance. This sweeps away moderately important tokens alongside truly redundant ones simply to meet the budget constraint.

Incremental Compression Enables More Selective Token Removal. In contrast, distributing the same compression workload across multiple smaller steps allows each decision to be made using a more up-to-date attention distribution. As shown in Figure 3, the tokens compressed by CTC carry a total attention score of 0.0807, whereas our incremental approach removes tokens with a total attention score of only 0.0376. This represents a 53.4% reduction in the cumulative attention score of compressed tokens, while keeping both the total number of removed tokens and the underlying scoring criterion unchanged. This observation motivates a straightforward redesign of the compression schedule: rather than waiting for the cache to saturate and performing one large compression, we split the bulk compression into multiple smaller compression operations that are executed proactively before the cache limit is reached. Further analysis is provided in Appendix C.

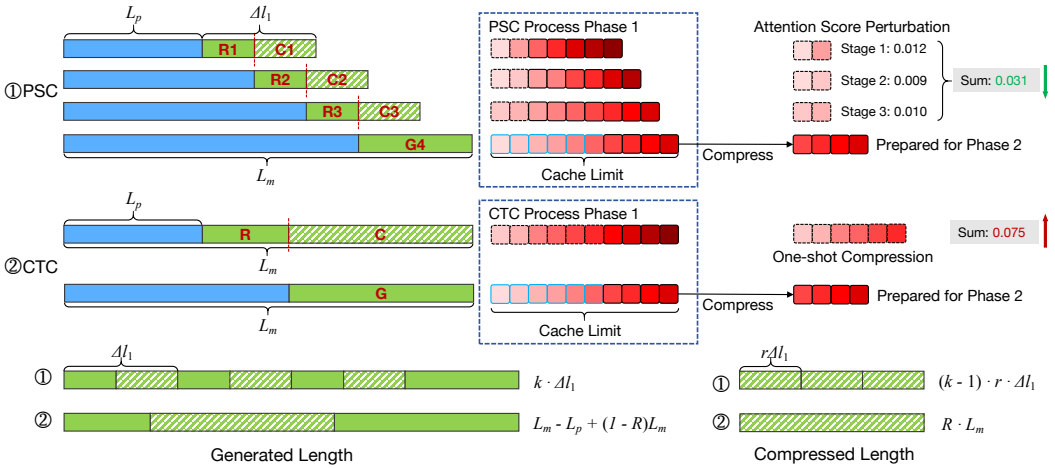

Figure 4: Overview of CTC and PSC Framework. **R**, **C** and **G** denote Retained, Compressed and Generated, respectively. When the decoding length far exceeds the cache budget, both CTC and PSC consist of two phases. In **Phase 1**, by the time PSC's cache first reaches L_m, it has already compressed $(1-R)L_m$ tokens incrementally; the corresponding CTC trajectory covers $L_p \to L_m$, applies one full compression, and then generates back to L_m. In **Phase 2**, one PSC cycle of $RL_m \to L_m$ (with incremental compression throughout) corresponds to *two* CTC compression events: $RL_m \to L_m \to RL_m \to L_m$.

3.3 Method: Proactive Staged Compression

3.3.1 Overview

Both CTC and PSC consist of two phases. In Phase 1, the sequence grows from the prefill length L_p to the maximum cache capacity L_m, after which a compression is applied to reduce the sequence to RL_m. In Phase 2, the model repeatedly generates from RL_m back to L_m and compresses KV Cache again. The key distinction lies in *when* compression is applied. CTC compresses reactively upon cache saturation: each time the cache reaches L_m, it compresses $(1-R)L_m$ tokens in a single pass. PSC, by contrast, distributes this compression budget across $k-1$ incremental steps during decoding, compressing a small fraction of tokens at regular intervals rather than all at once. This distinction induces a phase offset between the two strategies as shown in Figure 4.

3.3.2 Objective and Derivation of PSC

The core objective of PSC is to derive the compression step size (i.e., the number of tokens generated between successive compressions) and the compression ratio per round for each phase independently. We denote these as Δl_1, r_1 for Phase 1 and Δl_2, r_2 for Phase 2, respectively. The derivation is presented below, and the full PSC algorithm is provided in Appendix B.

Phase 1: Prefill to Budget ($L_p \to L_m$). Let L_i denote the cache size immediately after the i-th PSC compression. Beginning from $L_0 = L_p$, each step decodes Δl_1 tokens and immediately compresses $r_1 \cdot \Delta l_1$ of the current sequence, giving the recurrence

$$L_i \;=\; L_{i-1} + (1 - r_1)\,\Delta l_1, \tag{5}$$

where $i = 1, \ldots, k-1$. A final decoding step (without compression) fills the cache to the budget: $L_{k-1} + \Delta l_1 = L_m$. Expanding $L_{k-1} = L_p + (k-1)(1-r_1)\,\Delta l_1$ obtains the step size:

$$\Delta l_1 \;=\; \frac{L_m - L_p}{(k-1)(1-r_1) + 1}. \tag{6}$$

Two constraints jointly fix r_1. First, the total tokens decoded in Phase 1 equal the cache growth plus cumulative eviction:

$$k\,\Delta l_1 \;=\; (L_m - L_p) + (1 - R)\,L_m. \tag{7}$$

Second, the aggregate eviction across all $k-1$ PSC steps must match the CTC budget:

$$(k-1)\, r_1\, \Delta l_1 \;=\; (1-R)\, L_m. \tag{8}$$

Substituting equation 6 into equation 8 gives

$$r_1 = \frac{k(1-R)\, L_m}{(k-1)[(L_m - L_p) + (1-R)\, L_m]}. \tag{9}$$

Hence, Phase 1 decomposes the single bulk compression of $(1-R)L_m$ tokens into $k-1$ earlier, smaller compressions that collectively compress the identical volume.

Phase 2: Steady-State Cycle ($R\, L_m \to L_m$). Upon first reaching L_m, a standard CTC compression reduces the cache to RL_m, initiating a cycle that repeats for the remainder of decoding. Phase 2 has the same structure as Phase 1 with L_p replaced by the post-compression residual $R\, L_m$, yielding

$$\Delta l_2 \;=\; \frac{(1-R)\, L_m}{(k-1)(1-r_2)+1}. \tag{10}$$

The two analogous constraints become

$$k\, \Delta l_2 = 2(1-R)\, L_m, \tag{11}$$

$$(k-1)\, r_2\, \Delta l_2 = (1-R)\, L_m. \tag{12}$$

Dividing equation 12 by equation 11 and solving:

$$r_2 \;=\; \frac{k}{2(k-1)}. \tag{13}$$

Equation equation 13 has a notable property: r_2 depends solely on the number of rounds k and is entirely independent of the retention ratio R, the cache budget L_m, and the prefill length L_p. This universality renders the steady-state schedule hyperparameter-free with respect to the deployment context. Practitioners only need to select k once, after which the compression rate per-step is automatically determined for any model or task. In practice, each compression operation involves attention score computation, sorting, and similarity matrix calculation, which incurs non-trivial additional computational overhead. To minimize this overhead, we restrict the number of compression rounds by setting k to a small value, such as $k = 3$ and $k = 4$. Detailed empirical analysis is provided in our experiments.

4 Experiments

4.1 Experimental Setup

Tasks. To validate the effectiveness of PSC during the decoding phase, we select benchmarks that demand long output sequences, spanning three categories. For long document summarization, we adopt three subsets from LongBench (Bai et al., 2024): GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), and Multi-News (Fabbri et al., 2019). For open-ended text generation, we use WritingPrompts (Fan et al., 2018) and ROCStories (Mostafazadeh et al., 2016) to assess generation quality over extended outputs. For mathematical reasoning, we include MATH-500 (Lightman et al., 2024) and GSM8K (Cobbe et al., 2021) to evaluate the impact of PSC on long chain-of-thought reasoning.

Models and Baselines. Experiments are conducted on the Llama (Grattafiori et al., 2024) and Qwen (Qwen Team et al., 2024) model series. We compare against representative KV Cache compression methods, including eviction-based approaches StreamingLLM (Xiao et al., 2024) and H2O (Zhang et al., 2023), merging-based approaches CaM (Zhang et al., 2024) and D2O (Wan et al., 2025), and layer-wise and head-wise compression methods KeyDiff (Park et al., 2025) and KeepKV (Tian et al., 2026). By applying PSC on top of each baseline, we demonstrate its consistent effectiveness across diverse compression strategies.

Implementation Details. All experiments are conducted on NVIDIA A100 80GB GPUs. For each dataset, we configure the prefill length L_p, decoding length L_d, and the maximum KV Cache size L_m according to its characteristic input and output lengths, enabling a direct comparison between the CTC and PSC scheduling strategies. We sweep KV Cache budgets from 5% to 100% to evaluate PSC across a wide range of compression ratios. We also measure throughput and inference latency to quantify the overhead introduced by PSC's multi-round compression relative to the baselines. Finally, across different models and datasets, we vary k from 3 to 5 to study the trade-off between accuracy and throughput under different numbers of compression rounds. Further details on experimental deployment and measurement procedures are provided in Appendix E.

4.2 Performance Evaluation of PSC

4.2.1 Model Accuracy

Table 1 demonstrates performance of KV Cache compression methods on different models at 20% KV Cache budget. PSC consistently improves model performance over CTC across different methods and datasets.

Table 1: Performance of KV Cache compression methods on different models at a 20% KV Cache budget ($R = 0.2$). Summarization and long-context generation are evaluated with Rouge-L (%), and mathematical reasoning with accuracy (%). Boldface indicates the better result within each CTC–PSC pair.

		Summarization			Long-Context Generation		Mathematical Reasoning		
Method	**Setting**	**GovReport**	**QMSum**	**MultiNews**	**WritingPrompts**	**ROCStories**	**MATH-500**	**GSM8K**	**Avg**
Llama-3.1-8B-Instruct									
Full Cache	/	32.27	22.09	27.06	18.15	16.82	38.40	73.86	32.66
StreamingLLM	CTC	28.11	21.58	26.64	**18.10**	**16.21**	35.20	72.04	31.13
	PSC	**31.74**	**21.88**	**27.28**	18.01	16.06	**37.60**	**73.86**	**32.35**
H2O	CTC	29.44	21.69	26.98	17.95	**17.00**	32.80	72.04	31.13
	PSC	**31.54**	**22.07**	**27.10**	**18.11**	15.81	**36.80**	**72.64**	**32.01**
CaM	CTC	26.51	**21.36**	26.31	**17.53**	16.71	34.40	**71.73**	**30.65**
	PSC	**29.30**	21.17	**27.48**	16.58	**17.62**	**36.00**	61.09	29.89
D2O	CTC	20.50	19.19	21.67	15.95	**15.78**	30.40	65.05	26.93
	PSC	**32.21**	**21.88**	**27.36**	**18.44**	15.42	**35.20**	**72.64**	**31.88**
KeyDiff	CTC	27.57	21.58	26.48	**18.17**	14.82	37.60	73.25	31.35
	PSC	**31.17**	**22.19**	**27.26**	18.13	**17.23**	**39.20**	**73.56**	**32.68**
KeepKV	CTC	16.17	12.11	24.42	17.11	16.36	33.60	70.21	27.14
	PSC	**27.40**	**20.09**	**27.24**	**17.61**	**17.28**	**37.60**	**73.56**	**31.54**
Llama-3.2-3B-Instruct									
Full Cache	/	31.50	21.51	25.68	17.84	15.63	30.40	67.17	29.96
StreamingLLM	CTC	27.42	20.43	25.15	16.36	**15.52**	30.40	68.09	29.05
	PSC	**31.13**	**21.81**	**26.09**	**17.19**	15.34	**32.80**	**69.00**	**30.48**
H2O	CTC	28.42	21.29	25.28	17.41	**15.32**	30.40	66.57	29.24
	PSC	**30.75**	**21.67**	**25.97**	**17.57**	15.27	**36.00**	**67.78**	**30.72**
CaM	CTC	26.46	20.41	25.17	15.49	15.51	27.20	65.35	27.94
	PSC	**30.59**	**21.19**	**25.72**	**16.06**	**15.79**	**32.00**	**69.00**	**30.05**
D2O	CTC	15.78	17.85	19.22	13.25	**15.23**	27.20	59.88	24.06
	PSC	**30.91**	**21.64**	**25.74**	**17.09**	14.65	**30.40**	**66.57**	**29.57**
KeyDiff	CTC	27.55	**21.56**	25.43	16.17	**15.46**	28.80	65.65	28.66
	PSC	**30.63**	21.26	**25.84**	**16.78**	15.34	**32.80**	**65.96**	**29.80**
KeepKV	CTC	19.65	16.23	24.72	16.13	**15.46**	32.00	65.96	27.16
	PSC	**29.53**	**20.93**	**25.88**	**16.85**	14.51	**34.40**	**66.87**	**29.85**

Summarization Tasks. PSC exhibits a pronounced advantage on three summarization tasks of Longbench across different models and datasets. On Llama-3.1-8B-Instruct, D2O achieves over **57%** improvement on GovReport, and KeepKV recovers scores by nearly **70%** and **66%** on GovReport and QMSum respectively. All models and methods achieve consistent improvements, which we attribute to PSC's ability to enable further information filtering based on the partially generated summary during the decoding phase.

Long-Context Generation. On text generation benchmarks, PSC also yields notable gains compared to CTC, with D2O improving by over **15%** on WritingPrompts and KeyDiff gaining over **16%**

on ROCStories. In open-ended text generation, PSC still demonstrates its ability to reduce attention perturbation, retaining more useful information and improving output quality.

Mathematical Reasoning. Chain-of-thought reasoning requires the model to maintain coherent multi-step inference traces throughout decoding. The proactive staged compression of PSC preserves more of these critical context tokens, translating into consistent accuracy gains across reasoning benchmarks. Overall, PSC achieves higher average scores across all methods, confirming its effectiveness as a general-purpose enhancement to existing KV Cache compression strategies.

4.2.2 Performance Evaluation of PSC-k

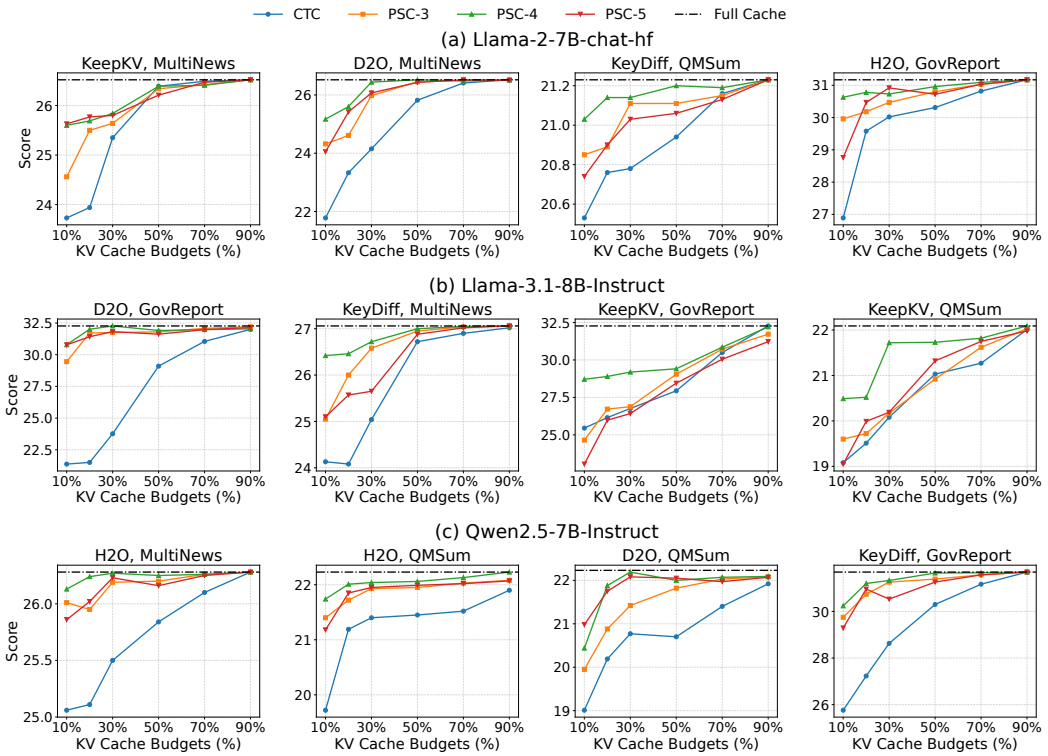

Figure 5: Rouge-L scores on summarization tasks of Longbench under four representative KV Cache compression methods on Llama and Qwen models. Each subplot compares CTC against PSC using different rounds (PSC-3/4/5) across budget levels from 10% to 90%, with the Full Cache score shown as a dashed baseline. PSC consistently closes the gap to Full Cache, especially at tight budgets.

Figure 5 compares PSC-3, PSC-4, and PSC-5 across models, compression methods, and KV Cache budgets. PSC-4 provides a practical default in these experiments, although the strongest setting varies across configurations and additional rounds do not consistently improve performance. PSC-4 remains competitive across the evaluated budgets, with particularly pronounced gains over CTC at a 10% budget. These results highlight the benefit of staged compression under tight memory constraints, where distributing token removal across decoding steps can mitigate the information loss associated with aggressive compression in a single pass.

4.2.3 Inference Efficiency and Operator Profiling

Table 2 evaluates Llama-3.1-8B-Instruct on an NVIDIA A100 80GB GPU using 100 GovReport samples per run. The prefill length, maximum decoding length, and KV Cache capacity are 4096, 512, and 4200 tokens, respectively. We compare CTC with PSC-3 through PSC-6 at 10%, 20%, and 50% retention budgets. Additional compression events do not produce a proportional increase in inference cost. At a 10% budget, PSC-6 improves StreamingLLM throughput by 7.4% and KeepKV throughput by 16.9% over CTC. Many other configurations retain comparable throughput, although some CaM

and D2O settings are slower and average latency generally increases modestly. For compressors with pairwise scoring, the total staged cost scales as $O(\sum_j n_j^2)$, where n_j is the cache length at each event. Smaller intermediate caches can therefore offset part of the additional invocation cost.

Table 2: Inference efficiency on an NVIDIA A100 80GB GPU with Llama-3.1-8B-Instruct and 100 GovReport samples per run. Boldface marks the best value and underlining marks the second-best value in each row.

(a) Throughput (tokens/s)

Method	Budget	CTC	PSC-3	PSC-4	PSC-5	PSC-6
StreamingLLM	10%	25.36	25.63	26.13	27.04	**27.24**
	20%	26.11	26.20	26.73	26.98	**27.60**
	50%	27.27	**27.56**	27.43	27.26	27.44
H2O	10%	19.26	18.70	19.20	19.11	**19.66**
	20%	19.42	18.94	19.25	19.34	**19.55**
	50%	19.30	19.23	19.49	19.55	**19.59**
CaM	10%	**18.18**	17.16	17.05	16.18	17.82
	20%	18.46	18.58	**18.59**	16.32	16.45
	50%	19.08	19.18	**19.41**	19.30	18.90
D2O	10%	**18.34**	17.62	17.79	18.20	17.88
	20%	**18.53**	18.02	18.19	18.08	17.77
	50%	**18.59**	18.22	17.94	17.92	18.39
KeepKV	10%	21.89	22.85	24.13	25.21	**25.59**
	20%	22.87	24.82	25.54	25.96	**26.40**
	50%	**28.00**	27.19	27.55	27.68	27.04
KeyDiff	10%	25.79	25.76	**26.51**	25.95	25.74
	20%	26.27	26.37	26.26	27.10	**27.15**
	50%	26.92	27.00	**27.17**	27.13	26.71

(b) Per-Sample Latency (s)

Method	Budget	CTC	PSC-3	PSC-4	PSC-5	PSC-6
StreamingLLM	10%	**12.83**	13.20	13.08	13.07	13.15
	20%	**12.78**	13.04	13.15	13.18	13.05
	50%	**12.61**	12.96	13.10	13.03	13.01
H2O	10%	**17.88**	18.29	18.34	18.51	18.38
	20%	**17.93**	18.45	18.32	18.42	18.49
	50%	**18.03**	18.47	18.42	18.29	18.34
CaM	10%	**18.30**	18.45	18.33	18.54	18.35
	20%	18.38	18.42	**18.26**	18.60	18.64
	50%	**18.21**	18.41	18.22	18.48	18.69
D2O	10%	**18.60**	19.19	19.16	18.90	18.93
	20%	**18.42**	18.93	18.89	19.20	19.15
	50%	**18.63**	18.89	19.25	19.13	18.93
KeepKV	10%	**13.20**	13.70	13.68	13.77	13.78
	20%	**13.23**	13.62	13.92	13.91	13.60
	50%	**13.09**	13.56	13.80	13.80	13.59
KeyDiff	10%	**13.14**	13.41	13.37	13.43	13.37
	20%	**12.92**	13.28	13.45	13.21	13.28
	50%	**12.95**	13.15	13.25	13.21	13.34

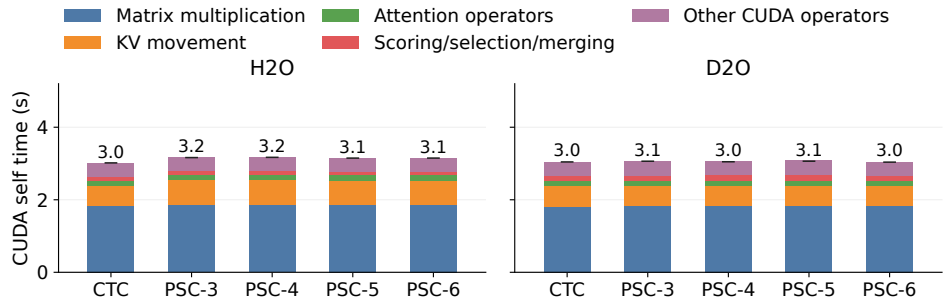

Figure 6: CUDA operator-time breakdown for CTC and PSC-3–PSC-6 on GovReport at a 10% KV Cache budget (NVIDIA A100 80GB).

Figure 6 shows that the total CUDA operator time across PSC's compression stages remains close to that of CTC. H2O requires approximately 3.0–3.2 seconds under PSC, compared with 3.0 seconds under CTC, and the time spent in the major operator categories is broadly similar. Thus, distributing compression across multiple stages does not multiply the computational overhead or introduce a substantial slowdown in this profiled workload. The additional compression calls are offset in part by processing smaller intermediate caches. D2O also remains close to its CTC control, although its fallback behavior in this configuration limits the interpretation of this comparison.

5 Conclusion

From the perspective of compression scheduling, we propose PSC, a model-agnostic compression strategy that decomposes the traditional compression triggered upon reaching the memory limit into multiple smaller compression operations performed proactively in advance. Unlike prior methods that focus solely on *what* to compress, PSC addresses *when* to compress, and can be integrated into existing eviction- and merging-based approaches, achieving significant improvements in model performance and preserving inference efficiency across diverse tasks and models. We believe that PSC can provide a new perspective for KV Cache compression, inspiring future research to explore better compression mechanisms from the standpoint of decoding scheduling.

AI USE STATEMENT

We used generative AI tools to assist with language polishing, LaTeX formatting and figure preparation. The research ideas, core methodology, experimental design, execution of experiments, and scientific conclusions were developed and determined by the authors. The authors reviewed and verified the AI-assisted text, code, and figures and take full responsibility for the final content and results of this work.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A Bilingual, Multitask Benchmark for Long-Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating Long-Context LLM Inference via Two-Stage KV Cache Compression. In *Forty-Second International Conference on Machine Learning*, 2025.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT But Faster. *arXiv preprint arXiv:2210.09461*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

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.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. QAQ: Quality Adaptive Quantization for LLM KV Cache. *arXiv preprint arXiv:2403.04643*, 2024.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-News: A Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. In *International Conference on Learning Representations*, 2025.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. In *International Conference on Learning Representations*, 2024.

Aditi Godbole, Jabin Geevarghese George, and Smita Shandilya. Leveraging Long-Context Large Language Models for Multi-Document Understanding and Summarization in Enterprise Applications. In *International Conference on Business Intelligence, Computational Mathematics, and Data Analytics*, pp. 208–224, 2025. doi: 10.1007/978-3-031-87511-3_15.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783v3.

Tanveer Hannan, Md Mohaiminul Islam, Jindong Gu, Thomas Seidl, and Gedas Bertasius. ReVisionLLM: Recursive Vision-Language Model for Temporal Grounding in Hour-Long Videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *Advances in Neural Information Processing Systems*, 2024.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient Attentions for Long Document Summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

Zhen Li, Yupeng Su, Runming Yang, Congkai Xie, Zheng Wang, Zhongwei Xie, Ngai Wong, and Hongxia Yang. Quantization Meets Reasoning: Exploring LLM Low-Bit Quantization Degradation for Mathematical Reasoning. *arXiv preprint arXiv:2501.03035*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. In *International Conference on Learning Representations*, 2024.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems*, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. *Advances in Neural Information Processing Systems*, 2024a.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024b.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. *Advances in Neural Information Processing Systems*, 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A Tuning-Free Asymmetric 2-Bit Quantization for KV Cache. *Proceedings of the 41st International Conference on Machine Learning*, 2024c.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. *Forty-First International Conference on Machine Learning*, 2024.

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. KeyDiff: Key Similarity-Based KV Cache Eviction for Long-Context LLM Inference in Resource-Constrained Environments. *Advances in Neural Information Processing Systems*, 2025.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024.

Isaac Rehg. KV-Compress: Paged KV-Cache Compression with Variable Compression Rates per Attention Head. *arXiv preprint arXiv:2410.00161*, 2024.

Noam Shazeer. Fast Transformer Decoding: One Write-Head Is All You Need. *arXiv preprint arXiv:1911.02150*, 2019.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *ACM Computing Surveys*, 2022.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. KeepKV: Achieving Periodic Lossless KV Cache Compression for Efficient LLM Inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 2017.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. D2O: Dynamic Discriminative Operations for Efficient Long-Context Inference of Large Language Models. In *International Conference on Learning Representations*, 2025.

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. LLMs Know What to Drop: Self-Attention Guided KV Cache Eviction for Efficient Long-Context Inference. *ICLR 2025 Workshop on Sparsity in LLMs (SLLM)*, 2025.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving Text Embeddings with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. SCOPE: Optimizing Key-Value Cache Compression in Long-Context Generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025a.

Wei Wu, Zhuoshi Pan, Kun Fu, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Zheng Wang, and Hui Xiong. TokenSelect: Efficient Long-Context Inference and Length Extrapolation for LLMs via Dynamic Token-Level KV Cache Selection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025b.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, 2024.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV Cache Compression for High-Throughput LLM Inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Yang Zhang, Hanlei Jin, Dan Meng, Jun Wang, and Jinghua Tan. A Comprehensive Survey on Process-Oriented Automatic Text Summarization with Exploration of LLM-Based Methods. *arXiv preprint arXiv:2403.02901*, 2025.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache Merging for Memory-Efficient LLMs Inference. In *Forty-First International Conference on Machine Learning*, 2024.

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

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223*, 2023.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. QMSum: A New Benchmark for Query-Based Multi-Domain Meeting Summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. DynamicKV: Task-Aware Adaptive KV Cache Compression for Long-Context LLMs. *arXiv preprint arXiv:2412.14838*, 2024.

APPENDIX

APPENDIX CONTENTS

Section	Page
A. Attention Mechanism	15
B. PSC Algorithm and Implementation Details	15
C. Diagnostic Experiments and Extended Ablations	16
C.1. Controlled Attention Perturbation Assay	16
C.2. Early Deletion Regret	18
D. Additional Experimental Results	20
D.1. Per-Budget Improvement Analysis	20
D.2. Context Summarization	20
D.3. Mathematical Reasoning	21
E. Experimental Deployment Details	22

A Attention Mechanism

The attention mechanism is central to Transformer-based LLMs (Vaswani et al., 2017; Shazeer, 2019; Tay et al., 2022). During auto-regressive decoding, each token is projected into query ($\mathbf{q}_t$), key ($\mathbf{k}_t$), and value ($\mathbf{v}_t$) vectors using learnable matrices $\mathbf{W_q}$, $\mathbf{W_k}$ and $\mathbf{W_v}$. To generate the next token, the model must attend to all preceding tokens, making recomputation of past keys and values highly inefficient. The KV Cache addresses this by storing historical k_t and v_t vectors, thereby avoiding redundant computation and serving as a critical component for efficient inference in long-context scenarios.

Without loss of generality, consider a single attention head in one Transformer layer. At decoding step t, let $\mathbf{X}_t = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t]$ denote the input hidden states, and $\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v} \in \mathbb{R}^{d\times d}$ the projection matrices. The corresponding queries, keys, and values are computed as:

$$\begin{aligned} \mathbf{K}_t &= \mathbf{X}_t\mathbf{W_k} = [\mathbf{k}_1, \ldots, \mathbf{k}_t], \\ \mathbf{V}_t &= \mathbf{X}_t\mathbf{W_v} = [\mathbf{v}_1, \ldots, \mathbf{v}_t], \\ \mathbf{q}_t &= \mathbf{x}_t\mathbf{W_q}. \end{aligned} \tag{14}$$

When KV Cache is employed, the key and value vectors from previous steps are preserved. At step t, the newly computed $\mathbf{k}_t$ and $\mathbf{v}_t$ are appended to the cache:

$$\begin{aligned} \mathbf{K}_t &= \mathrm{Concat}(\mathbf{K}_{t-1}, \mathbf{k}_t), \\ \mathbf{V}_t &= \mathrm{Concat}(\mathbf{V}_{t-1}, \mathbf{v}_t). \end{aligned} \tag{15}$$

The query $\mathbf{q}_t$ attends to all cached keys to produce the output:

$$\mathbf{O}_t = \sum_{i=1}^{t} \mathrm{softmax}\left(\frac{\mathbf{q}_t\mathbf{k}_i^\top}{\sqrt{d}}\right)\mathbf{v}_i. \tag{16}$$

While the KV Cache effectively eliminates redundant computation, its memory footprint grows linearly with the sequence length. As the context length increases, the resulting growth in memory consumption and attention computation progressively degrades inference throughput. Consequently, efficient KV Cache compression becomes essential for reducing memory overhead while maintaining model performance.

B PSC Algorithm and Implementation Details

Algorithm 1 summarizes the two phases of PSC. PSC determines when compression occurs and how many KV entries are removed, while the base compressor $\mathcal{C}$ determines which entries are evicted or merged. To preserve the total removal volume, we round cumulative targets rather than each event independently. Let q_i be the planned cumulative removal after event i, with $q_0 = 0$. Event i removes

$$d_i = \lfloor q_i \rfloor - \lfloor q_{i-1} \rfloor, \qquad \sum_{i=1}^{s} d_i = \lfloor q_s \rfloor. \tag{17}$$

For example, dividing ten removals across three events gives event sizes of three, three, and four. Any remaining rounding difference is assigned to the final event. If the proactive Phase 1 schedule is infeasible, PSC applies conventional compression at cache saturation.

In Phase 2, let $B = (1 - R)L_m$. Both policies begin at saturation and remove B entries, leaving RL_m entries. Over the next $2B$ generated tokens, PSC distributes another B removals across $k - 1$ events, whereas CTC removes them once at the next saturation point. Both therefore process the same continuation, remove the same total volume, and end with L_m entries. The final saturation action belongs to the next comparison window. For $k = 2$, PSC's single scheduled event occurs at saturation and removes B entries, making Phase 2 identical to CTC.

Algorithm 1 PSC Scheduling Framework with a Capacity-Triggered Fallback

Input: L_p, L_m, R, k, base compressor $\mathcal{C}$
Output: Compressed KV Cache throughout decoding

Compute r_1 via Equation 9, Δl_1 via Equation 6
if $L_m > L_p$ and $k \geq 1 + (1-R)L_m/(L_m - L_p)$ **then** ▷ feasible proactive Phase 1
  for $i = 1, \ldots, k-1$ **do**
    Decode Δl_1 tokens; apply $\mathcal{C}$ to cumulative target $ir_1\Delta l_1$
  end for
  Decode Δl_1 tokens ▷ cache reaches L_m
else
  Decode until L_m ▷ CTC fallback avoids cache overflow
end if
$r_2 \leftarrow k/[2(k-1)]$ ▷ independent of R, L_m, L_p
repeat ▷ Phase 2: steady-state
  Apply $\mathcal{C}$ to $(1-R)L_m$ entries at saturation
  Compute Δl_2 via Equation 10
  for $i = 1, \ldots, k-1$ **do**
    Decode Δl_2 tokens; apply $\mathcal{C}$ to cumulative target $ir_2\Delta l_2$
  end for
  Decode Δl_2 tokens ▷ cache reaches L_m
until end of sequence

C Diagnostic Experiments and Extended Ablations

C.1 Controlled Attention Perturbation Assay

Assay design. For each of 20 GovReport prompts, a full cache replay generates a fixed continuation. We replay the same tokens at identical absolute positions under full cache, D2O with CTC, and D2O with PSC-4. This experiment uses $L_p = 832$, $L_m = 1024$, a retention ratio of $R = 0.5$, and a continuation of 704 tokens. CTC removes 512 KV entries in one event when the cache reaches capacity. PSC-4 removes the same number over three proactive events. We verify the removal volume, event count, capacity bound, and final KV length for every prompt. Hidden queries may diverge after the cache trajectories separate. The comparison therefore measures perturbation across decoding trajectories rather than a causal effect for identical queries.

Perturbation metric. Let $o^{\pi}_{t,\ell,h}$ denote the attention output before projection for policy π at decoding step t, layer ℓ, and attention head h. Let $o^{\text{full}}_{t,\ell,h}$ denote the corresponding output from the full cache replay. We measure relative deviation as

$$P^{\pi}_{t,\ell,h} = \frac{\left\| o^{\pi}_{t,\ell,h} - o^{\text{full}}_{t,\ell,h} \right\|_2}{\left\| o^{\text{full}}_{t,\ell,h} \right\|_2 + \epsilon}. \tag{18}$$

We average the deviation across four sampled layers and all attention heads, then sum it over the common measurement window for each prompt. The measurement uses the actual compressed keys and values, including merged representations. In the D2O experiment, mean cumulative deviation is 117.83 for CTC and 50.30 for PSC-4. The mean paired difference between PSC-4 and CTC is -67.53, with a 95% bootstrap confidence interval over prompts of $[-74.43, -60.54]$. PSC-4 has lower cumulative deviation on all 20 prompts, yielding a mean reduction of 57.3% in this configuration.

We extend the assay to StreamingLLM, H2O, and CaM using Llama-3.1-8B-Instruct and 20 GovReport prompts at a 20% KV Cache budget. These experiments use $L_p = 608$ and $L_m = 1024$. For each $k \in \{3, 4, 5, 6\}$, CTC removes 820 KV entries in one event, while PSC-k distributes the same removal over $k-1$ proactive events. Each pair of CTC and PSC runs ends at the same KV length. We calculate each reduction against its paired CTC control. Because the measurement window changes with k, absolute cumulative deviations should not be compared across round counts.

Figures 7, 8, and 9 use the same visual conventions. Blue represents CTC and red represents PSC. Panels labeled a show stepwise deviation. Dotted vertical lines mark compression events, and shaded regions show 95% bootstrap intervals over prompts. Panels labeled b show cumulative deviation for individual prompts. Connecting lines identify paired CTC and PSC runs, and diamonds mark group means.

For StreamingLLM, PSC-3 through PSC-6 reduce mean cumulative deviation by 61.2%, 43.6%, 37.9%, and 35.7%, respectively. Figure 7 shows the stepwise trajectories and paired totals for individual prompts.

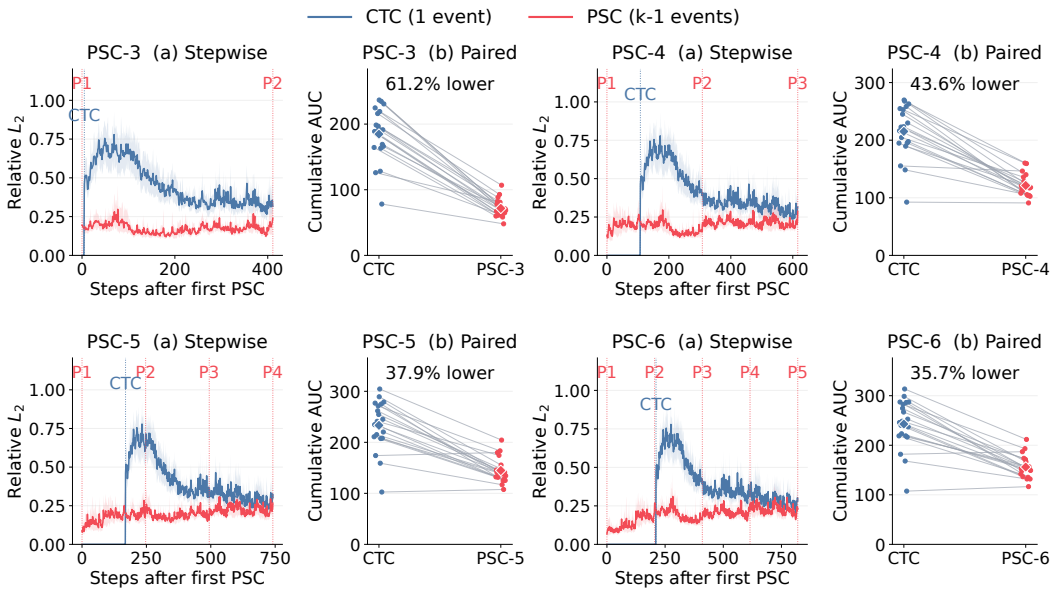

Figure 7: Attention output perturbation for StreamingLLM under CTC and PSC-3 through PSC-6 using Llama-3.1-8B-Instruct on 20 GovReport prompts at a 20% KV Cache budget.

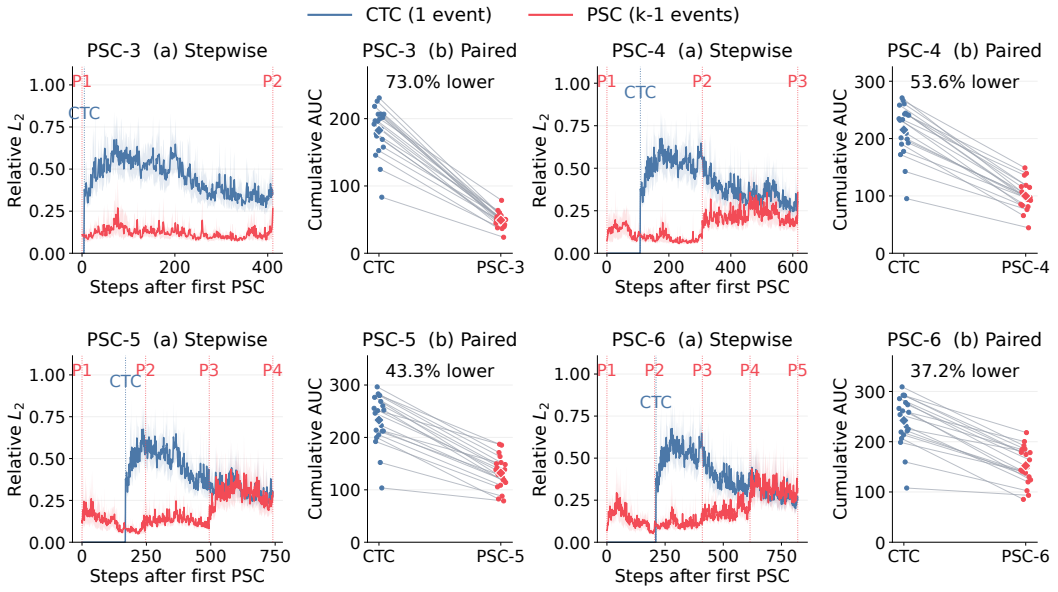

Figure 8: Attention output perturbation for H2O under CTC and PSC-3 through PSC-6 using Llama-3.1-8B-Instruct on 20 GovReport prompts at a 20% KV Cache budget.

For H2O, the corresponding reductions are 73.0%, 53.6%, 43.3%, and 37.2%. Figure 8 shows lower mean cumulative deviation under PSC at all four round counts.

For CaM, PSC-3 through PSC-6 reduce mean cumulative deviation by 59.2%, 35.7%, 29.9%, and 28.1%, respectively. Figure 9 shows the corresponding trajectories and paired totals. These experiments indicate that the reduction in attention output perturbation extends beyond D2O, although its magnitude depends on the base compressor and compression schedule.

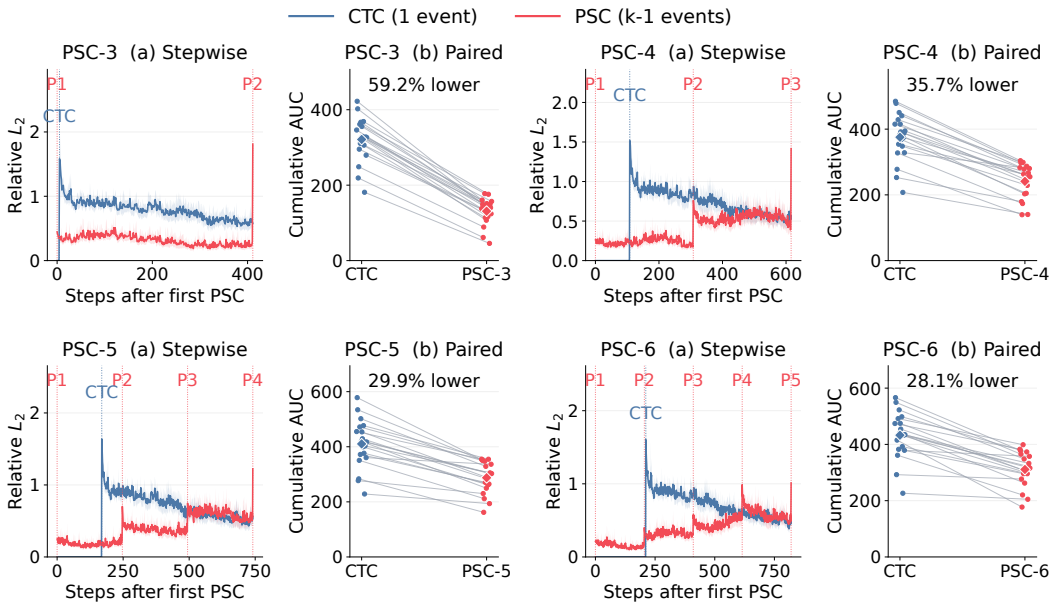

Figure 9: Attention output perturbation for CaM under CTC and PSC-3 through PSC-6 using Llama-3.1-8B-Instruct on 20 GovReport prompts at a 20% KV Cache budget.

C.2 Early Deletion Regret

Definition. We examine whether tokens removed during compression receive substantial attention later in decoding. For each prompt, we replay a fixed continuation under a full cache reference. Let t_{end} denote the common endpoint of a paired CTC and PSC comparison, and let $T = 64$ denote the length of the subsequent observation window.

Let $\mathcal{U}_{\ell,h}$ be the set of original token positions present at the endpoint in the full cache reference. For token $i \in \mathcal{U}_{\ell,h}$, we define its future attention score as

$$a^{\text{future}}_{i,\ell,h} = \frac{1}{T} \sum_{t=t_{\text{end}}+1}^{t_{\text{end}}+T} A^{\text{full}}_{t,\ell,h,i}, \tag{19}$$

where $A^{\text{full}}_{t,\ell,h,i}$ is the reference attention assigned to token i at decoding step t. For grouped query attention, we average the query heads associated with each KV head.

We rank the positions in $\mathcal{U}_{\ell,h}$ by their future attention scores and define the important set as

$$\mathcal{I}_{\ell,h} = \text{Top}_{\lceil 0.05|\mathcal{U}_{\ell,h}| \rceil} \left\{ a^{\text{future}}_{i,\ell,h} \mid i \in \mathcal{U}_{\ell,h} \right\}. \tag{20}$$

Here $\mathcal{I}_{\ell,h}$ denotes the token positions selected by this ranking. Let $\mathcal{E}^{\pi}_{\ell,h}$ be the set of original positions physically removed by policy π before or at the endpoint. The regret rate is

$$r^{\pi}_{\ell,h} = \frac{\left| \mathcal{E}^{\pi}_{\ell,h} \cap \mathcal{I}_{\ell,h} \right|}{\left| \mathcal{E}^{\pi}_{\ell,h} \right|}. \tag{21}$$

This quantity measures the fraction of removed positions that subsequently rank among the most attended positions under the reference trajectory. We average it across four sampled layers and all KV heads to obtain a regret rate r_s^π for each prompt s. The mean paired difference is

$$\Delta r = \frac{1}{S}\sum_{s=1}^{S}\left(r_s^{\mathrm{PSC}} - r_s^{\mathrm{CTC}}\right), \qquad S = 20. \tag{22}$$

A negative value indicates lower regret under PSC. Confidence intervals are obtained by resampling the paired prompts. For methods that merge tokens, physical removal is tracked separately from whether the original information remains represented in a retained token.

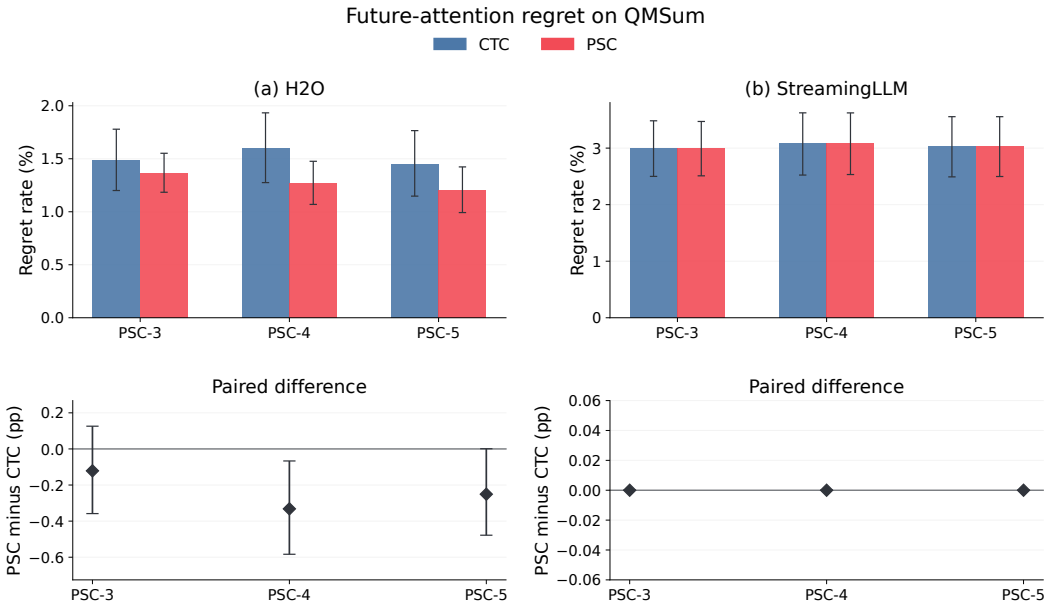

Figure 10: Future attention regret for H2O and StreamingLLM with PSC-3 through PSC-5 and paired CTC controls. The experiment uses Llama-3.1-8B-Instruct, 20 QMSum prompts, and a 20% KV Cache budget.

Experimental results. We evaluate H2O and StreamingLLM on 20 QMSum prompts using Llama-3.1-8B-Instruct at a 20% KV Cache budget. Figure 10 shows the regret rates and paired differences for PSC-3 through PSC-5. Blue bars represent CTC and red bars represent PSC. The lower panels report PSC minus CTC in percentage points. Error bars show 95% bootstrap intervals over prompts.

For H2O, the mean paired differences are -0.121, -0.332, and -0.251 percentage points for PSC-3, PSC-4, and PSC-5, respectively. Their corresponding confidence intervals are $[-0.358, 0.126]$, $[-0.584, -0.067]$, and $[-0.478, 0.001]$ percentage points. The PSC-4 interval lies entirely below zero, supporting a reduction in regret for this configuration. The PSC-3 and PSC-5 intervals include zero, so these comparisons remain inconclusive despite their lower mean estimates.

For StreamingLLM, CTC and PSC have identical measured regret rates at all three round counts. The mean rates are 3.01%, 3.08%, and 3.03% for the paired PSC-3, PSC-4, and PSC-5 comparisons. This equality indicates that the metric detects no difference between the schedules in these experiments.

Interpretation. These results provide limited evidence against an increase in regret under PSC in the evaluated configurations. They do not establish that PSC never removes important tokens prematurely. In particular, the reported metric includes all physically removed positions rather than only those removed before the corresponding CTC event. Equal regret rates can also conceal differences in which tokens are removed.

The importance criterion is based on reference attention over a finite observation window. It does not capture every form of token utility or directly measure the effect of deletion on generation quality. A

stronger claim would require an analysis restricted to tokens removed early, broader experimental coverage, and a predefined tolerance for an acceptable increase in regret.

D Additional Experimental Results

D.1 Per-Budget Improvement Analysis

Figure 11 compares PSC-4 with CTC across methods, cache budgets, and summarization datasets. The improvement is most concentrated at low budgets (10%–30%), where CTC's one-shot compression causes the greatest information loss. As the budget increases, the gap narrows and eventually converges near zero at 90%, consistent with both methods approaching Full Cache performance at high retention ratios. GovReport shows the strongest gains, reflecting the sensitivity of long-document summarization to compression scheduling.

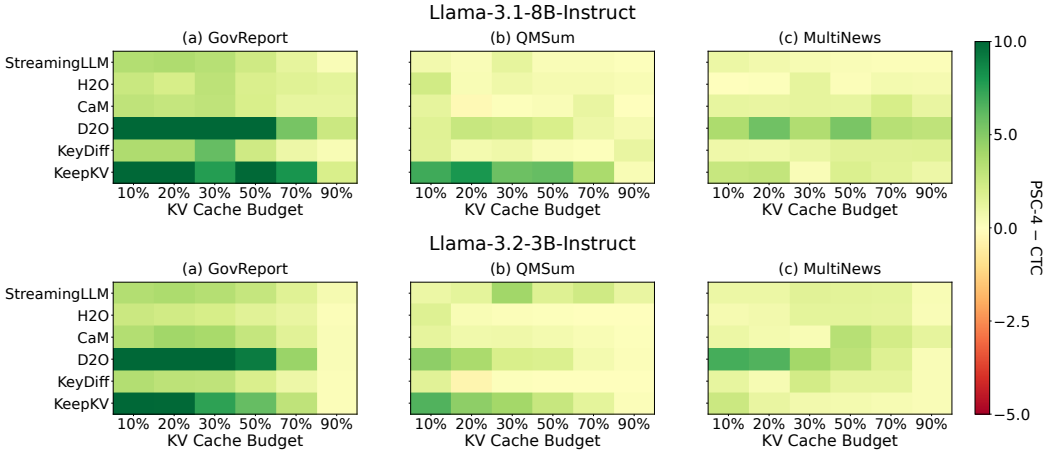

Figure 11: Cell-level PSC-4 improvement over CTC on Llama-3.1-8B-Instruct and Llama-3.2-3B-Instruct, across compression methods, budget levels, and datasets. Green indicates PSC-4 outperforms CTC and red indicates the reverse.

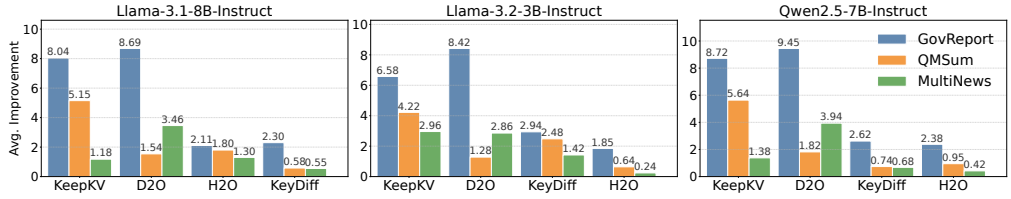

Figure 12: Average PSC improvement over CTC across four models, four compression methods, and three summarization datasets, averaged over KV Cache budgets from 10% to 90%.

D.2 Context Summarization

Overall Performance. Figure 12 presents the average PSC improvement over CTC across four models and four compression methods on three LongBench summarization benchmarks, namely GovReport, QMSum, and MultiNews. PSC yields gains across many configurations, with the degree of improvement varying notably across datasets and a small number of neutral or negative cells retained in the complete grid. GovReport benefits the most on average, while QMSum and MultiNews exhibit comparatively smaller margins.

Effect of Generation Length. One plausible explanation for this disparity is the target generation length of each dataset. GovReport requires substantially longer outputs, spanning a greater number of decoding steps over which PSC can apply its multi-stage compression schedule iteratively, each

time retaining a refined subset of the KV Cache that better reflects the current generative context. In contrast, QMSum and MultiNews involve shorter target summaries, which reduces the number of compression rounds and limits the cumulative benefit that staged compression can provide. This observation motivates a length-stratified analysis rather than establishing causality by itself. The final experiment will bin examples by realized output length and report paired PSC–CTC changes within each bin while controlling for dataset and retention budget.

D.3 Mathematical Reasoning

Performance under limited budgets. Figure 13 compares PSC accuracy gains over CTC on MATH-500 across six compression methods and four models. The largest gains generally occur at KV Cache budgets of 5% and 10%, with some configurations exceeding eight percentage points. These results suggest that distributing compression across decoding steps is particularly beneficial when the cache must be reduced aggressively.

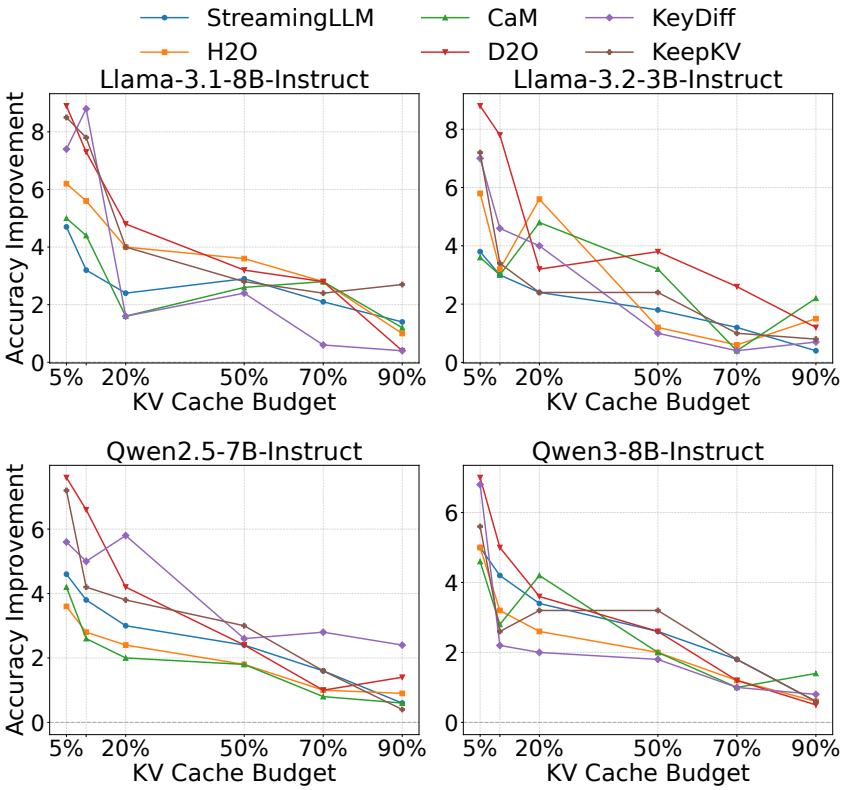

Figure 13: PSC accuracy gains over CTC on MATH-500 across four models, six compression methods, and KV Cache budgets from 5% to 90%.

As the budget increases, the advantage typically narrows because both schedules preserve more context. The trend is not strictly monotonic, however, as several methods show local increases at intermediate or high budgets.

Consistency across models and methods. Positive gains appear across both Llama and Qwen models and across eviction and merging methods, indicating that the benefit of PSC is not confined to a single model family or compression mechanism. The magnitude and relative ordering of these gains vary across configurations, suggesting that scheduling interacts with the underlying compression criterion. These accuracy results are consistent with PSC maintaining a more useful cache trajectory, but they do not establish the cause of the improvement. The attention perturbation and deletion regret analyses provide complementary evidence about how the compression schedule affects intermediate cache behavior.

E Experimental Deployment Details

Hardware and Paired Comparisons. All experiments are conducted on NVIDIA A100 GPUs with 80GB of memory. Within each comparison, CTC and PSC use the same model checkpoint, tokenizer, dataset subset, cache capacity, retention budget, and decoding configuration. PSC changes the compression schedule while retaining the selection and merging operations of the corresponding base compressor. The supplied experiment drivers use a random seed of 42 for sampling and random initialization.

Efficiency Configurations. The GovReport configuration uses a prefill length of 4096 tokens, a decoding horizon of 512 tokens, and a KV Cache capacity of 4200 entries. The additional QMSum configuration uses 4096, 256, and 4150, respectively, and MultiNews uses 2048, 350, and 2200. Each efficiency run evaluates 100 samples. The configured retention budgets are 10%, 20%, and 50%, with PSC-3 through PSC-6. The runner executes a fixed decoding horizon and truncates the returned text at the first EOS for task scoring.

Timing and Operator Profiling. Model loading occurs before the timed evaluation loop. CUDA synchronization is performed at the beginning and end of timing. The timed interval includes input preparation, prefill, decoding, and other operations inside the loop. Average latency is elapsed time divided by the number of evaluated samples. The separate operator profile uses a 128-token decoding horizon, one warm-up example, one active profiled example, and three independent repeats. CUDA operator self time is distinct from elapsed inference time.

Diagnostic Configurations. The diagnostic experiments use Llama-3.1-8B-Instruct and 20 prompts per configuration. The extended attention and regret assays use a prefill length of 608 tokens, a cache capacity of 1024 entries, and a retention budget of 20%. A full-cache reference supplies a fixed continuation, which is replayed under CTC and PSC at identical absolute positions. Assertions check removal volume, compression events, capacity, and final cache length. The original D2O attention assay uses a prefill length of 832 tokens, a cache capacity of 1024 entries, and a retention budget of 50%.

Software and Reproduction. The current environment for the newly added experiment drivers uses Python 3.10, PyTorch 2.6.0 with CUDA 12.4, Transformers 5.2.0, and Datasets 5.0.1. These versions describe the current environment rather than establishing the versions used by every historical run. The runner selects FlashAttention 2 when installed and otherwise uses the model's default attention implementation. The attention backend must therefore be recorded when reproducing timing results. Configuration files and per-sample outputs support verification.