Proactive Staged KV Cache Compression for Efficient LLMs Inference

Anonymous ACL submission

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 once the memory limit is reached, resulting in cumulative attention perturbation and model performance degradation. In this paper, we propose **PSC** (**P**roactive **S**taged **C**ompression), a model-agnostic framework designed to determine a multi-round compression strategy which can be integrated with any eviction- or merging-based KV Cache compression method. Our key insight is that decomposing compression into multiple rounds with small removal ratios significantly reduces attention perturbation. By analyzing and designing a new framework for multi-round compression during decoding, PSC enables LLMs to retain more important content under limited KV Cache budgets while maintaining model fidelity. Experimental results across various models and datasets show that incorporating PSC significantly enhances the performance of existing representative compression methods, achieving an average accuracy improvement of 25% and accelerating inference by up to 3.57× compared to representative baselines. Our code is available at: https://anonymous.4open.science/r/psc-84A0.

1 Introduction

Large language models (LLMs) (Brown et al., 2020; Zhao et al., 2023; Godbole et al., 2024; 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. Current research primarily focuses on two prevalent strategies, i.e., 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; Liu et al., 2024a; Bolya et al., 2022; Nawrot et al., 2024; Wan et al., 2025). Both aim to identify unimportant tokens and either discard them or merge them into critical ones, thereby reducing the memory footprint of KV Cache. Representative eviction-based methods such as StreamingLLM (Xiao et al., 2024) observe that only the initial tokens and the most recently generated tokens are critical for maintaining generation quality. In contrast, merging-based methods such as CaM (Zhang et al., 2024) and D2O (Wan et al., 2025) formulate token aggregation mechanisms that merge evictable tokens into representative ones, thereby retaining more semantic information. Recent studies further refine merging strategies by recognizing that token importance varies across different layers and even across different attention heads (Fu et al., 2025; Yang et al., 2024; Zhou et al., 2024; Park et al., 2026; 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

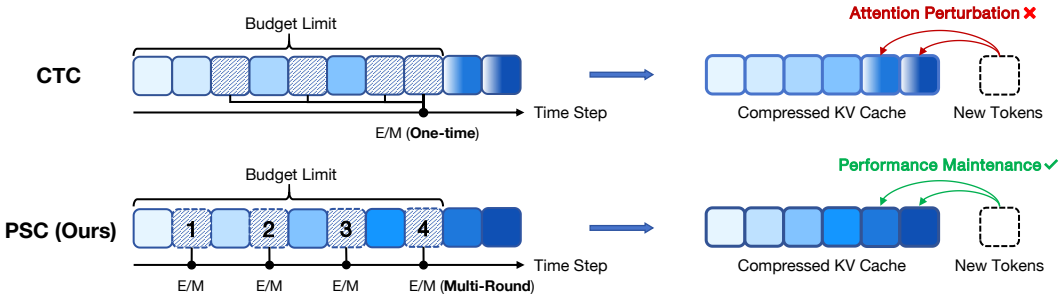

Figure 1: Comparison between CTC and PSC under the same compression ratio of 0.5. Dashed boxes indicate compressed KV Cache entries, where E/M denotes Eviction/Merging. CTC performs a single large-scale compression upon reaching the memory budget, whereas PSC distributes the compression across evenly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

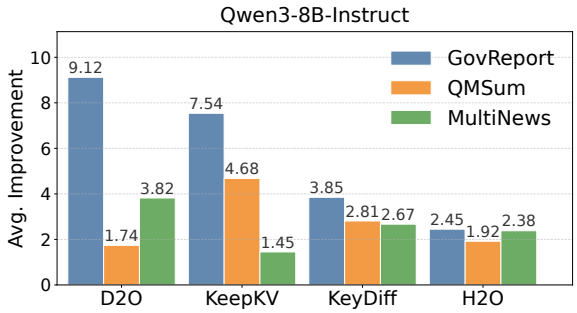

Figure 2: Under KV Cache budgets ranging from 10% to 90%, four representative compression methods integrated with PSC achieve consistent average Rouge-L improvements over CTC across the three summarization datasets in LongBench on Qwen3-8B-Instruct.

in generation quality.

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 theoretical 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 gentler, progressive 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. PSC also seamlessly integrates with existing eviction- or merging-based methods to effectively enhance their performance as shown in Figure 2.

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. To our knowledge, this is the first systematic study on the timing and granularity of compression in long-context inference.
- Theoretical 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 requires no modification to the underlying compressor's scoring mechanism and applies directly to any eviction- or merging-based KV Cache compression method.
- Experimental results demonstrate that PSC consistently improves model accuracy across representative KV Cache compression methods, while

achieving up to **3.57**× throughput improvement and **2.81**× per-sample inference speedup over the CTC baseline.

2 Related Work

Eviction and Merging. Quantization is one of the earliest techniques applied to KV Cache compression, converting high-precision values to low-precision representations to reduce memory footprint (Dong et al., 2024; Liu et al., 2024c; Hooper et al., 2024; Liu et al., 2024b). Despite their simplicity, quantization methods provide limited compression ratios and aggressive precision reduction often degrades model performance (Lin et al., 2024; Li et al., 2025). Eviction- and merging-based approaches instead reduce memory by shortening the effective sequence length. Representative eviction methods (Xiao et al., 2024; Zhang et al., 2023) show that attention concentrates on the earliest and most recent tokens, so retaining only these incurs limited degradation. To mitigate the risk of aggressive eviction, merging strategies (Zhang et al., 2024; Ge et al., 2024) consolidate less important tokens into salient ones via weight assignment. More recent methods combine both strategies: D2O (Wan et al., 2025) integrates similarity measures with heuristic weight allocation, achieving competitive performance under high compression ratios.

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., 2026; 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., 2026) 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-round 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 autoregressive 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:

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

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:

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

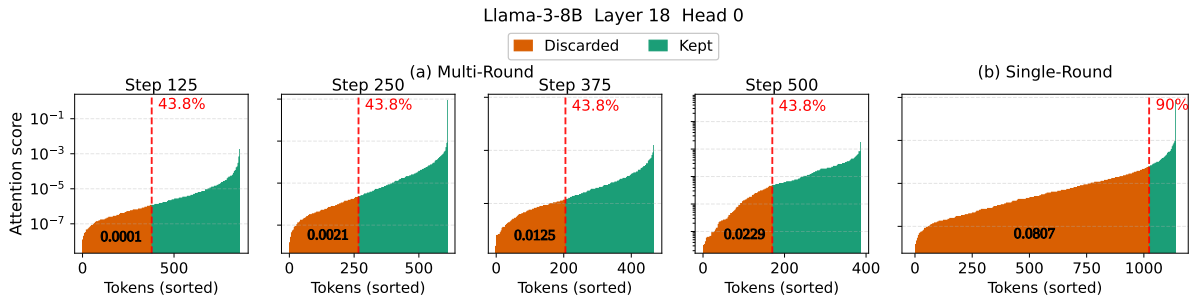

Figure 3: Under the same total compression budget, (a) single-round aggressive pruning incurs a cumulative attention loss of 0.0807, whereas (b) distributing the same compression across multiple rounds reduces this to 0.0376.

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 (Wan et al., 2025; Tian et al., 2026). Because merging only modifies preserved entries and evicted entries are removed entirely, the compression operations at different time steps remain independent.

3.2 Motivation

Capacity-Triggered Compression. 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., 2026). 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.

Single-Snapshot Scoring Leads to Suboptimal Compression. 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. As shown in Figures 3, this sweeps away moderately important tokens alongside truly redundant ones simply to meet the budget constraint.

To isolate the effect of compression timing, we conduct a controlled comparison under identical total compression budgets and generation lengths: CTC defers all compression to the saturation point, while our approach spreads the same total compression across multiple earlier steps. The tokens compressed by CTC carry a total attention score of 0.0807, versus 0.0376 for our incremental approach – a reduction of 53.4% achieved without changing the total number of tokens removed or the underlying scoring criterion. This insight inspires a straightforward redesign: rather than waiting for the cache to saturate and performing one large compression, we split the bulk compression into multiple smaller compressions executed proactively before the cache limit is reached.

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.

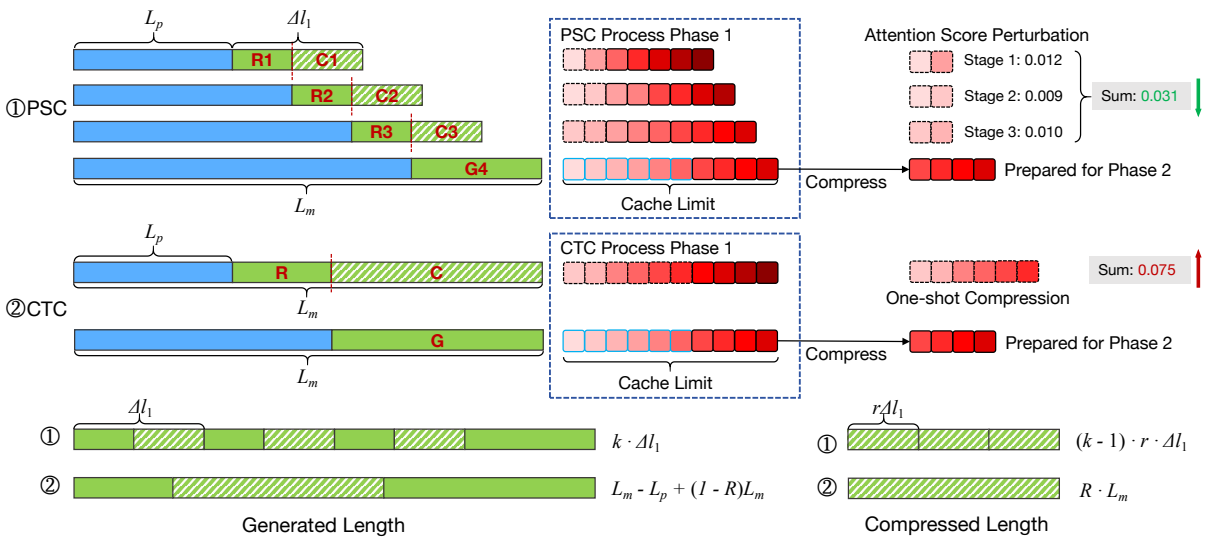

Figure 4: Overview of CTC and PSC Framework. **R**, **C** and **G** denote Retained, Compressed and Generated, respectively. When the decoding length far exceeds the cache budget, both CTC and PSC consist of two phases. In **Phase 1**, by the time PSC's cache first reaches L_m, it has already compressed $(1-R)L_m$ 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: $RL_m \to L_m \to RL_m \to L_m$.

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 (6) into (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 (12) by (11) and solving:

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

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 (Meta AI, 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., 2026) and KeepKV (Tian et al., 2026). By applying PSC on top of each baseline, we demonstrate its consistent effectiveness across diverse compression strategies.

(a) Throughput (tokens/s)

Budget	D2O CTC	D2O **PSC**	KeepKV CTC	KeepKV **PSC**
10%	6.64	**21.09**	17.82	**21.27**
30%	7.57	**32.15**	31.93	**33.53**
50%	9.09	**32.86**	32.28	**35.29**
Avg	7.77	**28.70**	27.34	**30.03**

(b) Per-Sample Latency (s)

Budget	D2O CTC	D2O **PSC**	KeepKV CTC	KeepKV **PSC**
10%	45.05	**16.75**	15.62	**15.41**
30%	32.26	**10.73**	11.32	**10.36**
50%	32.30	**10.76**	11.30	**10.59**
Avg	36.54	**12.75**	12.75	**12.12**

Table 1: Inference efficiency of GovReport for D2O and KeepKV on H20 SXM 96GB, evaluated on Llama-3.1-8B-Instruct. **Bold** values indicate the better result within each CTC–PSC pair.

Implementation Details. All experiments are conducted on NVIDIA A100 80GB GPUs and H20 SXM 96GB 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.

4.2 Performance Evaluation of PSC

4.2.1 Throughput and Inference Latency

Table 1 compares per-method throughput and per-sample latency under CTC and PSC across four budget levels on GovReport, focusing on the two methods that benefit most from multi-round compression. PSC achieves a **3.57× throughput improvement** and a **2.81× per-sample inference speedup** over CTC for D2O. For KeepKV, the

Method	Setting	Summarization			Long-Context Generation		Mathematical Reasoning		Avg
		GovReport	QMSum	MultiNews	WritingPrompts	ROCStories	MATH-500	GSM8K	
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**

Table 2: Performance of KV Cache compression methods on different models at 20% KV Cache budget ($R = 0.2$). Summarization and Long-Context Generation are evaluated with Rouge-L (%); Mathematical Reasoning with Accuracy (%). **Bold** values indicate the better result within each CTC–PSC pair per dataset.

PSC mechanism produces a **1.12× throughput improvement** along with reduced per-sample inference latency. Both methods employ compression kernels whose overhead scales with sequence length. Under CTC, deferred compression forces these kernels to operate on large KV matrices. PSC keeps the average KV Cache length shorter through earlier incremental compression, reducing the computational burden on both attention and the compression kernel. More inference efficiency results are provided in Appendix C.4.

4.2.2 Model Accuracy

Table 2 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.

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

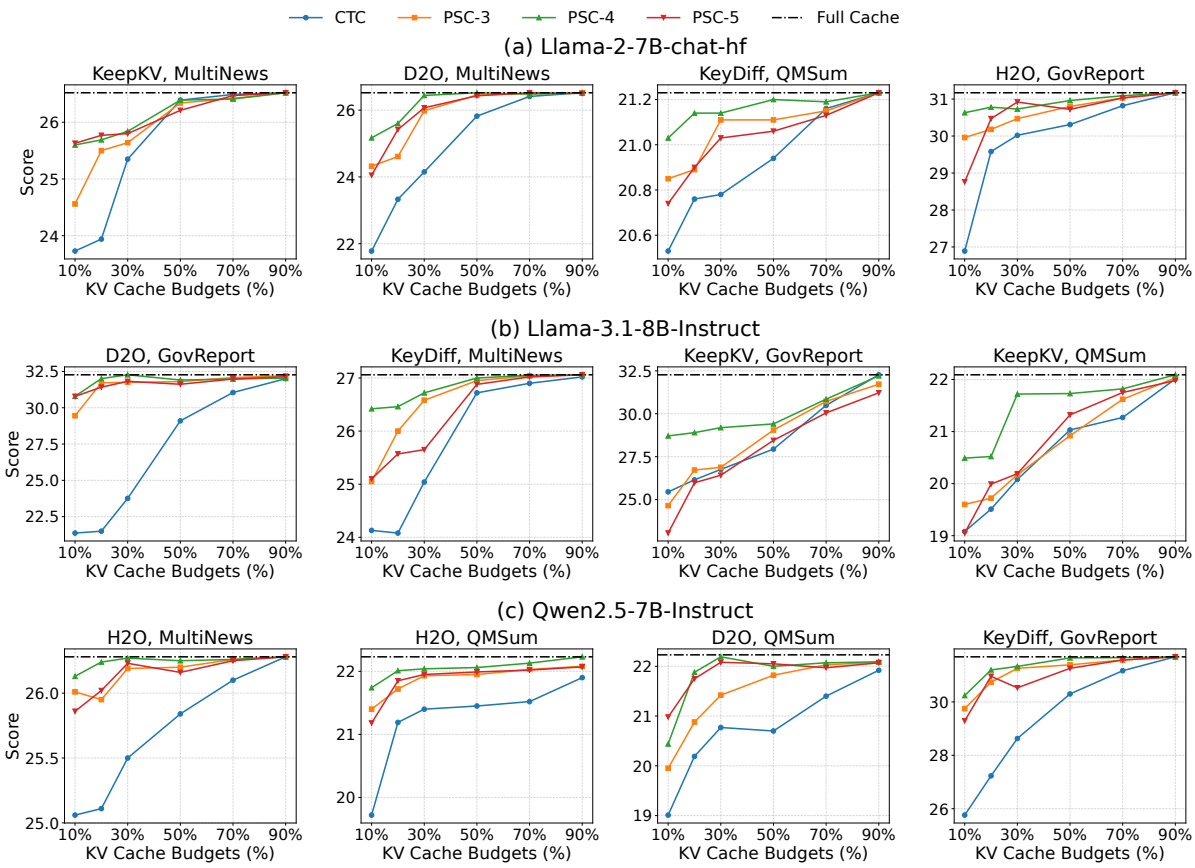

Figure 5: Rouge-L scores on summarization tasks of Longbench under four representative KV Cache compression methods on Llama and Qwen models. Each subplot compares CTC against PSC using different rounds (PSC-3/4/5) across budget levels from 10% to 90%, with the Full Cache score shown as a dashed baseline.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

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.

4.2.3 Performance Evaluation of PSC-k

Figure 5 reports Rouge-L scores on GovReport, QMSum and MultiNews across methods under Qwen2.5-7B-Instruct, Llama-2-7B-chat-hf and Llama-3.1-8B-Instruct, where PSC-k denotes the number of compression rounds applied. Among these, PSC-4 achieves the best overall performance, demonstrating that a moderate number of compression rounds strikes the optimal balance between compression granularity and computational overhead. The experimental results show that PSC-4 successfully recovers competitive performance across different KV Cache budgets, especially at 10% budgets, highlighting PSC's particular strength under high compression ratios, where the multi-round schedule effectively compensates for the information loss inherent in aggressive single-pass compression.

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.

6 Limitations

PSC adopts a uniform schedule of equally spaced compression intervals with fixed compression volumes per round, which is a deliberate simplification that may be suboptimal. More sophisticated policies that adaptively determine both the interval length and compression volume across rounds represent a natural next step, as they could better track the evolving importance distribution of cached tokens during decoding. Multi-round compression inevitably introduces computational redundancy from repeated kernel invocations. Although our experiments demonstrate that this redundancy is negligible in practice, lower-latency compression scheduling strategies remain a worthwhile direction to explore.

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. 2024. 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)*.

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

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

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, and 12 others. 2020. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems*.

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. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. 2025. Expected Attention: KV Cache Compression by Estimating Attention from Future Queries Distribution. *arXiv preprint arXiv:2510.00636*.

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

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. 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*.

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

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

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

Aditi Godbole, Jabin Geevarghese George, and Smita Shandilya. 2024. 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*.

Tanveer Hannan, Md Mohaiminul Islam, Jindong Gu, Thomas Seidl, and Gedas Bertasius. 2025. 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*.

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

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. 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*.

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

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe.

2024. Let's Verify Step by Step. In *International Conference on Learning Representations*.

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

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

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. 2024b. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*.

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

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

Meta AI. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. 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*.

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

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. 2026. KeyDiff: Key Similarity-Based KV Cache Eviction for Long-Context LLM Inference in Resource-Constrained Environments. *Advances in Neural Information Processing Systems*.

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, and 24 others. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

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

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

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

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

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

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

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. 2025. LLMs Know What to Drop: Self-Attention Guided KV Cache Eviction for Efficient Long-Context Inference. *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*.

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

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. 2025a. 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)*.

Wei Wu, Zhuoshi Pan, Kun Fu, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Zheng Wang, and Hui Xiong. 2025b. 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*.

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

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

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

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

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. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *Advances in Neural Information Processing Systems*.

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, and 3 others. 2023. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223*.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. 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*.

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

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 Theoretical Analysis of PSC

Background. Prior work has established that merging keys with high pairwise cosine similarity preserves model performance under KV Cache compression (Park et al., 2026; Tian et al., 2026). Building on this finding, we show that performing compression in multiple rounds retains higher-similarity pairs than a single large compression step under the same budget.

Preliminaries. Let Ω_t denote the set of all pairwise key similarity scores at decoding step t. As new tokens are generated, the candidate set grows monotonically: $\Omega_{t_1} \subset \Omega_{t_2} \subset \cdots \subset \Omega_T$, where T is the final decoding step. Let $N = (1-R)L_m$ be the total number of key pairs to merge, where $R \in (0,1)$ is the retention ratio and L_m is the maximum KV Cache capacity. PSC splits this budget into k rounds of $n = N/k$ merges each, triggered at intermediate steps $t_1 < t_2 < \cdots < t_k$.

Analysis. Traditional compression waits until the final step T and selects the globally optimal top-N pairs from Ω_T:

$$S_{\text{CTC}} = \sum_{p=1}^{N} \hat{s}_p = \sum_{p=1}^{nk} \hat{s}_p, \tag{17}$$

where $\hat{s}_1 \geq \hat{s}_2 \geq \cdots \geq \hat{s}_{|\Omega_T|}$ is the sorted similarity sequence of Ω_T in descending order. PSC instead performs k rounds of greedy local selection, accumulating:

$$S_{\text{PSC}} = \sum_{m=1}^{k}\sum_{j=1}^{n} s_{m,j}, \tag{18}$$

where $s_{m,j}$ is the j-th largest similarity in Ω_{t_m}.

The critical observation is that high-similarity key pairs emerge continuously during decoding: each newly generated segment introduces a fresh batch of highly similar pairs. PSC captures these pairs immediately upon appearance, before their relative ranking is diluted by the accumulation of lower-similarity pairs in later steps. Formally, at each round m, the locally selected top-n pairs satisfy:

$$\sum_{j=1}^{n} s_{m,j} \geq \sum_{j=1}^{n} \hat{s}_{(m-1)n+j}. \tag{19}$$

This holds because $\Omega_{t_m} \subseteq \Omega_T$, yet at step t_m the freshest high-similarity pairs have not yet been competed away by subsequent lower-similarity ar-

Algorithm 1 Proactive Staged Compression (PSC)

Input: L_p, L_m, R, k, importance scorer
Output: Compressed KV Cache throughout decoding

Compute r_1 via Eq. (9), Δl_1 via Eq. (6)
for $i = 1, \ldots, k-1$ **do** ▷ Phase 1: $L_p \to L_m$
  Decode Δl_1 tokens; compress $\lfloor r_1 \Delta l_1 \rfloor$ entries
end for
Decode Δl_1 tokens ▷ cache reaches L_m
$r_2 \leftarrow k \,/\, 2(k-1)$ ▷ independent of R, L_m, L_p
repeat ▷ Phase 2: steady-state
  Compress $(1-R)L_m$ entries; compute Δl_2 via Eq. (10)
  for $i = 1, \ldots, k-1$ **do**
    Decode Δl_2 tokens; compress $\lfloor r_2 \Delta l_2 \rfloor$ entries
  end for
  Decode Δl_2 tokens ▷ cache reaches L_m
until end of sequence

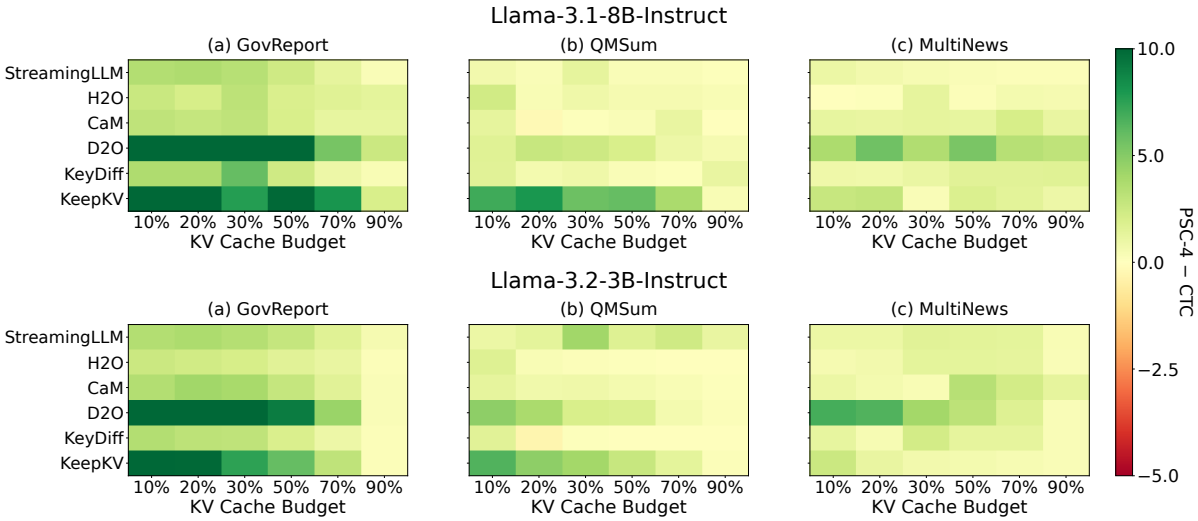

Figure 6: Cell-level PSC-4 improvement over CTC on Llama-3.1-8B-Instruct and Llama-3.2-3B-Instruct, across compression methods, budget levels, and datasets. Green indicates PSC-4 outperforms CTC and red indicates the reverse.

rivals. Summing over all k rounds yields:

$$\begin{aligned} S_{\text{PSC}} &= \sum_{m=1}^{k} \sum_{j=1}^{n} s_{m,j} \\ &> \sum_{m=1}^{k} \sum_{j=1}^{n} \hat{s}_{(m-1)n+j} \\ &= \sum_{p=1}^{nk} \hat{s}_p = S_{\text{CTC}}. \end{aligned} \tag{20}$$

Traditional compression must fill all N slots simultaneously at the final step, so the lower-ranked slots (positions $n+1$ through nk) are inevitably filled with pairs of lower similarity. PSC avoids this by ensuring every slot at each round corresponds to the locally highest-similarity candidates available at that moment. Empirically, Figure 3 shows that the mean attention score of discarded tokens decreases from 0.0807 under single-round compression to 0.0376 under PSC, confirming that PSC more precisely identifies and removes the least informative tokens. The PSC Algorithm is shown in Algorithm 1.

C Additional Experimental Results

C.1 Per-Budget Improvement Analysis

Figure 6 visualizes the cell-level PSC-4 improvement over CTC across methods, budget levels, and

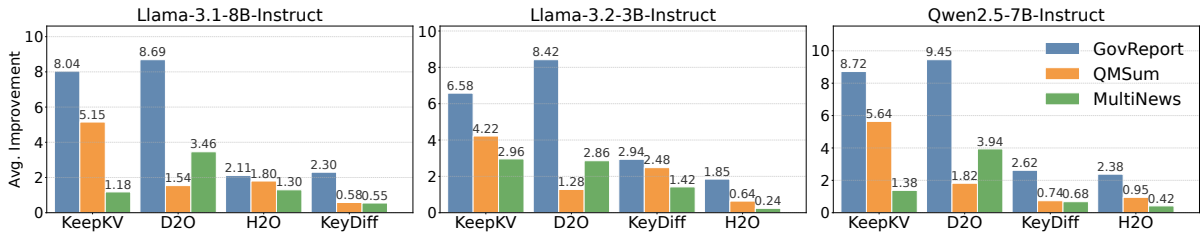

Figure 7: Average PSC improvement over CTC of four representative KV Cache compression methods across different models, measured on three summarization datasets. For each method, the reported value is the mean accuracy gain averaged over KV Cache budgets from 10% to 90%.

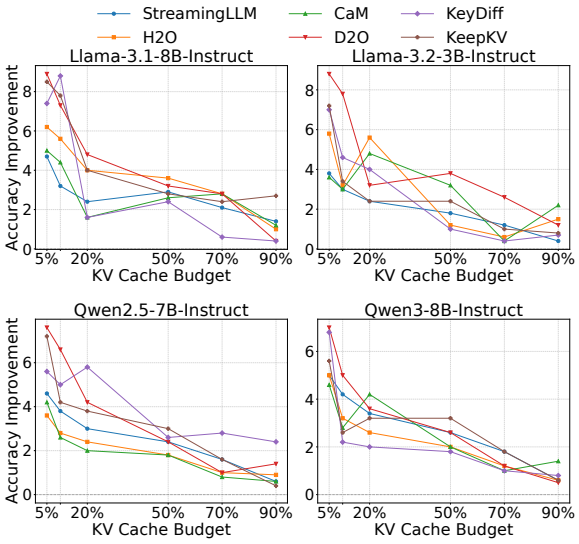

Figure 8: PSC accuracy improvement over CTC across 5% - 90% KV Cache budgets, grouped by compression method and dataset for Llama and Qwen models on MATH-500.

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 the expectation that both methods approach Full Cache performance at high retention ratios. Across datasets, GovReport shows the deepest green cells, reflecting the stronger sensitivity of long-document summarization to compression scheduling.

C.2 Context Summarization

Overall Performance. Figure 7 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 consistent gains across all configurations, with the degree of improvement varying notably across datasets: GovReport benefits the most, while QMSum and MultiNews exhibit comparatively smaller margins.

Effect of Generation Length. This disparity is closely related to 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 finding highlights that the effectiveness of PSC scales with generation length, offering a particularly compelling advantage over reactive strategies such as CTC when outputs are inherently long.

C.3 Mathematical Reasoning

Overall Trend. Figure 8 presents the accuracy improvement of PSC over CTC on MATH-500 for both Llama and Qwen model families, evaluated across six representative KV Cache compression methods under budgets ranging from 5% to 90%. Across all models and methods, PSC consistently outperforms CTC, with the magnitude of improvement exhibiting a clear inverse relationship with the available budget.

Low-Budget Advantage. At extremely low budgets (5%–10%), accuracy gains reach their peak, in some cases exceeding 8 percentage points, while the margin narrows progressively as the budget increases toward 90%. This pattern stems from the fundamental limitation of CTC under resource-constrained conditions. When KV Cache capacity is severely restricted, its reactive compression mechanism frequently discards contextually important tokens before their contribution to generation can be fully realized. PSC alleviates this issue by

(a) Throughput (tokens/s)

	StreamingLLM		H2O		CaM		D2O		KeepKV		KeyDiff	
Budget	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**
10%	22.52	**22.68**	16.50	**16.63**	**16.88**	14.80	6.64	**21.09**	17.82	**21.27**	22.96	**23.21**
20%	22.78	**23.15**	**16.64**	16.62	**17.08**	16.31	6.68	**21.03**	18.86	**22.42**	23.14	**23.59**
30%	34.49	**35.45**	16.76	**17.71**	24.56	**25.05**	7.57	**32.15**	31.93	**33.53**	35.16	**35.90**
50%	**34.19**	33.79	**18.73**	17.49	**23.32**	23.16	9.09	**32.86**	32.28	**35.29**	**34.07**	33.72
Avg	28.50	**28.77**	**17.16**	17.11	**20.46**	19.83	7.50	**26.78**	25.22	**28.13**	28.83	**29.11**

(b) Per-Sample Latency (s)

	StreamingLLM		H2O		CaM		D2O		KeepKV		KeyDiff	
Budget	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**	CTC	**PSC**
10%	**14.54**	14.89	**20.80**	21.00	**19.55**	20.49	45.05	**16.75**	15.62	**15.41**	**14.62**	14.90
20%	**14.50**	14.87	**20.77**	21.01	**19.50**	20.48	45.06	**16.74**	15.59	**15.49**	**14.61**	14.86
30%	**9.53**	9.82	20.28	**19.95**	**13.92**	14.01	32.26	**10.73**	11.32	**10.36**	**9.65**	9.78
50%	**10.16**	10.50	**18.71**	20.20	**14.95**	15.17	32.30	**10.76**	11.30	**10.59**	**10.06**	10.36
Avg	**12.18**	12.52	**20.14**	20.54	**16.98**	17.54	38.67	**13.74**	13.46	**12.96**	**12.24**	12.48

Table 3: Inference efficiency on H20 SXM 96GB (Llama-3.1-8B-Instruct, 200 samples of GovReport). Each method is evaluated under two configurations: CTC applies budget compression in a single pass at the target ratio, while **PSC** columns apply the proposed multi-round compression schedule on top of the same base method. **Bold** values mark the better result within each CTC–PSC pair.

proactively staging compression across multiple decoding steps, preserving a broader candidate set until later phases when relevance can be more accurately assessed.

Generality. The consistency of this trend across both model families and all six compression methods suggests that the advantage of PSC is not specific to any particular architecture or compression criterion, but rather reflects a structural improvement in how budget allocation is managed over the course of generation.

C.4 Inference Efficiency

Experimental Results. PSC inevitably introduces additional overhead per compression event, as each round involves similarity matrix computation, attention score calculation, and KV Cache eviction or merging. Our results show that restricting the number of rounds to a moderate value of k between 3 and 5 keeps this cost negligible. Table 3 reports throughput and per-sample latency for six representative compression methods under CTC and PSC-4 scheduling. PSC remains highly competitive in inference efficiency, and the multi-round compression schedule does not incur unacceptable latency. Moreover, for computationally intensive methods such as D2O and KeepKV, PSC actually improves throughput, further demonstrating its practical efficiency.

Method	**CTC**	**PSC-k**
StreamingLLM	$O(n)$	$O(kn)$
H2O	$O(n \log n)$	$O(kn \log n)$
CaM	$O(nd)$	$O(knd)$
KeyDiff	$O(nd)$	$O(knd)$
KeepKV	$O(n^2 d)$	$O(kn^2 d)$
D2O	$O(n^2 d)$	$O(kn^2 d)$

Table 4: Per-event time complexity of each compression method, where n denotes the current KV Cache size, d denotes the head dimension, and k is the number of PSC rounds. The PSC-k bound is a loose upper bound, as the effective n decreases across rounds due to progressive compression.

Complexity Analysis. Table 4 summarizes the complexity per-event of each method. For lightweight methods such as StreamingLLM and H2O, whose cost is at most $O(n \log n)$, PSC-k introduces only a small constant multiplicative factor over CTC. For computationally heavier methods including D2O and KeepKV, each compression event incurs $O(n^2 d)$ cost, yielding an asymptotic complexity of $O(kn^2 d)$ under PSC-k. In practice, however, PSC compresses tokens progressively across rounds, so the effective cache size n strictly decreases at each stage, making the true overhead substantially lower than the asymptotic estimate. Together with the moderate value of k, this accounts for why the empirical latency penalty observed in Table 3 remains marginal across all methods.