EvolKV: Evolutionary KV Cache Compression for LLM Inference

Bohan Yu[1,2] and **Yekun Chai**[3*]
[1]School of Advanced Interdisciplinary Sciences,
University of Chinese Academy of Sciences, Beijing, China
[2]The Key Laboratory of Cognition and Decision Intelligence for Complex Systems,
Institute of Automation, CAS, Beijing, China
[3]ETH Zurich
yubohan23@mails.ucas.ac.cn yechai@ethz.ch

Abstract

Existing key-value (KV) cache compression methods typically rely on heuristics, such as uniform cache allocation across layers or static eviction policies, however, they ignore the critical interplays among layer-specific feature patterns and task performance, which can lead to degraded generalization. In this paper, we propose EvolKV, an adaptive framework for layer-wise, task-driven KV cache compression that jointly optimizes the memory efficiency and task performance. By reformulating cache allocation as a multi-objective optimization problem, EvolKV leverages evolutionary search to dynamically configure layer budgets while directly maximizing downstream performance. Extensive experiments on 11 tasks demonstrate that our approach outperforms all baseline methods across a wide range of KV cache budgets on long-context tasks and surpasses heuristic baselines by up to 7 percentage points on GSM8K. Notably, EvolKV achieves superior performance over the full KV cache setting on code completion while utilizing only 1.5% of the original budget, suggesting the untapped potential in learned compression strategies for KV cache budget allocation.

1 Introduction

Key-value (KV) cache (Shi et al., 2024; Cai et al., 2024; Xiao et al., 2024; Li et al., 2024) has become a cornerstone of efficient inference in large language models (LLMs) (OpenAI et al., 2024; Touvron et al., 2023; Huang et al., 2024; Bai et al., 2023; Lozhkov et al., 2024), allowing them to reuse previously computed hidden states and thus reduce redundant computation. However, the memory footprint of a KV cache scales linearly with input sequence length, and the quadratic complexity of self-attention makes long-range inference prohibitively slow when the full cache is retained.

To address these challenges, existing KV cache compression methods predominantly rely on rule-based heuristics. Current approaches can be categorized into three paradigms: (1) fixed-position retention across all layers (Child et al., 2019; Beltagy et al., 2020; Xiao et al., 2024), (2) uniform layer allocation with attention-weighted eviction (Li et al., 2024; Ge et al., 2024; Zhang et al., 2023; Liu et al., 2023b; Oren et al., 2024), and (3) pyramidal strategies with predefined depth-wise attenuation (Cai et al., 2024; Yang et al., 2024). While effectively for memory reduction, these heuristics fail to account for two critical aspects: (1) the varying functional roles of transformer layers in information processing (Wang and Tu, 2020; Zhang et al., 2024; Skean et al., 2025), (2) the dynamic relationship between cache and task performance. Relying solely on rule-based allocation of KV cache budgets across layers can lead to suboptimal retention of task-relevant information.

In response to these limitations, we employ evolutionary algorithms (Holland, 1992; Storn and Price, 1997) to directly search for optimal KV cache allocation based on the task performance, inspired by (Chai et al., 2022). We introduce EvolKV, an evolutionary framework that adaptively allocates KV cache budgets across transformer layers, as shown in Figure 1. It formulates per-layer KV cache budgets as optimization variables, partitions them into groups, and employs an evolutionary algorithm (Hansen et al., 2003) to iteratively search for group-wise configurations that directly maximize downstream task fitness scores. By integrating task-driven optimization with layer-specific cache pruning, EvolKV achieves fine-grained, performance-aware allocation aligned with the varying contributions of different layers.

In contrast to rigid heuristics, EvolKV provides a flexible and effective mechanism for layer-wise KV cache budget allocation guided by downstream task objectives. First, it formulates layer/group-wise cache budget as learnable parameters, in which we group layers into optimization units for efficient search. Then, we directly maximize the performance of downstream tasks using black-box evolutionary optimization methods. By doing so, our approach enables task-aware, granular cache allocation that automatically adapts to each group or layer's functional contribution. Specifically, it can accommodate diverse evaluation criteria, such as accuracy and F1 score, and discover non-uniform distributions (*i.e.*, patterns deviated from heuristic fixed-length or pyramidal patterns) without predefined assumptions.

We conduct comprehensive experiments on Mistral-

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 1673–1689
November 4-9, 2025

7B-Instruct and Llama-3-8B-Instruct, evaluating EvolKV across four distinct benchmarks (eleven tasks), covering long-context retrieval, long-context reasoning, and mathematical tasks. Our results demonstrate that task-optimized KV cache allocation yields consistent improvements: (1) On the Needle-in-a-Haystack benchmark, EvolKV achieves up to a 13% improvement over the best baseline. (2) In the RULER benchmark, EvolKV delivers up to a 3.6% gain over the strongest baseline. (3) Across LongBench evaluations, it consistently outperforms all baseline methods across a wide range of target KV cache budgets (ranging from 128 to 2048), and remarkably exceeds the full model's performance while utilizing only 1.5% of its KV cache budget. (4) For GSM8K, EvolKV achieves up to a 7 percentage points improvement in accuracy over the strongest baseline under a 128 KV cache budget, preserving up to 95.7% of the full-model performance, while the strongest baseline retains only up to 84.5% under a 512 KV cache budget.

In conclusion, our key contributions are as follows:

- We propose EvolKV, the first framework to formulate layer-wise KV cache budget allocation as a black-box optimization problem.
- EvolKV operates on frozen LLMs, supports arbitrary evaluation metrics, without requiring fine-tuning or architectural modifications.
- Empirical results show that task-aware KV cache allocations consistently diverge from conventional heuristics, favoring non-uniform distributions that depart from fixed or pyramidal rules. EvolKV consistently outperforms strong baselines across long-context and reasoning tasks, even surpassing full-model performance under extreme compression.

2 Related Work

KV Cache Compression The substantial parameter counts of LLMs present significant challenges for inference. To overcome this difficulty, many efforts have been made to improve the inference efficiency of LLMs, such as the methods described in (Zhang et al., 2023; Sheng et al., 2023; Liu et al., 2023b; Li et al., 2024; Oren et al., 2024), which evict the KV caches with the lowest accumulated attention weights. StreamingLLM (Xiao et al., 2024) identifies the phenomenon of attention sink by retaining only the initial and most recent KV caches, thus maintaining a fixed positional alignment in the KV cache. These methods maintain a uniform KV cache budget across all layers, disregarding the actual demand for KV cache in each individual layer. Meanwhile, previous studies such as (Cai et al., 2024; Yang et al., 2024) have reported a progressive reduction in the number of important KV cache across layers, forming a pyramid-shaped distribution. However, existing methods often set an attenuation coefficient to control the KV cache budget in each layer, thus ignoring that the cache budget actual needed in each layer do not necessarily exhibit a monotonically decreasing pattern. Instead of rule-based or heuristic methods, this study aims to compress the KV cache through an evolutionary, layer-wise optimization approach.

Evolutionary Algorithms These are stochastic optimization algorithms inspired by the principles of natural evolution (Holland, 1992; Vent, 1975; Alam et al., 2020). It is widely used to address complex optimization and search problems by iteratively evolving high-quality solutions through the simulation of biological mechanisms such as selection, crossover (recombination), and mutation (Koza, 1992; Kennedy and Eberhart, 1995; Storn and Price, 1997; Hansen et al., 2003). In this study, we leverage evolutionary algorithms in conjunction with downstream task performance to optimize the configuration of layer-wise KV cache budgets.

3 Evolutionary KV Cache Compression

3.1 Motivation

We observe that the existing methods mainly present the following categories in the KV cache budget allocation of layers:

- **Fixed Position:** Each layer preserves KV caches at the same position (Child et al., 2019; Beltagy et al., 2020; Xiao et al., 2024).
- **Identical Budget:** Each layer retains the same budget of KV caches, but their positions vary across layers (Zhang et al., 2023; Liu et al., 2023b; Li et al., 2024).
- **Pyramidal Allocation:** The KV cache budget allocation follows a pyramidal pattern, decreasing progressively across layers (Cai et al., 2024; Yang et al., 2024).

Previous studies (Wang and Tu, 2020; Zhang et al., 2024; Skean et al., 2025) have shown that different layers of LLMs vary in their importance and process information at different levels of granularity. However, most existing compression methods overlook such heterogeneity and instead adopt rule-based or heuristic strategies for KV cache compression, which often results in suboptimal inference performance. These observations highlight the necessity of adapting the KV cache budget individually for each layer.

3.2 EvolKV

To address the limitations of rule-based or heuristic allocation strategies, as shown in Figure 1, we introduce EvolKV, a dynamic and task-driven evolutionary framework that adaptively allocates the KV cache budget for each layer by leveraging performance feedback from downstream tasks. We present a comparison of budget allocations between EvolKV and other methods in Figure 2a.

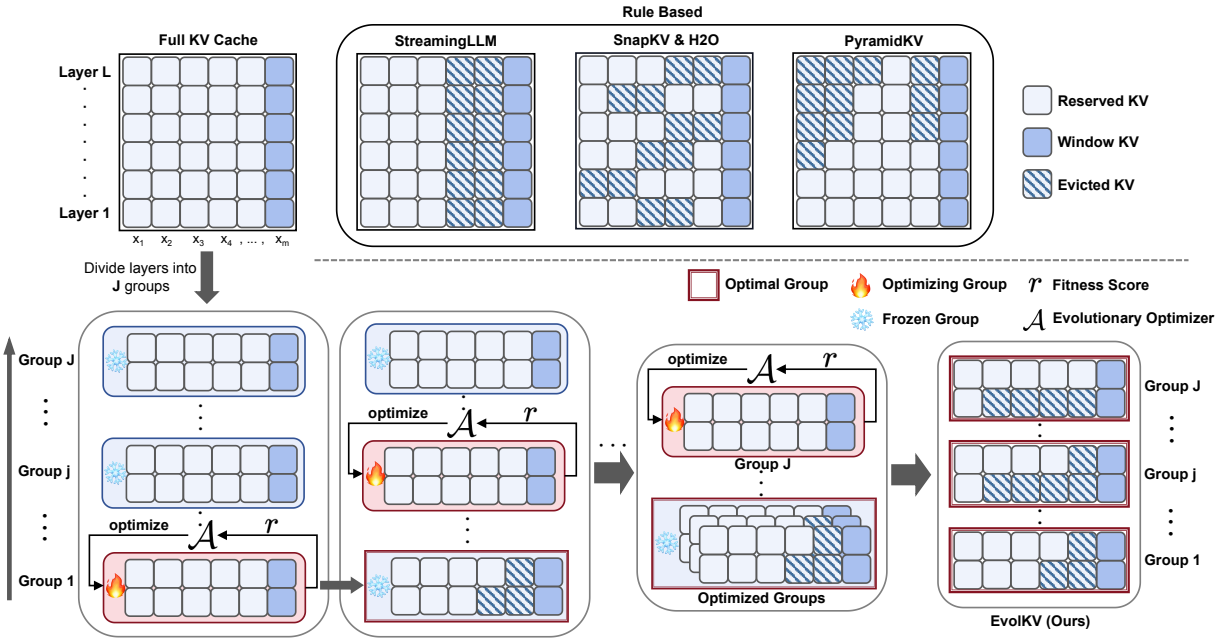


Figure 1: Illustration of the EvolKV framework. Compared to rule-based strategies (top row), EvolKV performs bottom-up, group-wise KV cache budget optimization using evolutionary search, progressively refining each layer group based on task-specific fitness feedback.

Optimization Objectives of Evolutionary Compression Evolutionary algorithms generate candidate solutions and evaluate their fitness, iteratively refining the search strategy based on fitness feedback to progressively guide the population toward better solutions. In this paper, EvolKV treats performance feedback from downstream tasks as fitness and leverages evolutionary algorithms to guide the per-layer KV cache compression. Specifically, in a language model with L transformer layers, we denote the KV cache budget of layer i as $k_i \in \mathbb{N}, \forall i \in \{1, \ldots, L\}$. Given a set of candidate compression schemes $\mathbb{S}$ produced by an evolutionary algorithm for a downstream task $f(\cdot)$, we aim to identify the optimal scheme S^* that maximizes task performance while minimizing deviation from the target average KV cache budget c:

$$S^* = \arg\max_{S \in \mathbb{S}} f(S) \left(1 + \lambda \, \textsc{CacheScore}(S, c)\right)$$
$$s.t. \quad \frac{1}{L} \sum_{i=1}^{L} k_i \leq c \tag{1}$$

where $f(S)$ is the downstream-task performance obtained with compression scheme $S \in \mathbb{S}$, and the hyperparameter $\lambda > 0$ balances raw performance against cache efficiency. Given the wide variety and differing value ranges of downstream performance metrics (e.g., accuracy, F1, ROUGE), we adopt a cache-efficiency term that is directly weighted against task performance to ensure comparability. The cache-efficiency term $\textsc{CacheScore}(S, c) \in [0, 1]$ assigns a lower value to schemes whose average per-layer cache budget $\bar{k} = \frac{1}{L} \sum_{i=1}^{L} k_i^{(S)}$ exceeds the target budget c, while applying a smooth discount to those that stay within the target:

$$\textsc{CacheScore}(S, c) = \begin{cases} \max\left(0, 1 - \frac{\bar{k} - c}{c}\right), & \text{if } \bar{k} > c \\ 1 - \gamma\left(1 - \frac{\bar{k}}{c}\right), & \text{if } \bar{k} \leq c \end{cases} \tag{2}$$

where $\gamma \in (0, 1]$ is a smoothing factor. Thus, the objective favors compression schemes that (i) deliver strong task performance and (ii) keep their average KV cache budgets close to, or below, the desired budget.

Grouping of KV Cache Budgets To improve optimization efficiency, we introduce a group size parameter n_g to partition the KV cache budgets $\mathbf{K} = \{k_1, k_2, \ldots, k_L\}$ into $J = \lceil L/n_g \rceil$ groups, denoted as $G = \{g_1, g_2, \ldots, g_J\}$. Each group g_j contains a contiguous subset of cache budgets defined as $g_j = \{k_{(j-1)\cdot n_g+1}, \; k_{(j-1)\cdot n_g+2}, \; \ldots, \; k_{\min(j\cdot n_g, \, L)}\}, \forall j \in \{1, 2, \ldots, J\}$. For simplicity, we assume that the total number of layers L is divisible by the group size n_g, such that $L = J \cdot n_g$. Under this formulation, candidate compression schemes $\mathbb{S}$ are applied at the group level and denoted by $\mathbb{S}_g$. The optimal scheme selected for each group, based on downstream task performance, is denoted by S_g^*. This group-wise formulation significantly reduces the search space and facilitates more stable optimization dynamics during the evolutionary search process.

Iterations of Evolutionary Compression Our KV cache budget optimization is conducted in a group-wise manner, as shown in Algorithm 1, proceeding sequentially from the bottom to the top layers. During the

Algorithm 1 EvolKV — Evolutionary and Group-wise KV Cache Budget Optimization

Require: Target average KV cache budget c; cache budget efficiency weight λ; smoothing factor γ; group size n_g; the number of model layers L; max iterations M; CACHESCORE function; downstream task scorer $f(\cdot)$; evolutionary optimizer $\mathcal{A}$
Ensure: Globally optimal group KV cache budgets G^*

Initialize KV cache budgets $\mathbf{K} \leftarrow (c, \ldots, c) \in \mathbb{N}^L$
Partition $\mathbf{K}$ into $J = \lceil L/n_g \rceil$ groups $G = \{g_1, \ldots, g_J\}$
$G^* \leftarrow G$ ▷ initialize group KV cache budgets
$F_{\text{best}} \leftarrow -\infty$ ▷ initialize global best fitness
for $j \leftarrow 1$ **to** J **do** ▷ optimize one group at a time
  $\mathcal{A}$.INITIALIZE(g_j) ▷ initialize parameters of $\mathcal{A}$
  for $m \leftarrow 1$ **to** M **do**
    Obtain candidate group compression schemes $\mathbb{S}_g$ from $\mathcal{A}$
    Evaluate the fitness r of each $S_g \in \mathbb{S}_g$:
      $\tilde{G} = G^*$ with $g_j := S_g$
      $r \leftarrow f(\tilde{G})(1 + \lambda \text{CACHESCORE}(S_g, c))$
        ▷ evaluate with g_j of G^* replaced by S_g, others fixed
    Update F_{best} with r, G^* with $\tilde{G}$ if $r > F_{\text{best}}$
    Update the evolutionary optimizer $\mathcal{A}$ using r and S_g
  end for
end for
return G^*

optimization of each group, the KV cache budgets of previously optimized groups are fixed to their respective optimal schemes S_g^*, while the remaining groups retain their initial values. If a candidate scheme S_g achieves a higher fitness score r than the current best, the KV cache budgets of the current group are updated accordingly. This process is repeated iteratively until all groups have been optimized.

KV Cache Budget Completion To ensure fairness in evaluation, we complete any KV cache budget optimization result whose total size deviates from the target. Specifically, we first compute the discrepancy between the achieved total KV cache budget $A = \sum_{i=1}^{L} k_i$ and the target total budget $T = c \cdot L$, denoted as $\Delta_{\text{cache}} = T - A$. This discrepancy is then proportionally redistributed across layers based on their original share of A. The completed KV cache budgets $B = \{b_1, b_2, \ldots, b_L\}$, where $b_i = \left\lceil k_i + \frac{k_i}{A} \cdot \Delta_{\text{cache}} \right\rceil, i \in \{1, 2, \ldots, L\}$.

4 Experiments

4.1 Experiment Settings

Models We employ two open-source models, Mistral-7B-Instruct[1] (Jiang et al., 2023) with 32K context length and Llama-3-8B-Instruct (Grattafiori et al., 2024) with 8K context length.

Datasets Our proposed EvolKV is evaluated on four benchmarks: LongBench (Bai et al., 2024), GSM8K (Cobbe et al., 2021), Needle-in-a-Haystack (NIAH)[2], and RULER (Hsieh et al., 2024). In RULER, we evaluate EvolKV on eleven sub-datasets across three major tasks: *Retrieval* (Single NIAH with three sub-datasets, Multi-keys NIAH with three sub-datasets, Multi-queries NIAH, Multi-values NIAH), *Aggregation* (CWE, FWE), and *Multi-hop Tracing*. For LongBench, we select sixteen representative sub-datasets spanning six major task categories: *single-document QA* (NarrativeQA (Kočiský et al., 2017), Qasper (Dasigi et al., 2021), MultiFieldQA-en), *multi-document QA* (HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022)), *summarization* (GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019)), *few-shot learning* (TREC (Li and Roth, 2002), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019)), *synthetic reasoning* (PassageCount, PassageRetrieval-en), and *code completion* (LCC (Guo et al., 2023), RepoBench-P (Liu et al., 2023a)). For detailed introduction of datasets, see Table 6 in Appendix A.

Baselines and Settings For a fair comparison with the current strong baselines, we keep the KV cache budget and all other hyperparameters identical in all evaluations.

(1) StreamingLLM (Xiao et al., 2024). This method retains the original KV cache together with those from the most recent window and initial cache, preserving the KV cache at fixed positions in every layer.

(2) SnapKV (Li et al., 2024). This method selects relevant KV cache from the preceding sequence based on the most recent query states (window query states), with a uniform cache budget applied across all layers. We employ SnapKV as the base method and apply the KV cache budgets optimized by EvolKV for downstream task inference and baseline comparison. Notably, when the optimized budgets are uniform across all layers, the resulting configuration reduces to standard SnapKV.

(3) PyramidKV (Cai et al., 2024). A rule-based and pyramid-shaped strategy that progressively reduces the KV cache budget from the lower to higher layers.

Experimental Setup We apply Covariance Matrix Adaptation Evolution Strategy (CMA-ES (Hansen et al., 2003)) as our evolutionary optimizer. We set the window size to 32, kernel size to 7 and apply max pooling. For EvolKV, we fix the hyperparameters to λ = 0.3, γ = 0.2, learning rate of CMA-ES σ = 0.3 and group size n_g = 8. The population size of CMA-ES is calculated according to the following empirical formula: $4 + \lfloor 3 \cdot \ln(n_g) \rfloor$ (Belkhir et al., 2015).

4.2 Results

4.2.1 Experiments on LongBench

Settings We evaluate EvolKV on LongBench, a comprehensive benchmark consisting of six major task categories designed to assess a model's capacity for long-context understanding and reasoning. We first employ Mistral-7B-Instruct as the backbone model and use F1 score as the optimization objective. KV cache budgets are optimized using only 30 randomly sampled instances from NarrativeQA, under a target average cache budget of $c = 128$. The resulting allocation, illustrated in Fig-

[1] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

[2] https://github.com/gkamradt/LLMTest_NeedleInAHaystack

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
Full	26.95	32.99	49.78	44.23	27.51	18.49	33.09	24.51	27.12	71.00	86.23	43.09	2.91	86.31	57.27	53.88	42.84
KV Size = 128																	
SnapKV	21.89	21.28	42.76	37.76	21.71	14.71	19.32	21.71	**21.28**	49.00	83.62	39.99	2.66	68.05	51.78	**48.36**	35.37
PyramidKV	20.40	21.39	43.95	38.96	23.80	15.46	19.69	**22.36**	21.23	51.00	84.77	**40.07**	2.56	72.20	**52.36**	47.62	36.11
StreamingLLM	17.28	13.21	27.11	30.82	21.94	11.87	15.48	19.37	17.98	44.00	80.22	37.32	**3.75**	23.77	51.43	45.50	28.82
EvolKV	**22.76**	**22.59**	**44.02**	**39.47**	**24.16**	**15.64**	**19.90**	21.93	21.20	**52.00**	**86.83**	39.83	2.35	**74.81**	51.64	47.05	**36.64**
KV Size = 256																	
SnapKV	22.81	24.18	47.95	38.39	23.22	15.31	21.91	**23.13**	23.30	61.50	**85.90**	**41.37**	3.01	**84.81**	**55.23**	51.16	38.95
PyramidKV	21.68	**24.90**	47.59	39.13	23.03	16.74	21.62	23.01	22.83	62.50	84.65	40.71	3.13	82.65	54.29	50.68	38.70
StreamingLLM	19.43	15.31	27.97	31.57	21.78	11.30	18.04	19.18	19.94	51.00	80.75	39.60	**3.65**	16.90	53.96	47.28	29.85
EvolKV	**22.87**	24.50	**48.28**	**40.26**	**25.74**	**17.39**	**22.33**	22.39	**23.61**	**65.50**	84.57	40.66	2.85	79.35	54.10	**51.20**	**39.10**
KV Size = 512																	
SnapKV	24.68	27.97	48.80	40.32	24.89	16.99	23.73	**23.57**	24.63	67.00	86.12	41.43	2.47	88.06	56.37	52.80	40.61
PyramidKV	24.39	27.49	48.78	40.92	24.58	16.16	23.44	23.48	24.05	67.00	85.87	41.42	2.86	86.23	55.62	51.88	40.26
StreamingLLM	21.39	16.36	30.75	30.89	22.20	10.95	21.53	20.02	23.10	61.50	81.86	**41.72**	**3.14**	18.57	55.16	48.65	31.74
EvolKV	**24.89**	**29.00**	**49.78**	**41.57**	**26.27**	**18.34**	**24.41**	23.18	**25.00**	**68.00**	**87.07**	41.64	2.87	**89.74**	**56.58**	**52.85**	**41.32**
KV Size = 1024																	
SnapKV	25.46	29.15	**49.03**	41.58	25.30	**18.96**	26.19	**23.99**	**25.99**	69.50	**86.63**	**43.01**	2.84	89.21	**57.41**	53.25	41.72
PyramidKV	25.45	29.97	48.72	41.02	**25.85**	18.53	25.27	23.66	25.52	69.00	86.31	42.20	2.66	86.67	56.39	**53.38**	41.29
StreamingLLM	22.74	18.51	31.03	33.03	22.57	11.85	24.09	20.75	25.54	64.00	84.71	41.26	**3.49**	22.40	55.89	50.99	33.30
EvolKV	**25.63**	**30.30**	48.96	**42.84**	25.78	18.21	**26.99**	23.79	25.95	**70.00**	86.09	41.74	3.06	**89.82**	57.01	53.26	**41.84**
KV Size = 2048																	
SnapKV	**26.29**	**32.65**	49.09	41.70	27.39	18.49	28.77	**24.42**	26.55	70.00	86.27	**42.47**	2.79	87.56	**57.42**	53.41	42.20
PyramidKV	25.61	31.33	48.89	41.90	26.64	17.65	28.09	23.86	26.52	**71.50**	**86.30**	42.27	2.52	86.85	57.41	**53.66**	41.94
StreamingLLM	22.28	23.08	35.22	33.66	22.90	13.47	26.85	20.95	26.45	66.00	85.68	41.95	2.40	25.75	57.13	52.17	34.75
EvolKV	26.13	32.31	**49.18**	**42.69**	**27.63**	**18.64**	**29.11**	24.10	**26.72**	71.00	86.25	42.07	**2.85**	**87.81**	57.12	53.63	**42.33**

Table 1: Comparison of KV cache compression methods on Mistral-7B-Instruct across LongBench tasks. EvolKV outperforms all baseline methods on average across KV cache budgets ranging from 128 to 2048, and even surpasses the full model on certain tasks such as TriviaQA.

ure 3a, is then extrapolated to target cache budgets of 256, 512, 1024, and 2048 using the method described in Section 3.2, without further optimization. Following the same protocol, we conduct experiments on Llama-3-8B-Instruct, selecting six representative sub-datasets from LongBench—NarrativeQA, HotpotQA, QMSum, TREC, PassageRetrieval-en, and LCC—each with five randomly sampled instances. We perform EvolKV optimization with $c = 128$, and report the corresponding downstream task metrics applied in optimization in Table 6 (Appendix A). The optimized KV cache budgets are subsequently extended to target average values of 256, 512, and 1024. For $c = 2048$, we conduct a separate optimization to obtain the dedicated cache budget allocation.

Results Table 1 reports the evaluation results on 16 LongBench sub-datasets using Mistral-7B-Instruct, with all training samples removed. Across all evaluated KV cache budgets, EvolKV consistently achieves the highest average performance, outperforming all rule-based baselines. Furthermore, on several sub-datasets, including MultiFieldQA-en, 2WikiMultihopQA, MuSiQue, TriviaQA, and PassageRetrieval-en, EvolKV not only remains competitive with the uncompressed full model but even surpasses it at certain KV cache budgets. Table 2 presents analogous results on Llama-3-8B-Instruct, again with training samples excluded. EvolKV demonstrates superior performance across all KV cache budgets. Remarkably, at a cache budget of 128, EvolKV outperforms the strongest baseline by 7.69 percentage points on the TREC subset, highlighting its strong adaptability to diverse downstream tasks.

Analysis We conduct a detailed analysis of EvolKV's performance across the six major task categories in the LongBench, with the results presented in Figure 5 and 6 (Appendix B.1). EvolKV already surpasses all baselines under low-budget settings (e.g., $c = 128$ and 256) across multiple tasks: on Mistral-7B-Instruct it outperforms every baseline in single- and multi-document QA, and few-shot learning, while on Llama-3-8B-Instruct it achieves the top scores in few-shot learning and code completion. When the budget is relaxed to the range from 512 to 2048, EvolKV's advantage becomes even more pronounced: on Mistral it exceeds the best baseline by up to 1.5 points in multi-document QA and surpasses the full model by 1.8 points in the synthetic task at $c = 1024$, whereas on Llama it surpasses all baselines in few-shot learning, synthetic, and code tasks, achieving 1–3 point gains over the full model in code task. Notably, this advantage persists even under extreme compression, as EvolKV still outperforms the full model at $c = 128$ (1.5% of the context length), where all other baselines fall short.

These results collectively demonstrate EvolKV's strong and stable performance across both constrained and relaxed budget conditions, particularly in long-context scenarios. In contrast, PyramidKV's pyramid-style allocation incurs significant losses on the code task, and StreamingLLM trails behind on nearly all tasks, reinforcing the limitation of static, rule-based allocation schemes. Notably, The KV cache budgets optimized by EvolKV at $c = 128$ generalize smoothly to larger average cache budgets ranging from 256 to 2048, suggesting that the method captures stable, task-aligned importance patterns rather than overfitting to a specific allocation regime. Even when the 30 optimization instances are excluded from evaluation, EvolKV continues to outperform all baselines, suggesting that its performance improvements stem from more effective KV cache budget allocation rather than memorization of training examples. These findings collectively highlight the superiority of EvolKV's adaptive and task-aware strategy over traditional heuristic approaches in real-world inference scenarios.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
Full	25.51	31.49	39.80	43.62	35.96	21.39	28.74	23.19	26.79	73.85	90.50	42.89	4.18	68.21	58.89	53.59	41.79
KV Size = 128																	
SnapKV	21.96	**13.51**	30.85	35.68	29.14	19.21	**19.37**	21.63	20.14	46.15	88.32	38.28	4.30	**68.21**	57.33	54.85	35.56
PyramidKV	**22.25**	13.20	31.54	39.01	27.57	**20.18**	19.19	**21.91**	**20.71**	50.26	87.34	**38.53**	4.55	**68.21**	**57.55**	54.25	36.02
StreamingLLM	18.91	7.79	20.90	33.79	24.85	14.94	16.37	20.41	18.49	46.15	74.40	35.78	4.75	67.18	55.59	52.32	32.04
EvolKV	21.18	13.12	**33.64**	**40.70**	**32.42**	19.81	17.29	21.26	19.03	**57.95**	**89.30**	37.38	**5.00**	67.86	57.41	**55.11**	**36.78**
KV Size = 256																	
SnapKV	**24.49**	**18.37**	33.36	**42.95**	**34.05**	20.36	**20.62**	**22.03**	**22.51**	59.49	89.61	**39.22**	5.00	**68.21**	59.35	56.27	38.49
PyramidKV	24.14	15.75	**34.15**	40.36	29.78	**21.22**	20.11	21.95	22.30	64.62	90.03	39.16	**5.08**	**68.21**	59.21	53.79	38.12
StreamingLLM	8.88	10.80	20.34	34.21	25.62	15.54	19.29	20.30	20.75	53.33	78.75	39.20	4.83	66.36	58.29	54.50	33.81
EvolKV	22.68	17.48	34.09	42.40	32.92	19.91	19.34	21.99	21.62	**67.18**	**90.27**	39.08	5.06	**68.21**	**59.71**	**56.70**	**38.67**
KV Size = 512																	
SnapKV	**25.26**	22.45	36.84	42.84	**36.13**	20.53	**22.38**	22.44	**24.11**	69.74	**90.39**	**40.40**	4.98	**68.21**	**60.20**	55.58	40.16
PyramidKV	24.97	21.58	36.45	**43.36**	32.97	19.55	22.24	**22.64**	23.75	68.72	90.31	40.30	5.00	67.95	58.72	54.61	39.57
StreamingLLM	21.20	11.02	21.82	35.97	26.61	15.23	20.90	20.66	23.66	62.56	84.04	40.27	4.75	66.66	59.74	55.14	35.64
EvolKV	**25.26**	**22.91**	**37.17**	42.26	35.99	**20.55**	21.72	22.45	23.32	**71.28**	90.36	39.45	**5.26**	**68.21**	60.04	**58.12**	**40.27**
KV Size = 1024																	
SnapKV	**25.16**	25.92	38.39	43.48	34.98	**20.25**	**24.05**	22.09	25.21	72.82	90.43	**40.96**	5.08	**68.21**	59.79	56.50	40.83
PyramidKV	24.90	24.82	**38.75**	43.33	34.86	20.16	24.03	22.86	25.36	72.82	90.34	40.69	5.12	**68.21**	59.05	53.87	40.57
StreamingLLM	21.43	15.37	26.18	36.26	28.41	15.56	23.26	21.24	**25.46**	66.67	86.01	40.59	3.96	68.03	**60.11**	56.11	37.17
EvolKV	24.91	**27.31**	37.41	**43.62**	**36.03**	20.09	23.44	**23.09**	24.70	**74.36**	**90.51**	40.55	**5.28**	**68.21**	59.90	**57.43**	**41.05**
KV Size = 2048																	
SnapKV	**25.91**	**29.45**	38.98	43.64	35.55	**21.52**	**25.83**	23.12	26.25	**73.85**	**90.56**	41.82	**4.99**	**68.21**	59.55	55.03	41.52
PyramidKV	25.62	29.21	37.43	43.55	**36.77**	21.48	25.75	22.85	26.32	**73.85**	90.34	**41.85**	4.45	**68.21**	59.47	54.15	41.33
StreamingLLM	23.54	23.26	30.13	39.28	32.61	17.15	25.05	21.59	26.26	70.77	89.78	41.50	4.62	67.69	**60.01**	**57.17**	39.40
EvolKV	25.52	29.44	**39.60**	**44.19**	36.58	20.91	25.78	**23.34**	**26.40**	**73.85**	90.48	41.50	4.91	**68.21**	59.45	54.83	**41.56**

Table 2: Comparison of KV cache compression methods on Llama-3-8B-Instruct across LongBench tasks. EvolKV outperforms all baselines on average across KV cache budgets from 128 to 2048, and even surpasses the full model on tasks like RepoBench-P under the 128 budget.

4.2.2 Experiments on GSM8K

Settings We quantify the logical reasoning ability of EvolKV under different KV cache budgets on GSM8K. To eliminate prompt-format bias, all models receive identical few-shot chain-of-thought examples. For detailed prompt formatting and evaluation setup, we refer readers to the Qwen implementation[3]. For EvolKV optimization, we randomly sample 30 instances from the GSM8K training set and perform KV cache budget optimization at $c = 128$, using accuracy as the objective metric. The resulting allocation is subsequently up-scaled to $c = 256$ and $c = 512$, without conducting additional searches. For Mistral-7B-Instruct, a separate optimization is performed under the setting $c = 256$.

Results Figure 3b presents the KV cache budget allocations optimized by EvolKV for Llama-3-8B-Instruct, and Table 3 reports the corresponding test set accuracies for both Llama-3-8B-Instruct and Mistral-7B-Instruct. Across all configurations, EvolKV consistently outperforms baseline methods on both models. Specifically, on Llama-3-8B-Instruct, it achieves substantial improvements over the strongest competitor, with accuracy gains of at least 7.28, 2.05, and 7.58 at KV cache budgets of 128, 256, and 512, respectively. Notably, EvolKV achieves 95.7% of the full-model performance using a reduced cache budget ($c = 512$), significantly outperforming all baselines, whose best result reaches only 84.5%.

Analysis KV cache budgets optimized only at $c = 128$ transfer effectively to larger budgets, indicating that the evolutionary search captures stable layer-importance signals rather than overfitting to a single setting. Notably, StreamingLLM performs poorly on this task, suggesting that fixed-position KV cache strategies are suboptimal for reasoning-oriented tasks. These results validate our central claim and extend our findings in LongBench: evolutionary, task-aware and layer-wise KV cache allocation unveils latent layer-specific cache requirements that fixed heuristics miss, yielding superior reasoning accuracy while retaining substantial memory savings.

[3] https://github.com/QwenLM/Qwen/blob/main/eval/evaluate_chat_gsm8k.py

Method	**Llama-3-8B-Instruct** Accuracy	**Mistral-7B-Instruct** Accuracy
Full	67.85	50.80
KV Size = 128		
SnapKV	23.58	40.26
PyramidKV	40.71	41.32
StreamingLLM	3.64	2.58
EvolKV	**47.99**	**41.47**
KV Size = 256		
SnapKV	44.50	44.05
PyramidKV	49.05	43.59
StreamingLLM	3.41	3.03
EvolKV	**51.10**	**44.81**
KV Size = 512		
SnapKV	56.94	46.02
PyramidKV	57.32	46.02
StreamingLLM	4.02	3.87
EvolKV	**64.90**	**46.70**

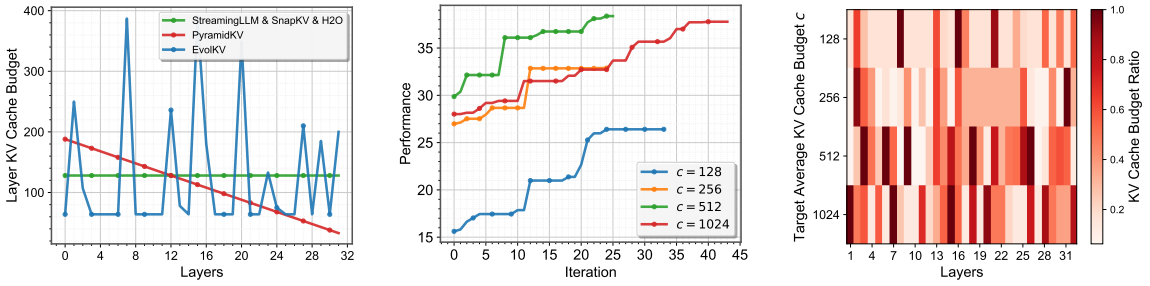
Table 3: GSM8K results of Llama-3-8B-Instruct and Mistral-7B-Instruct under different KV cache budgets.

4.2.3 Experiments on NIAH and RULER

We evaluate the long-context retrieval capability of EvolKV alongside all baselines on NIAH. For optimization, EvolKV applies no more than 35 instances whose average scores on both Llama-3-8B-Instruct and Mistral-7B-Instruct are below 60, while the target KV

Method	Retrieval								Aggregation		Multi-hop Tracing	Avg.
	S-NIAH-1	S-NIAH-2	S-NIAH-3	MK-NIAH-1	MK-NIAH-2	MK-NIAH-3	MQ-NIAH	MV-NIAH	CWE	FWE	VT	
Mistral-7B-Instruct												
Full	97.00	51.40	59.00	65.80	80.60	1.20	60.15	75.55	33.34	81.27	23.44	57.16
KV Size = 128												
SnapKV	29.40	8.20	0.20	6.00	3.80	0.00	0.05	2.90	4.04	30.53	7.16	8.39
PyramidKV	33.40	**9.60**	0.00	**7.40**	4.20	0.00	0.10	**3.65**	**4.48**	32.00	8.08	9.36
StreamingLLM	0.20	0.80	**3.00**	2.20	0.60	0.00	**1.50**	1.65	0.18	**53.07**	0.16	5.76
EvolKV	**46.60**	6.20	0.00	6.00	**8.60**	0.00	0.00	1.45	3.36	31.80	**9.80**	**10.35**
KV Size = 1024												
SnapKV	94.80	45.20	2.80	33.40	28.80	0.00	9.05	9.20	21.54	58.60	23.44	29.71
PyramidKV	**95.60**	44.60	2.20	30.60	**29.60**	0.00	**10.15**	**9.80**	13.94	54.13	**23.60**	28.57
StreamingLLM	2.20	1.80	**5.60**	4.00	2.80	0.00	2.40	2.45	0.16	**91.53**	2.80	10.52
EvolKV	94.60	**47.00**	3.00	**40.20**	24.80	0.00	10.05	9.50	**22.02**	58.33	23.04	**30.23**
Llama-3-8B-Instruct												
Full	99.40	98.00	97.40	95.20	87.00	95.00	98.95	96.15	97.94	85.20	40.08	90.03
KV Size = 128												
SnapKV	75.80	72.40	1.40	29.80	0.40	0.00	20.65	10.55	**3.36**	48.00	**10.24**	24.78
PyramidKV	81.20	69.60	1.40	**35.20**	0.60	0.00	23.15	**10.75**	2.50	46.93	9.48	25.53
StreamingLLM	1.60	1.40	**2.20**	1.80	**1.00**	**0.40**	2.30	2.85	1.72	**92.07**	2.68	10.00
EvolKV	**90.00**	**79.20**	1.80	33.40	0.60	0.00	**26.90**	10.35	1.88	44.40	8.52	**27.00**
KV Size = 1024												
SnapKV	**100.00**	93.00	2.20	82.20	27.20	6.00	96.50	58.50	**56.70**	77.87	**51.88**	59.28
PyramidKV	**100.00**	**97.40**	2.20	82.60	18.20	3.40	96.30	52.90	44.80	73.27	49.84	56.45
StreamingLLM	13.20	12.80	**11.80**	15.00	11.20	11.00	14.20	13.15	22.04	**90.93**	17.68	21.18
EvolKV	**100.00**	95.40	2.80	**86.00**	**34.60**	**18.40**	**97.70**	**75.50**	53.94	77.47	49.92	**62.88**

Table 4: Comparison of KV cache compression methods on Mistral-7B-Instruct (top) and Llama-3-8B-Instruct (bottom) across RULER tasks. EvolKV consistently outperforms all baseline methods across KV cache budgets of 128 and 1024, achieving the highest average performance in both settings.



(a) Comparison of EvolKV's layer-wise KV cache budgets with existing KV cache compression methods.

(b) EvolKV optimizes layer-wise KV caches with target averages $c \in \{128, 256, 512, 1024\}$.

(c) KV cache budget heatmap (compared to the maximum of layer KV cache budget).

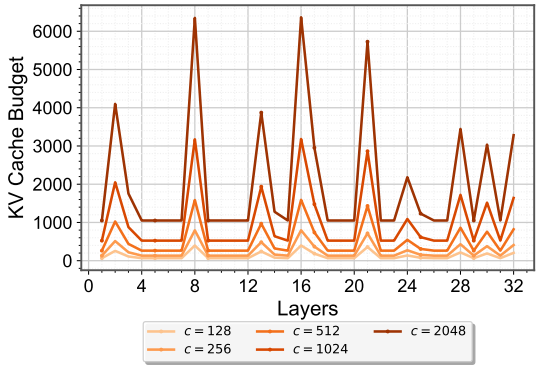
Figure 2: KV cache budget allocation of EvolKV—comparison, optimization trajectory, and allocation heatmap.

cache budget c is 128 and the optimization objective is the recall score. During the evaluation, the KV cache budget is fixed at 128. Figure 7 (Appendix B.2) presents the results: compared to the baselines, EvolKV achieves an improvement of over 4 percentage points on Llama-3-8B-Instruct and a substantial gain of more than 13 percentage points on Mistral-7B-Instruct. These results demonstrate that EvolKV effectively explores and leverages the latent layer-wise KV cache allocation of the model in long-context retrieval.

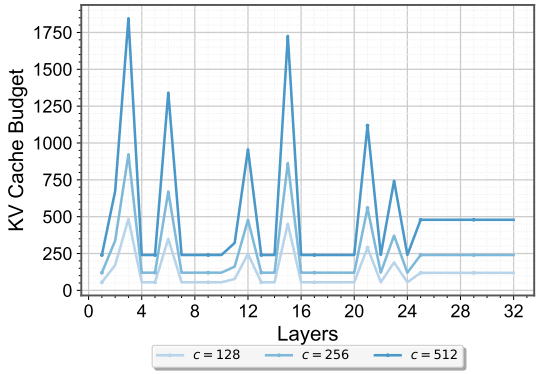
We further evaluate the KV cache allocations optimized in NIAH on the RULER benchmark, using Mistral-7B-Instruct and Llama-3-8B-Instruct with input context lengths set to 32K and 8K, respectively. Notably, the 1024 KV budget is extrapolated based on results from 128. As shown in Table 4, EvolKV consistently outperforms all baselines in average scores, with improvements of up to 0.99 points on Mistral-7B-Instruct and 3.6 points on Llama-3-8B-Instruct. These results further demonstrate the strong generalization, long-context retrieval and reasoning abilities of EvolKV, as the optimized KV budget can be effectively transferred to other benchmark evaluations, suggesting that EvolKV reveals the latent layer-wise allocation strategy.

4.3 Analysis

Downstream Task Performance during Optimization We randomly sample 30 instances from NarrativeQA as optimization data and conduct KV cache budget optimization experiments on Mistral-7B-Instruct, under target average cache budgets $c = 128$, 256, 512, and 1024. EvolKV can effectively optimize per-layer KV cache budget for downstream tasks during optimization. The results (depicted in Figure 2b) show a steady increase in model performance on the train data as the number of iterations grows. This suggests that the original, uniform per-layer KV cache budget allocation leaves substantial room for improvement on downstream tasks and that simple rule-based heuristics are insufficient to identify an optimal KV cache budget distribution.



(a) Mistral-7B-Instruct's layer KV cache budgets optimized in 30 instances of NarrativeQA dataset.



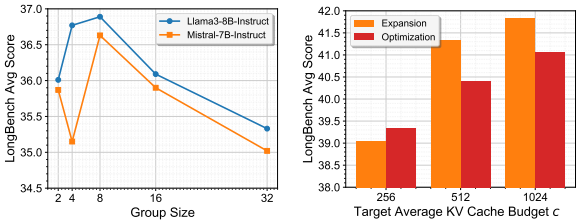
(b) Llama-3-8B-Instruct's layer KV cache budgets optimized in 30 instances of GSM8K trainset data.

Figure 3: Visualization of Mistral-7B-Instruct's layer KV cache budgets in LongBench and Llama-3-8B-Instruct' budgets in GSM8K.

Discussion on Optimized Budget Allocation We optimize EvolKV on Mistral-7B-Instruct with randomly selected 30 NarrativeQA instances at $c = 128, 256, 512$ and 1024. Our experiments reveal that EvolKV discovers totally distinct KV cache allocation patterns compared to heuristic approaches, as shown in Figure 2a. We observe consistent budget peaks at middle-model layers, suggesting these layers serve as computation bottlenecks for contextual inference. This non-intuitive pattern persists across tasks and budgets, showing that fixed heuristics inherently fail to capture transformer's dynamic layer utilization patterns. The allocation exhibits substantial variation across layers, depending on the target average cache budget c, as shown in Figure 2c. Under low-budget conditions (e.g., $c = 128$), the optimization tends to concentrate cache budget resources on a small subset of layers. In contrast, higher budgets result in a more distributed allocation across the model. These patterns reveal that simple rule-based heuristics fail to capture the model's latent cache-size preferences. Notably, as shown in Figure 2a, task-optimized KV cache allocations consistently diverge from rule-based heuristics, favoring non-uniform distributions that deviate from fixed or pyramidal rules. The cache-size requirements do not follow a pyramidal or monotonically decreasing pattern: several lower layers receive minimal allocation, while some higher layers are assigned significantly larger budgets, underscoring their greater relevance to downstream performance.

Effect of Group Size To construct the optimization dataset, we randomly sample five instances from each of the six LongBench subsets—NarrativeQA, HotpotQA, QMSum, TREC, PassageRetrieval-en, and LCC—for Llama-3-8B-Instruct. For Mistral-7B-Instruct, we sample 30 instances exclusively from NarrativeQA. We subsequently apply EvolKV to optimize the KV cache budgets for both models under a target average budget of $c = 128$. During optimization, the group size n_g—i.e., the number of layers optimized concurrently—is set to 2, 4, 8, 16, and 32. The corresponding population sizes are determined using the empirical formula $4 + \lfloor 3 \cdot \ln(n_g) \rfloor$, yielding values of 6, 8, 10, 12, and 14, respectively. As shown in Figure 4a and detailed results in Table 7 (Appendix C.1), downstream task performance generally improves as the group size increases, peaking at $n_g = 8$. Beyond this point, performance degrades notably. The inferior results with smaller group sizes likely stem from overfitting the layer-wise KV cache budget to limited optimization training data, while excessively large groups hinder effective budget allocation across layers. Accordingly, we select $n_g = 8$ as the optimal trade-off between performance and optimization efficiency.

Robustness Analysis of EvolKV Using Llama-3-8B-Instruct and a randomly sampled set of 30 NarrativeQA instances, we evaluate the stability of EvolKV at $c = 128$. As detailed results reported in Table 8 (Appendix C.2), after three optimization rounds, the grand average reaches 35.88. The deviation of each individual average from this overall average is within 0.1 and the standard deviation is 0.078, indicating that EvolKV exhibits consistent and stable behavior in all trials. Additionally, We explore the impact of the selection of evolution training data on Mistral-7B-Instruct. Specifically, we randomly sample 30 instances drawn from six subsets (five items each) across the single- and multi-document QA categories. In parallel, we select another 30 instances, again five per subset, from six subsets (NarrativeQA, HotpotQA, QMSum, TREC, PassageRetrieval-en, and LCC). The experimental results are summarized in Table 9 (Appendix C.3), where only the results obtained after removing the corresponding training data are reported. For the single- and multi-document QA group, we evaluate configurations with $c = 512$ and 1024. While the overall performance is slightly lower than the main results, the $c = 512$ setting still outperforms the baselines, and the $c = 1024$ setting shows only a marginal drop. For the six-subsets group, we similarly examine $c = 1024$ and 2048, both of which yield average scores that exceed the baselines. These results highlight the robustness of EvolKV and its strong adaptability across downstream tasks.



(a) Comparison results of different group sizes.

(b) Comparison of expansion and optimization KV cache.

Figure 4: Effects of layer grouping and KV cache expansion vs. KV cache optimization on LongBench.

Generalization Analysis of EvolKV We optimize the layer-wise KV cache budgets of Mistral-7B-Instruct on a 30-sample subset of NarrativeQA with target average KV cache budget $c = 256, 512,$ and 1024. The results are shown in Figure 4b and the detailed results are shown in Table 10 in Appendix C.4. As c increases, direct cache expansion even outperforms direct optimization when $c = 1024$ and 2048. This finding suggests that when optimizing under strict budget constraints, EvolKV is able to effectively reveal the actual budget required for each layer and is able to try to generalize to higher target average budgets. Furthermore, we evaluate the KV cache budgets optimized at $c = 128$ on the NIAH dataset by applying them to LongBench using Llama-3-8B-Instruct. The configuration for $c = 256$ is derived from the $c = 128$ allocation using the method described in Section 3.2. As shown in Table 11 (Appendix C.5), EvolKV consistently outperforms baselines under both KV cache budget settings, indicating its ability to transfer cache budget allocations across datasets and demonstrating the ability of generalization.

Experiments on Models from Different Series We conduct KV cache budget optimization on the Qwen (Bai et al., 2023) series models at $c = 128$ using 30 randomly sampled NarrativeQA instances. As shown in Table 5, with all training data excluded, EvolKV outperforms all other compression methods in terms of average score across both single- and multi-document QA tasks, demonstrating its consistent advantage across different model series. Notably, PyramidKV performs significantly worse than EvolKV, suggesting pyramidal budget allocation is suboptimal and highlighting EvolKV's superior adaptability to downstream tasks.

5 Conclusion

We introduce EvolKV, a task-driven framework that leverages evolution algorithm to optimize layer-wise KV cache budgets in LLMs. Unlike rule-based or heuristic methods, EvolKV directly maximizes downstream performance (e.g., accuracy, F1) without modifying model parameters and requires only a handful of labeled examples. Extensive experiments demonstrate that evolutionary, task-aware cache budget allocation uncovers latent layer-importance patterns overlooked by existing methods, and delivers state-of-the-art performance under stringent cache constraints and scaling gracefully to larger target cache budgets. Thus, EvolKV provides a practical, plug-and-play solution for efficient inference in downstream tasks. In the future, it would be valuable to investigate the robustness of KV cache compression under different tokenization schemes, as well as tokenization-free approaches (Rust et al., 2022; Chai et al., 2024a,b; Cao et al., 2023).

Method	Single-Document QA			Multi-Document QA			Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	
Qwen2.5-1.5B-Instruct							
Full	10.89	32.86	47.66	42.66	36.45	20.57	31.85
SnapKV	9.13	18.38	**37.91**	37.04	31.52	**14.87**	24.81
PyramidKV	9.23	17.40	35.85	36.11	31.54	12.46	23.77
StreamingLLM	6.99	17.23	23.97	28.35	29.82	8.20	19.09
EvolKV	**9.34**	**18.49**	37.80	**37.62**	**32.98**	14.25	**25.08**
Qwen2.5-3B-Instruct							
Full	11.33	37.30	49.26	46.68	37.64	21.00	33.87
SnapKV	**10.41**	23.58	37.49	**39.85**	32.72	**14.65**	26.45
PyramidKV	8.36	21.01	33.52	32.66	30.19	11.24	22.83
StreamingLLM	8.11	19.98	23.52	32.60	28.95	10.26	20.57
EvolKV	9.68	**24.93**	**38.70**	39.58	**33.43**	13.77	**26.68**

Table 5: Comparison of KV cache compression methods on Qwen models across LongBench single- and multi-document tasks. EvolKV outperforms all baselines on average KV cache budget $c = 128$.

Limitations

While EvolKV demonstrates strong performance on various downstream tasks, there is still room to explore budget allocation at the attention-head level. Future work will focus on combining downstream task performance with attention-head budgets.

References

Tanweer Alam, Shamimul Qamar, Amit Dixit, and Mohamed Benaida. 2020. Genetic algorithm: Reviews, implementations, and applications.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report.

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.

Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2015. Parameter setting for multicore cma-es withźlarge populations. In *Revised Selected*

Papers of the 12th International Conference on Artificial Evolution - Volume 9554, page 109–122, Berlin, Heidelberg. Springer-Verlag.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling.

Qi Cao, Takeshi Kojima, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Unnatural error correction: GPT-4 can almost perfectly handle unnatural scrambled text. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8898–8913, Singapore. Association for Computational Linguistics.

Yekun Chai, Yewei Fang, Qiwei Peng, and Xuhong Li. 2024a. Tokenization falling short: On subword robustness in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1582–1599, Miami, Florida, USA. Association for Computational Linguistics.

Yekun Chai, Qingyi Liu, Jingwu Xiao, Shuohuan Wang, Yu Sun, and Hua Wu. 2024b. Autoregressive pretraining on pixels and texts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3106–3125, Miami, Florida, USA. Association for Computational Linguistics.

Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2022. Clip-tuning: Towards derivative-free prompt learning with a mixture of rewards. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 108–117, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

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.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers.

Alexander 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*, pages 1074–1084, Florence, Italy. Association for Computational Linguistics.

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.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and et al Amy Yang. 2024. The llama 3 herd of models.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion.

Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps.

John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models?

Hui Huang, Bing Xu, Xinnian Liang, Kehai Chen, Muyun Yang, Tiejun Zhao, and Conghui Zhu. 2024. Multi-view fusion for instruction mining of large language model. *Information Fusion*, 110:102480.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

John R. Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2017. The narrativeqa reading comprehension challenge.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023a. Repobench: Benchmarking repository-level code auto-completion systems.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023b. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, and et al Mohammad Bavarian. 2024. Gpt-4 technical report.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741, Miami, Florida, USA. Association for Computational Linguistics.

Phillip Rust, Jonas F Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. 2022. Language modelling with pixels. *arXiv preprint arXiv:2207.06991*.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark W. Barrett, Joseph Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm' s kv-cache consumption.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. Layer by layer: Uncovering hidden representations in language models.

Rainer Storn and Kenneth Price. 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, and et al Lukas Blecher. 2023. Llama 2: Open foundation and fine-tuned chat models.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multihop questions via single-hop question composition.

W. Vent. 1975. Rechenberg, ingo, evolutionsstrategie — optimierung technischer systeme nach prinzipien der biologischen evolution. 170 s. mit 36 abb. frommann-holzboog-verlag. stuttgart 1973. broschiert. *Feddes Repertorium*, 86(5):337–337.

Wenxuan Wang and Zhaopeng Tu. 2020. Rethinking the value of transformer components.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering.

Yang Zhang, Yanfei Dong, and Kenji Kawaguchi. 2024. Investigating layer importance in large language models.

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. H_2o: Heavy-hitter oracle for efficient generative inference of large language models.

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.

A Datasets

In this study, we apply LongBench (Bai et al., 2024), GSM8K (Cobbe et al., 2021), Needle-in-a-Haystack[4] and RULER (Hsieh et al., 2024) to evaluate EvolKV. LongBench is a bilingual, multitask benchmark containing 21 datasets across six categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. GSM8K (Grade School Math 8K) is a dataset comprising 8,792 high-quality grade school-level math word problems, including 7,473 for training and 1,319 for testing. The Needle-In-A-Haystack test is an evaluation method that measures a language model's ability to retrieve specific information ("needle") embedded within extensive, potentially distracting textual contexts ("haystack"). RULER is a long-context benchmark that is substantially more complex than NIAH, comprising four major tasks: retrieval, aggregation, multi-hop tracing, and question answering. Compared with NIAH, which mainly focuses on single retrieval accuracy, RULER emphasizes reasoning over long sequences and tests a model's ability to handle compositional challenges such as integrating multiple retrieved pieces of evidence, following reasoning chains across documents, and synthesizing information over extended contexts. In this paper, we do not evaluate the question answering task, as one of its source datasets overlaps with LongBench. The detailed dataset information is presented in Table 6.

B Visualization results

B.1 Performance Results Across the Six Major Task Categories in LongBench

In this section, we present the performance of EvolKV and other baselines across the six major task categories in LongBench. As shown in Figure 5, EvolKV consistently outperforms all baselines in terms of average scores across the six categories. Figure 6 also demonstrates that EvolKV maintains an overall advantage across various task types. EvolKV consistently demonstrates superior performance across major task categories. Under both low and high KV cache budget settings, it maintains a leading position across various task types and even surpasses baseline methods under extremely constrained budgets in tasks such as synthetic, few-shot learning, and code. Notably, most of these results are based on the expanded KV cache budgets, indicating that EvolKV effectively discovers layer-wise cache allocation patterns tailored to downstream tasks, thereby highlighting its evolutionary and task-aware advantages.

[4] https://github.com/gkamradt/LLMTest_NeedleInAHaystack

B.2 Performance Results on NIAH

We evaluate EvolKV against other rule-based methods on NIAH at $c = 128$, as shown in Figure 7. EvolKV consistently surpasses all baselines, achieving improvements of up to 13% over the strongest competitor. In contrast, StreamingLLM exhibits weak performance on long-context retrieval, highlighting the inherent limitations of fixed-position methods and their tendency to lose critical contextual information. Notably, on Mistral-7B-Instruct with a 32K context length, SnapKV and PyramidKV already degrade significantly at 8K, whereas EvolKV maintains robust and superior retrieval ability across longer contexts.

C Detailed Results of Analysis

C.1 Comparison Results of Different Group sizes

Table 7 presents the detailed downstream task performance across different group sizes. The overall performance peaks at $n_g = 8$, which motivates our choice of setting the group size to 8 in practical experiments. Meanwhile, we argue that with only 30 training samples for optimization, overly small group sizes may cause EvolKV to overfit the limited data when allocating per-layer budgets, while overly large group sizes force EvolKV to handle too many layers simultaneously, making it difficult to perform fine-grained and effective allocation. Given that the models used in our experiments all have 32 layers, setting the group size to 8 serves as an optimal choice.

C.2 Optimization Results Across Three Rounds

We present in Table 8 the results of three optimization runs using EvolKV on the Llama-3-8B-Instruct model, conducted on 30 randomly sampled instances from the NarrativeQA dataset. The overall average score is 35.88, with a variance of 0.078, demonstrating the stable optimization performance of EvolKV.

C.3 Comparison Results of Different Selection of Training Data for Optimization

We present in Table 9 the results of EvolKV optimized with different training data configurations. The training data consists of two settings: the single-multi doc set, comprising 30 examples sampled from the single- and multi-document QA categories, and the six-subsets set, constructed from samples drawn from six datasets—NarrativeQA, HotpotQA, QMSum, TREC, PassageRetrieval-en, and LCC. The optimized results suggest that EvolKV holds a performance advantage, indicating its potential to maintain effective results across diverse downstream training data and demonstrating its strong adaptability.

C.4 Comparison Results of KV Cache Budgets Expansion and Optimization

Table 10 compares the performance of two approaches: expanding KV cache budgets using the method described in Section 3.2 versus directly optimizing at the

	Datasets	Source	Avg len	Metric	Language	#Sample
LongBench	*Single-Document QA*					
	NarrativeQA	Literature, Film	18,409	F1	English	200
	Qasper	Science	3,619	F1	English	200
	MultiFieldQA-en	Multi-field	4,559	F1	English	150
	Multi-Document QA					
	HotpotQA	Wikipedia	9,151	F1	English	200
	2WikiMultihopQA	Wikipedia	4,887	F1	English	200
	MuSiQue	Wikipedia	11,214	F1	English	200
	Summarization					
	GovReport	Government report	8,734	ROUGE-L	English	200
	QMSum	Meeting	10,614	ROUGE-L	English	200
	MultiNews	News	2,113	ROUGE-L	English	200
	Few-shot Learning					
	TREC	Web question	5,177	Accuracy (CLS)	English	200
	TriviaQA	Wikipedia, Web	8,209	F1	English	200
	SAMSum	Dialogue	6,258	ROUGE-L	English	200
	Synthetic Reasoning					
	PassageCount	Wikipedia	11,141	Accuracy (EM)	English	200
	PassageRetrieval-en	Wikipedia	9,289	Accuracy (EM)	English	200
	Code Completion					
	LCC	GitHub	1,235	Edit Sim	Python/C#/Java	500
	RepoBench-P	GitHub repository	4,206	Edit Sim	Python/Java	500
RULER	*Retrieval*					
	Single NIAH					
	S-NIAH-1	–	–	Recall-based Accuracy	English	500
	S-NIAH-2	–	–	Recall-based Accuracy	English	500
	S-NIAH-3	–	–	Recall-based Accuracy	English	500
	Multi-keys NIAH					
	MK-NIAH-1	–	–	Recall-based Accuracy	English	500
	MK-NIAH-2	–	–	Recall-based Accuracy	English	500
	MK-NIAH-3	–	–	Recall-based Accuracy	English	500
	Multi-queries NIAH (MQ-NIAH)	–	–	Recall-based Accuracy	English	500
	Multi-values NIAH (MV-NIAH)	–	–	Recall-based Accuracy	English	500
	Aggregation					
	Common Words Extraction (CWE)	–	–	Recall-based Accuracy	English	500
	Frequent Words Extraction (FWE)	–	–	Recall-based Accuracy	English	500
	Multi-hop Tracing					
	VT	–	–	Recall-based Accuracy	English	500
Needle-in-a-Haystack		PaulGrahamEssays	–	Recall	English	–
GSM8K		Grade-school math word problems	239	Accuracy	English	1319

Table 6: Datasets introduction. "Accuracy (CLS)" denotes classification accuracy; "Accuracy (EM)" denotes exact-match accuracy. "Recall-based Accuracy" denotes the proportion of reference strings that appear in the model output.

corresponding target average budget. It can be observed that when $c \geq 512$, the expansion-based method consistently outperforms direct optimization. This motivates our preference for using the expansion strategy to obtain target budgets rather than performing direct optimization. These results also demonstrate the simplicity of EvolKV's optimization process—KV cache budget allocations optimized under low-budget settings can be directly extended to higher budgets without re-tuning, and even outperform directly optimized high-budget results in some tasks. This suggests that the low-budget phase enables more thorough exploration of task-aware per-layer KV cache allocation.

C.5 Transfer Results of KV Cache Budget Optimization to Other Datasets

Table 11 presents the evaluation results on the LongBench benchmark by directly applying the KV cache budget scheme optimized on the Needle-in-a-Haystack (NIAH) dataset using the Llama-3-8B-Instruct model. Notably, even without further adaptation, EvolKV consistently achieves the highest average performance across downstream tasks under low-budget constraints ($c = 128$ and 256), demonstrating its robust generalization capability in transferring optimized budget allocations from a single dataset to diverse task settings. Furthermore, the budget configuration at $c = 256$ is derived by directly expanding the allocation optimized under $c = 128$, further reinforcing the scalability of EvolKV's budget allocation results, as discussed in Section C.4. This not only emphasizes the method's efficiency in low-resource settings but also highlights its potential for progressive extension to higher budgets without retraining, offering a compelling advantage in practical deployment scenarios.

C.6 Experiments on Specialized Task Optimization

To evaluate the specialized task optimization performance of EvolKV, we perform KV cache budget opti-

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
Llama-3-8B-Instruct																	
EvolKV-n_g2	20.64	**15.22**	**34.71**	38.48	30.07	18.84	**18.40**	21.36	19.49	48.50	88.52	37.14	4.58	**69.00**	56.99	54.18	36.01
EvolKV-n_g4	**22.27**	15.04	33.35	40.59	31.77	19.59	18.29	**21.88**	**20.05**	54.00	**89.33**	**37.47**	4.48	68.75	57.02	54.40	36.77
EvolKV-n_g8	21.40	13.12	33.64	41.30	**32.42**	**19.81**	17.29	21.35	19.03	**58.00**	89.30	37.38	**5.00**	68.67	**57.49**	**55.11**	**36.89**
EvolKV-n_g16	21.62	13.31	34.26	**41.64**	31.76	18.37	17.07	21.22	19.20	48.50	88.30	36.81	4.65	68.75	57.13	54.87	36.09
EvolKV-n_g32	19.74	13.62	32.13	38.62	28.77	16.78	17.25	21.26	18.87	51.50	87.85	37.01	4.86	**69.00**	55.32	52.71	35.33
Mistral-7B-Instruct																	
EvolKV-n_g2	20.33	22.09	**45.64**	38.19	23.01	15.82	19.65	21.66	21.10	49.00	85.17	40.34	2.71	69.14	52.35	47.67	35.87
EvolKV-n_g4	21.26	20.21	41.92	36.61	21.10	15.66	18.80	21.75	20.79	46.00	84.86	**40.57**	2.19	72.24	50.88	47.60	35.15
EvolKV-n_g8	22.72	**22.59**	44.02	**39.47**	**24.16**	15.64	19.90	21.93	21.20	52.00	**86.83**	39.83	2.35	**74.81**	51.64	47.05	**36.63**
EvolKV-n_g16	**23.57**	22.09	43.32	38.61	24.07	**16.31**	**20.42**	**22.09**	**21.63**	**58.00**	84.70	39.16	2.89	55.50	**54.11**	**48.00**	35.90
EvolKV-n_g32	20.97	20.09	43.24	37.00	22.19	14.95	19.01	21.99	20.85	45.50	86.49	39.11	**3.77**	67.37	50.64	47.08	35.02

Table 7: Comparison results of different group size (n_g). We use the empirical formula $4 + \lfloor 3 \cdot \ln(n_g) \rfloor$ to obtain the population size.

Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
21.96	12.24	**35.47**	37.92	**29.68**	**20.87**	18.76	**22.06**	**20.16**	**45.00**	**88.72**	38.40	3.98	68.27	56.96	55.19	**35.98**
22.40	11.88	33.04	**39.96**	29.27	19.97	18.93	21.83	19.78	43.50	87.53	**38.42**	**4.50**	**69.00**	**58.39**	55.39	35.86
21.34	**12.42**	32.36	36.06	29.60	19.18	**19.01**	21.37	19.99	44.50	88.20	**38.42**	4.35	68.60	57.10	**60.11**	35.79

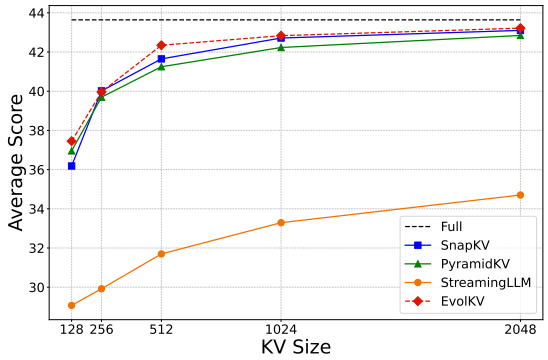
Table 8: On Llama-3-8B-Instruct, 30 randomly selected instances from the NarrativeQA dataset are used to perform EvolKV layer-wise KV cache budget optimization at $c = 128$ three times to evaluate stability.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
Single-Multi Doc																	
KV Size = 512																	
SnapKV	**24.44**	**28.14**	48.09	40.16	25.09	16.45	23.73	23.57	**24.63**	67.00	86.12	41.43	2.47	88.06	56.37	**52.80**	40.53
PyramidKV	23.20	27.83	48.15	**40.77**	25.08	15.78	23.44	23.48	24.05	67.00	85.87	41.42	2.86	**86.23**	55.62	51.88	40.17
StreamingLLM	21.16	16.48	30.25	30.83	22.64	10.34	21.53	20.02	23.10	61.50	81.86	41.72	**3.14**	18.57	55.16	48.65	31.68
EvolKV	23.72	27.98	**48.41**	40.69	**26.02**	**17.13**	**23.87**	**23.64**	24.62	**68.50**	**86.52**	**42.30**	2.93	83.06	**56.99**	52.44	**40.55**
KV Size = 1024																	
SnapKV	**25.23**	29.45	48.54	**41.45**	25.51	18.04	**26.19**	23.99	**25.99**	69.50	**86.63**	**43.01**	2.84	**89.21**	**57.41**	53.25	**41.64**
PyramidKV	24.27	**30.26**	48.09	40.88	**26.39**	17.88	25.27	23.66	25.52	69.00	86.31	42.20	2.66	86.67	56.39	53.38	41.18
StreamingLLM	22.25	18.70	30.38	33.01	23.02	11.69	24.09	20.75	25.54	64.00	84.71	41.26	**3.49**	22.40	55.89	50.99	33.26
EvolKV	24.89	29.91	**48.84**	41.28	26.03	**18.58**	25.94	**24.20**	25.92	**70.00**	86.14	41.90	2.66	87.93	57.23	**53.64**	41.57
Six-Subsets																	
KV Size = 1024																	
SnapKV	**25.49**	29.15	49.03	**40.59**	25.30	18.96	**26.19**	**23.86**	25.99	69.23	**86.63**	**43.01**	2.84	**88.94**	57.24	53.25	41.61
PyramidKV	24.22	29.97	48.72	40.03	**25.85**	18.53	25.27	23.51	25.52	69.23	86.31	42.20	2.66	86.33	56.20	**53.38**	41.12
StreamingLLM	22.22	18.51	31.03	32.79	22.57	11.85	24.09	20.65	25.54	64.10	84.71	41.26	**3.49**	21.44	55.71	50.99	33.18
EvolKV	24.74	**30.53**	**49.44**	40.15	25.58	**19.60**	26.12	23.61	**26.16**	**70.26**	86.53	42.06	2.52	88.27	**57.58**	53.37	**41.66**
KV Size = 2048																	
SnapKV	**25.73**	32.65	**49.09**	40.78	27.39	18.49	28.77	**24.34**	26.55	69.74	86.27	**42.47**	**2.79**	**87.24**	57.24	53.41	42.06
PyramidKV	22.14	23.08	35.22	32.94	22.90	13.47	26.85	20.85	26.45	66.15	85.68	41.95	2.40	24.87	56.95	52.17	34.63
StreamingLLM	25.32	31.33	48.89	40.92	26.64	17.65	28.09	23.70	26.52	**70.77**	**86.30**	42.27	2.52	86.51	57.24	**53.66**	41.77
EvolKV	25.28	**32.73**	48.83	**40.94**	**27.67**	**18.97**	**28.93**	24.17	**26.94**	69.74	86.27	42.12	2.64	86.94	**57.25**	53.65	**42.07**

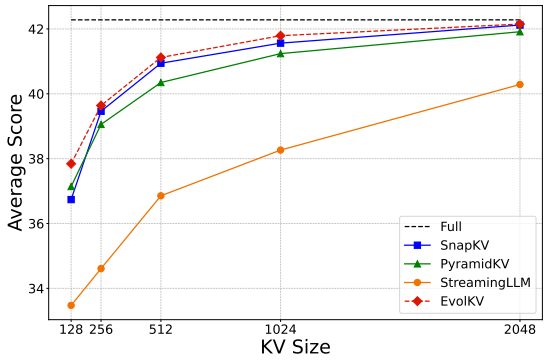
Table 9: Comparison results of different selection of training data for EvolKV optimization. We report the results that removed training data.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
KV Size = 256																	
EvolKV.ex	22.12	**24.50**	**48.28**	**40.26**	**25.74**	**17.39**	**22.33**	**22.39**	**23.61**	**65.50**	84.57	40.66	2.85	79.35	54.10	51.20	39.05
EvolKV.opt	**24.53**	24.14	47.14	40.04	24.54	17.23	21.62	22.85	23.09	**65.50**	**86.50**	**40.88**	**3.69**	**81.17**	**54.85**	**51.59**	**39.34**
KV Size = 512																	
EvolKV.ex	24.99	**29.00**	**49.78**	**41.57**	**26.27**	**18.34**	**24.41**	**23.18**	**25.00**	**68.00**	**87.07**	41.64	2.87	**89.74**	**56.58**	**52.85**	**41.33**
EvolKV.opt	**25.69**	27.59	48.79	41.16	25.97	17.16	22.65	22.90	24.21	67.50	86.27	**42.16**	**3.55**	82.77	56.04	51.99	40.40
KV Size = 1024																	
EvolKV.ex	25.51	**30.30**	48.96	**42.84**	25.78	**18.21**	**26.99**	23.79	**25.95**	**70.00**	86.09	41.74	3.06	**89.82**	57.01	**53.26**	**41.83**
EvolKV.opt	**26.29**	29.48	**49.15**	40.36	**26.61**	17.22	24.15	**23.84**	24.70	67.00	**86.45**	**42.69**	**3.29**	85.31	**57.06**	53.16	41.05

Table 10: Comparison results of optimization and expansion. ".ex" denotes using the budget allocation optimized at $c = 128$ to expand to target average KV cache budgets c; ".opt" denotes using EvolKV optimization at the corresponding c.



(a) Average performance comparison across six major task categories in LongBench between baseline methods and our proposed EvolKV on Mistral-7B-Instruct.



(b) Average performance comparison across six major task categories in LongBench between baseline methods and our proposed EvolKV on Llama-3-8B-Instruct.

Figure 5: Average performance comparison across six major task categories in LongBench between baseline methods and our proposed EvolKV on Llama-3-8B-Instruct and Mistral-7B-Instruct.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRE	Lcc	RB-P	
KV Size = 128																	
SnapKV	22.12	**13.51**	30.85	35.98	29.14	19.21	**19.37**	21.67	20.14	45.50	88.32	38.28	4.30	**69.00**	57.39	54.85	35.60
PyramidKV	22.42	13.20	31.54	**39.24**	27.57	**20.18**	19.19	**21.92**	**20.71**	**50.00**	87.34	**38.53**	4.55	**69.00**	**57.57**	54.25	36.08
StreamingLLM	18.96	7.79	20.90	33.13	24.85	14.94	16.37	20.45	18.49	45.50	74.40	35.78	4.75	68.00	55.65	52.32	32.02
EvolKV	**22.78**	13.02	**36.17**	38.33	**31.08**	19.89	18.22	21.57	19.92	48.00	**88.98**	37.39	**5.08**	**69.00**	56.26	**55.65**	**36.33**
KV Size = 256																	
SnapKV	**24.60**	**18.37**	33.36	**43.01**	**34.05**	20.36	**20.62**	22.07	**22.51**	60.00	89.61	**39.22**	5.00	**69.00**	59.47	56.27	38.60
PyramidKV	24.20	15.75	34.15	40.48	29.78	**21.22**	20.11	22.00	22.30	**65.00**	**90.03**	39.16	**5.08**	**69.00**	59.33	53.79	38.21
StreamingLLM	18.90	10.80	20.34	33.54	25.62	15.54	19.29	20.31	20.75	53.00	78.75	39.20	4.83	67.20	58.39	54.50	33.81
EvolKV	22.89	18.27	**35.60**	41.76	33.48	20.54	20.47	**22.68**	22.38	63.50	89.09	39.03	4.51	**69.00**	**59.66**	**56.52**	**38.71**

Table 11: Evaluation of Llama-3-8B-Instruct using KV cache budgets optimized on the Needle-in-a-Haystack from LongBench.

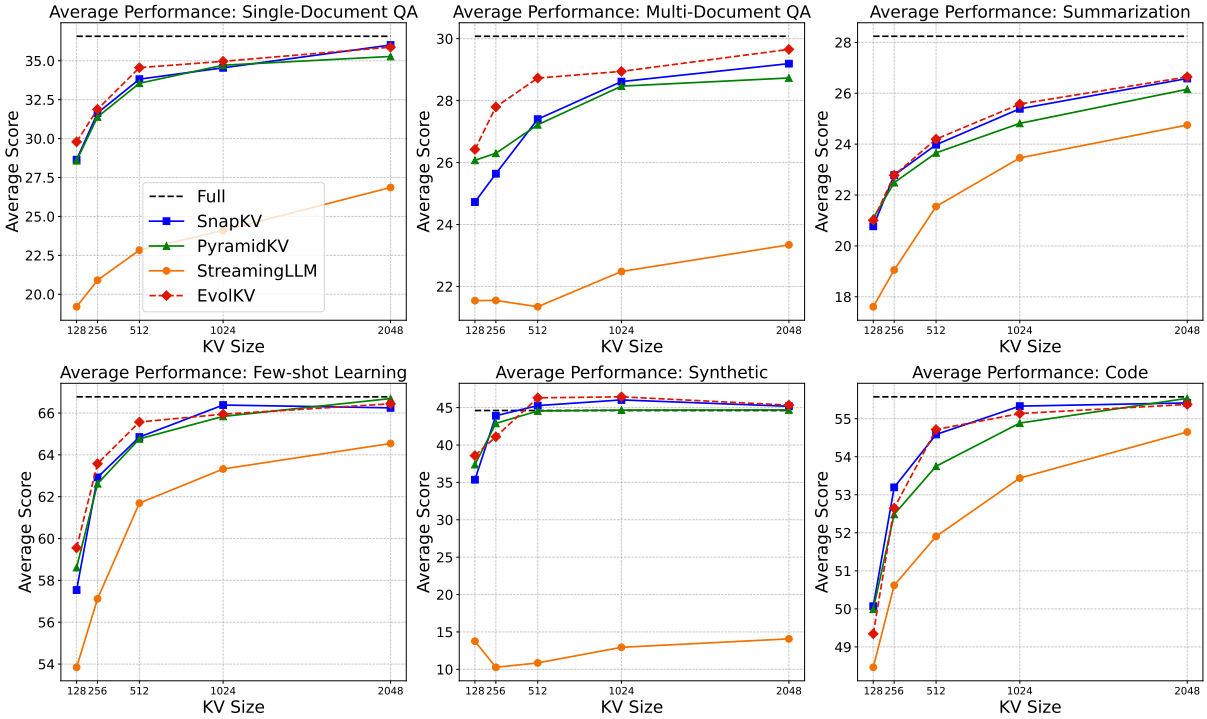
Method	Single-Document QA			Avg.	Multi-Document QA			Avg.
	NrtvQA	Qasper	MF-en		HotpotQA	2WikiMQA	Musique	
Mistral-7B-Instruct								
SnapKV	20.13	21.41	41.67	27.74	36.48	21.53	14.43	24.15
PyramidKV	18.91	**21.67**	**43.17**	27.92	**37.65**	23.47	15.12	25.41
StreamingLLM	16.93	13.47	26.70	19.03	29.91	21.51	11.44	20.95
EvolKV	**21.56**	21.49	42.61	**28.55**	36.79	**24.22**	**16.37**	**25.79**
Llama-3-8B-Instruct								
SnapKV	**21.48**	**13.80**	31.25	22.18	34.71	27.96	19.10	27.26
PyramidKV	21.40	13.55	32.48	22.48	**38.05**	26.75	19.63	28.14
StreamingLLM	18.06	7.81	21.45	15.77	32.13	23.84	14.62	23.53
EvolKV	21.24	13.13	**34.72**	**23.03**	37.86	**28.60**	**20.33**	**28.93**

Table 12: Specialized task optimization results of EvolKV across LongBench single- and multi-document tasks. EvolKV outperforms all baselines on average KV cache budget $c = 128$.

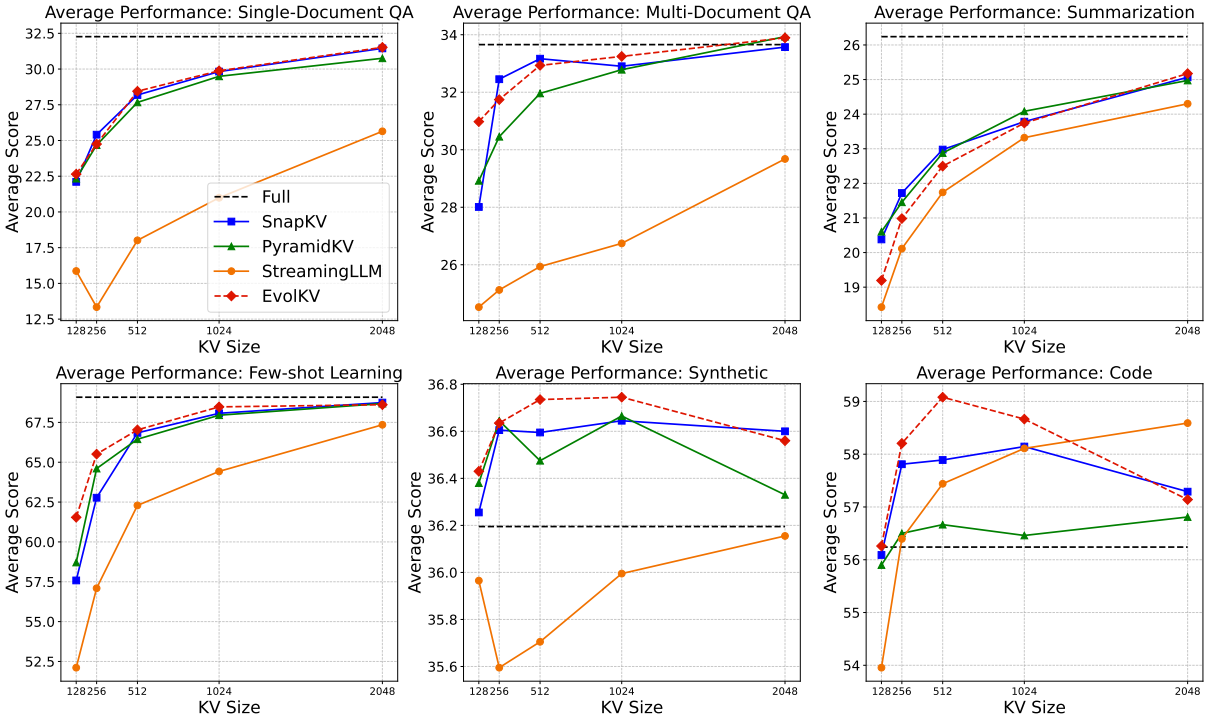
mization on both single-document and multi-document QA tasks, using 30 randomly sampled instances for each setting, with 10 instances per sub-task. As shown in Table 12, after removing all training data, EvolKV achieves the highest average scores at $c = 128$ on both tasks, surpassing all baselines and demonstrating its strong adaptability to downstream tasks.

C.7 Evaluation of Inference Time and Memory Cost

We compare the inference time and peak memory usage of EvolKV against baseline methods on Mistral-7B-Instruct, using the NIAH context materials and FlashAttention (Dao et al., 2022) with a target average KV cache budget $c = 128$. As shown in Figure 8a, compared to other KV cache compression methods, EvolKV exhibits negligible variation in inference time (prefill + decoding) across different generation lengths. Additionally, as shown in Figure 8b, the peak memory usage of EvolKV across different context lengths is comparable to that of other compression methods, while significantly reducing memory consumption compared to the full cache.



(a) Comparison between baseline methods and our proposed EvolKV on Mistral-7B-Instruct across six major task categories in LongBench.



(b) Comparison between baseline methods and our proposed EvolKV on Llama-3-8B-Instruct across six major task categories in LongBench.

Figure 6: Performance comparison of baselines and EvolKV on Mistral-7B-Instruct and Llama-3-8B-Instruct across six LongBench task categories.

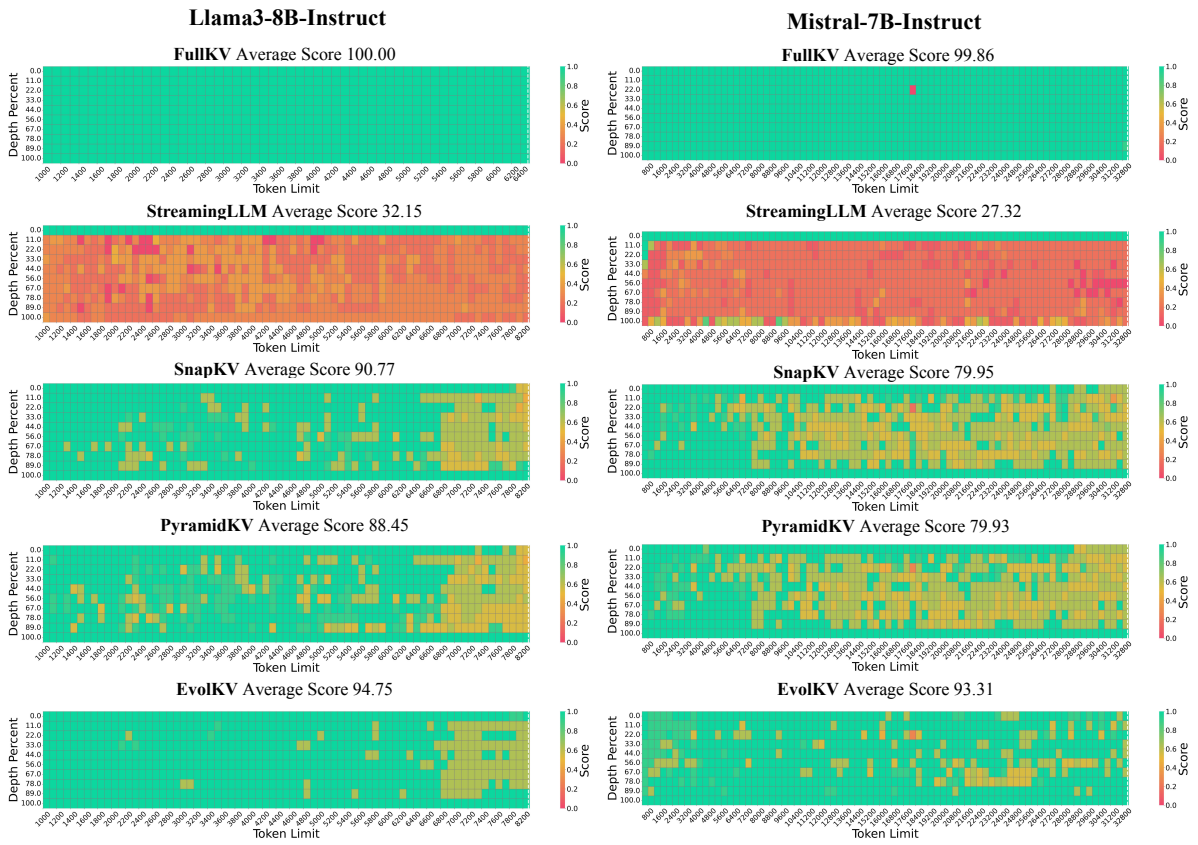
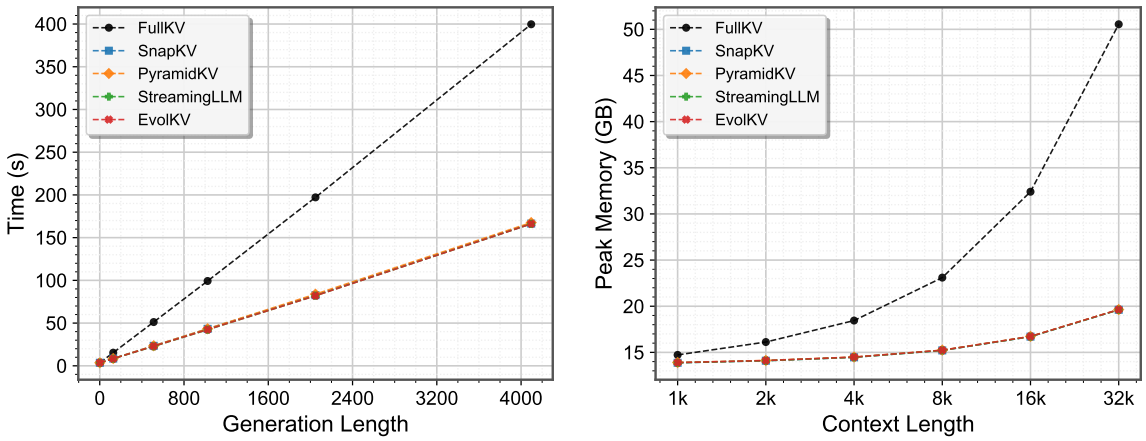


Figure 7: Evaluation of Llama-3-8B-Instruct and Mistral-7B-Instruct on NIAH at a KV cache budget of 128.



(a) Comparison results of inference time.

(b) Comparison results of peak memory usage.

Figure 8: Comparison of inference time and peak memory usage between baseline methods and EvolKV.