DISCO: Dispersion-Guided Sparse KV Cache Compression for Efficient Long-Context Inference in Large Language Models

Anonymous ACL submission

Abstract

Large Language Models (LLMs) face severe memory bottlenecks in long-context inference due to the growth of the Key-Value (KV) cache. Existing approaches rely on attention heuristics and uniform layer-wise budgets, resulting in inefficient memory use and performance loss. We propose DISCO, a dispersion-guided KV cache compression framework that exploits intrinsic layer-wise redundancy in Transformer representations. DISCO grounds redundancy in geometric dispersion, allocates layer-wise KV budgets using PCA-based Dispersion Scores to prioritize information-rich layers, and applies a lightweight token eviction strategy that preserves dispersion by favoring geometrically informative tokens. Extensive experiments demonstrate that DISCO preserves model performance while using only 3.13% of the original KV cache, significantly improving memory efficiency and throughput under fixed-budget settings. We further evaluate DISCO on a real-world medical benchmark, providing evidence that dispersion-based redundancy modeling preserves low-frequency, domain-critical information and remains effective in high-fidelity reasoning settings.

1 Introduction

Recent LLMs such as GPT-5 (Bubeck et al., 2025) and Claude 3.5 (Anthropic, 2024) support extremely long context windows for long-range reasoning (Li et al., 2024a; Taguchi et al., 2025; Hao et al., 2025). Although KV caching alleviates the quadratic cost of self-attention, KV cache memory still scales linearly with sequence length, making efficient KV cache compression essential for practical long-context inference (Sun et al., 2024).

Existing approaches mainly rely on attention-based token eviction and uniform layer-wise budget allocation. However, attention-based eviction (e.g., H2O (Zhang et al., 2023)) is biased by autoregressive masking, overemphasizing early attention sinks and risking the removal of semantically important mid-context tokens (Gu et al., 2025). Uniform budgets further ignore layer-wise functional heterogeneity, leading to inefficient memory use and degraded performance (Jin et al., 2024; Ben-Artzy and Schwartz, 2024; Cai et al., 2024; Li and Vargas, 2024). Non-uniform schemes such as PyramidKV (Cai et al., 2024) partially mitigate this issue but incur per-step allocation overhead that reduces throughput.

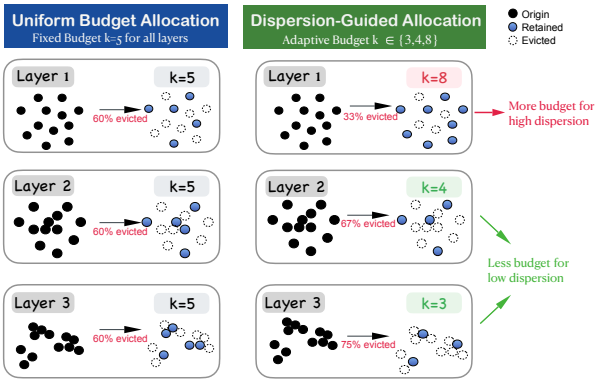


Figure 1: Comparison of uniform and dispersion-guided budget allocation strategies. Uniform allocation assigns equal budget across layers, while dispersion-guided allocation emphasizes performance-critical layers.

We offer a geometric perspective on KV cache compression by asking a largely overlooked question: *Can redundancy be characterized as an input-agnostic, layer-wise geometric property of the model?* Rather than treating redundancy as a transient, input-dependent effect of attention, we find that it is largely an *intrinsic* property of the model. Across diverse inputs, each layer exhibits a remarkably stable geometric distribution of token representations, yielding an input-agnostic redundancy fingerprint. This stability suggests that redundancy is encoded in the layer's representation geometry itself. We therefore measure redundancy through *distributional dispersion*: layers whose representations concentrate in a narrow subspace

are highly redundant and naturally compressible, whereas layers with broader geometric spread preserve critical, non-redundant distinctions. Motivated by this insight, we quantify layer-wise redundancy using a *Dispersion Score* and use it to drive non-uniform KV budgeting that matches cache capacity to a layer's intrinsic information demand.

Building on the layer-wise redundancy structure revealed by dispersion, we introduce **DI**spersion-based **S**parse KV cache **CO**mpression (DISCO), which manages KV memory using a unified geometric criterion at both the layer and token levels. DISCO estimates a stable dispersion profile offline to allocate layer-wise KV budgets without inference overhead, and enforces these budgets online via dispersion-augmented token selection that integrates geometric contribution with attention relevance. Figure 1 illustrates how dispersion-driven allocation matches cache capacity to layer-specific representational demand under a fixed budget. Our contributions are summarized as follows:

1. To our knowledge, we are the first to characterize layer-wise redundancy via geometric dispersion and establish an information-theoretic link to layer-wise information capacity using a log-determinant functional with effective-rank control, showing redundancy to be a stable, intrinsic property of model geometry rather than an attention artifact.

2. We propose DISCO, a dispersion guided KV cache compression framework derived from our empirical and theoretical analysis, using geometric dispersion for layer adaptive budgeting and redundancy aware token selection.

3. We conduct extensive evaluations of DISCO on LongBench and NeedleBench across multiple models, where it consistently outperforms strong baselines across cache budgets and model families, and further demonstrates robust gains on the domain-specific MedConceptsQA benchmark.

2 Revisiting KV Cache Redundancy through Geometric Dispersion

Recent studies (Pizarroso et al., 2022; Li et al., 2021) show that Transformer layers exhibit strong functional heterogeneity in how they process and accumulate information. We find that this heterogeneity is reflected in the KV cache: layers differ substantially in how much meaningful variation they preserve, resulting in highly uneven redundancy across the network. To formalize this observation, we characterize layer-wise redundancy using information dispersion, which measures how broadly token representations are distributed in representation space. We quantify dispersion with a Dispersion Score based on Principal Component Analysis (PCA), capturing the dominant variance structure of KV representations and providing a principled measure of layer-wise redundancy.

Let $Z \in \mathbb{R}^{N \times D}$ denote the mean-centered token representations from a given layer, where N is the number of tokens and D the hidden dimension. The empirical covariance matrix is

$$C = \frac{1}{N} Z^\top Z. \tag{1}$$

Dispersion Score. For layer l, we define its Dispersion Score as the total variance captured by the top-d principal components:

$$D_l(d) = \sum_{i=1}^{d} \lambda_i. \tag{2}$$

We interpret a smaller $D_l(d)$ as indicating that representations concentrate in a low-dimensional subspace and are therefore highly redundant, whereas a larger $D_l(d)$ reflects greater representational diversity. In the following, we empirically examine this relationship.

2.1 Stable Layer-wise Dispersion Reveals Intrinsic Redundancy Patterns

To verify whether layer-wise dispersion reflects an intrinsic model property rather than task-specific behavior, we measure the layer-wise key dispersion score $D_l^K(d)$ across multiple LongBench (Bai et al., 2024) datasets. As shown in Figure 2, the resulting dispersion distributions are nearly identical across tasks. Across layers and diverse input samples, the dispersion curves exhibit a consistent structure that remains invariant to dataset domain or task type. These results indicate that layer-wise dispersion is largely input-agnostic and captures a stable, layer-specific representational characteristic of the model rather than variations induced by particular tasks.

We further examine whether this phenomenon persists across different model architectures. As shown in Figure 3, layer-wise dispersion $D_l(d)$ computed for multiple Transformer models exhibits highly similar patterns, despite substantial

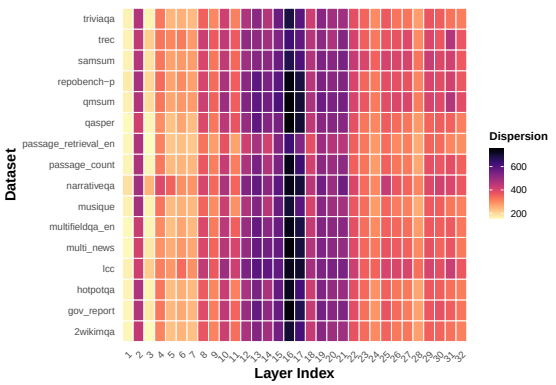


Figure 2: Heatmap of the layer-wise key Dispersion Score $D_l^K(d)$ across datasets.

differences in depth, hidden size, and attention design. This consistency indicates that stable dispersion is not a model-specific artifact, but a general property shared across transformer-based LLMs.

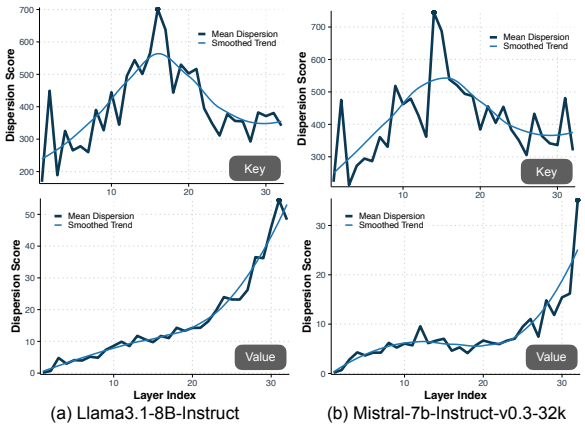


Figure 3: Layer-wise dispersion across models. Stable layer-wise dispersion patterns are consistently observed across different models.

2.2 Dispersion Predicts Functional Redundancy under Token Removal

To test whether the Dispersion Score reflects functional token redundancy, we conduct a controlled subsampling experiment on LLaMA3.1-8B-Instruct, selecting layers with contrasting dispersion levels (high: L15, L16; low: L3, L28). For each layer, we randomly remove KV tokens and measure output distortion via the final-layer hidden state.

Figure 4 shows that high-dispersion layers (L15 and L16) incur substantially larger output distortion at all drop ratios and enter a rapid degradation regime at much lower removal rates, whereas low-dispersion layers (L3 and L28) remain stable until a significantly higher fraction of tokens is removed.

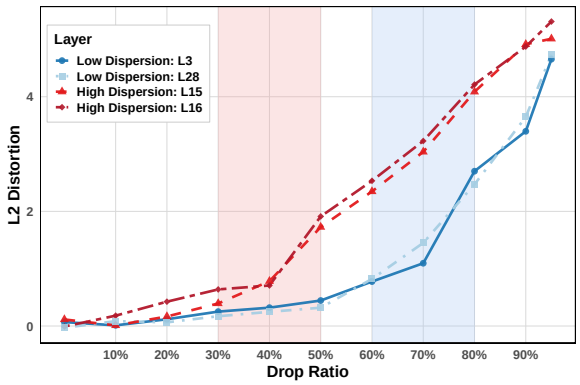


Figure 4: Output distortion under random token removal across layers with different dispersion levels. High-dispersion layers exhibit earlier and more severe degradation as tokens are removed, while low-dispersion layers remain stable over a wider range of drop ratios. The color band denotes the region of rapid degradation.

Since tokens are deleted uniformly at random, this difference in sensitivity reflects intrinsic layer-wise redundancy rather than token-specific importance. These results show that the Dispersion Score predicts a layer's tolerance to token-level KV cache compression.

3 Theoretical Justification of Dispersion as a Redundancy Measure

In this section, we provide a theoretical justification for the connection between dispersion and layer-wise redundancy. We show that, under mild and empirically supported conditions on Transformer representations, the Dispersion Score $D_l(d)$ induces a monotone ordering of layers by their effective information capacity within the dominant representation subspace.

3.1 Setup

Given C_l defined in Eq (1) with eigenvalues $\lambda_1^{(l)} \geq \cdots \geq \lambda_D^{(l)} \geq 0$. We focus on the top-$d$ principal subspace and define:

$$\Sigma_l \triangleq \mathrm{diag}(\lambda_1^{(l)}, \ldots, \lambda_d^{(l)}), \quad D_l(d) \triangleq \mathrm{Tr}(\Sigma_l). \tag{3}$$

To characterize the spectral shape of Σ_l, we use the *effective rank*:

$$r_l \triangleq \frac{\mathrm{Tr}(\Sigma_l)^2}{\mathrm{Tr}(\Sigma_l^2)} \in (0, d], \tag{4}$$

which measures how evenly variance is distributed across the top-d directions and can be directly computed from PCA eigenvalues.

3.2 Dispersion and Information Capacity

To connect dispersion with information content, we consider the log-determinant functional:

$$\mathcal{C}_l(\sigma^2) \triangleq \frac{1}{2}\log\det\Big(I + \frac{1}{\sigma^2}\Sigma_l\Big), \tag{5}$$

which corresponds to the information capacity of a Gaussian channel with covariance Σ_l and fixed noise level σ^2. This functional is monotone in the eigenvalues of Σ_l.

Upper envelope. By the AM-GM inequality, for any Σ_l with $\mathrm{Tr}(\Sigma_l) = D_l(d)$,

$$\mathcal{C}_l(\sigma^2) \le \frac{d}{2}\log\Big(1 + \frac{D_l(d)}{d\sigma^2}\Big). \tag{6}$$

Lower envelope. Conversely, among spectra with fixed trace $D_l(d)$ and effective rank r_l, the capacity is minimized when variance is evenly distributed across r_l directions, yielding

$$\mathcal{C}_l(\sigma^2) \ge \frac{r_l}{2}\log\Big(1 + \frac{D_l(d)}{r_l\sigma^2}\Big). \tag{7}$$

Together, Eq (6) and Eq (7) show that, for layers whose spectral shape r_l varies within a limited range—as observed empirically—the information capacity $\mathcal{C}_l(\sigma^2)$ is a monotone function of the dispersion $D_l(d)$ up to a controlled, explicitly quantifiable deviation.

3.3 Ordering Layers by Redundancy

These bounds imply a robust ordering principle, for two layers l_1 and l_2, if:

$$D_{l_1}(d) < D_{l_2}(d) \quad \text{and} \quad r_{l_1} \approx r_{l_2}, \tag{8}$$

then $\mathcal{C}_{l_1}(\sigma^2) < \mathcal{C}_{l_2}(\sigma^2)$. In other words, larger dispersion corresponds to higher information capacity in the dominant subspace.

Since redundancy is inversely related to capacity, layers with smaller $D_l(d)$ encode information in a more compact and redundant form, while high-dispersion layers carry more non-redundant information. Importantly, this conclusion relies on relative monotonicity rather than exact constants, making it robust to moderate spectral variation across layers.

4 Dispersion-Guided Sparse KV Cache Compression

Building on the empirical and theoretical results above, we present DISCO, a dispersion-guided KV cache compression framework. As shown in Figure 5, DISCO comprises (i) an offline Dispersion-Guided Budget Allocation (DGBA) module that assigns layer-wise KV capacities from intrinsic dispersion, and (ii) an online Dispersion-Preserving Core-Set Selection (DPCS) module that retains KV tokens according to their contribution to information capacity.

4.1 Dispersion-Guided Budget Allocation

Unlike prior methods that rely on attention signals, we adopt a distributional dispersion perspective. Since a layer's dispersion reflects its effective information capacity, we allocate KV cache budgets non-uniformly across layers rather than distributing them evenly.

Layer-wise dispersion estimation. Since layer-wise dispersion is an intrinsic property of a model, it can be estimated offline prior to inference. We compute the Dispersion Score for each layer using a small, task-agnostic corpus of M samples drawn from The Pile (Gao et al., 2020) and Proof-Pile-2 (Azerbayev et al., 2024). We estimate dispersion separately for keys and values, yielding $\widehat{\mathcal{D}}_l^K$ and $\widehat{\mathcal{D}}_l^V$, since they play distinct roles in attention (retrieval versus content).

$$\widehat{\mathcal{D}}_l^s := \frac{1}{M}\sum_{m=1}^{M} D_l^{s,(m)}(d), \qquad s \in \{K, V\}. \tag{9}$$

We then combine them into a single layer score for budgeting via a multiplicative fusion:

$$\widehat{\mathcal{D}}_l := \big(\widehat{\mathcal{D}}_l^K\big)^{1/\tau_K}\big(\widehat{\mathcal{D}}_l^V\big)^{1/\tau_V}, \tag{10}$$

where τ_K and τ_V control the relative sensitivity to key and value dispersion.

Budget Allocation Given the offline estimated dispersion profile $\widehat{\mathcal{D}}$, for each layer l, based on these scores, a fixed total KV cache budget C_{total} is distributed across layers according to:

$$C_l \;=\; C_{\text{total}} \cdot \frac{\widehat{\mathcal{D}}_l}{\sum_{i=1}^{L}\widehat{\mathcal{D}}_i}. \tag{11}$$

The resulting budget C_l constrains the total number of KV entries retained for layer l during inference. Although KV tensors are stored per attention head, the layer-wise budget is enforced jointly across all heads within the same layer: eviction is applied to a shared set of tokens, and the retained

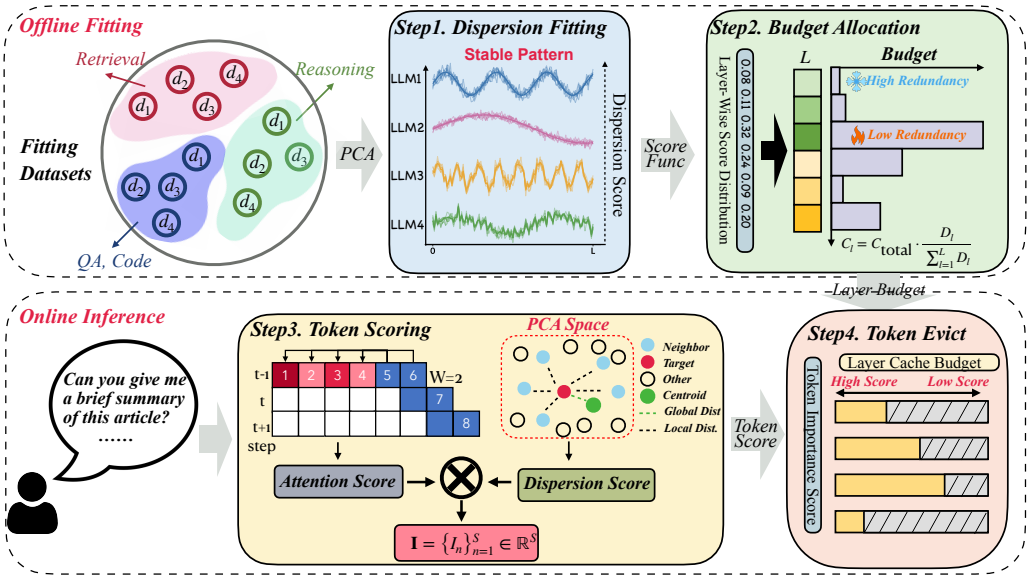


Figure 5: Overview of the DISCO framework. The offline stage computes layer-wise dispersion scores and PCA subspaces used for budget allocation and online DPCS. The online stage performs dispersion-preserving core-set selection to retain KV tokens that contribute most to each layer's information capacity.

tokens are kept consistently across heads. This design aligns cache allocation with layer-level redundancy while remaining fully compatible with multi-head attention architectures.

4.2 Dispersion Preserving Core-Set Selection

Although we estimate the information capacity of each layer and allocate an appropriate budget accordingly, effective token selection further requires identifying and removing the tokens whose eviction incurs the smallest reduction in a layer's information content.

For layer l, DPCS views the cached KV entries as a geometric point set in the dispersion defined subspace. Specifically, each cached token is projected onto the pretrained PCA subspace obtained in the offline stage, yielding a set of vectors:

$$\{p_i\}_{i=1}^{N} \subset \mathbb{R}^d. \tag{12}$$

Since this subspace captures the dominant variation of the layer's representations, operating in it preserves the essential geometry while keeping computation inexpensive. Under this view, token eviction becomes a core-set selection problem: given a fixed budget, the goal is to retain a subset of tokens whose geometric configuration best preserves the dispersion of the full set.

Centroid Based Token Dispersion Loss. To efficiently characterize the geometric structure of cached tokens within a layer, DPCS represents the projected token set using a centroid-based formulation. Given the projected token vectors $\{p_i\}_{i=1}^{N}$, we compute their centroid $\mu = \frac{1}{N}\sum_{j=1}^{N} p_j$ and the associated total variance: $V = \sum_{j=1}^{N} \|p_j - \mu\|_2^2$. The centroid is the unique point that minimizes the sum of squared distances, and the total variance decomposes as the sum of per-token contributions.

When a token p_i is removed, both the centroid and the total variance change. Using a standard variance decomposition identity (Mardia et al., 2024), the resulting reduction in dispersion admits a closed-form expression:

$$\Delta_l(i) = V - V' = \frac{N}{N-1}\|p_i - \mu\|_2^2, \tag{13}$$

where V' denotes the variance after removing p_i.

Eq. 13 shows that evicting token i incurs a dispersion loss proportional to $\|p_i - \mu\|_2^2$: tokens near the centroid are geometrically redundant, while those farther away capture distinctive directions. We use $\Delta_l(i)$ as the token-level dispersion score and rank tokens accordingly for dispersion-preserving core-set selection.

Augmenting Dispersion with Attention. Dispersion captures geometric redundancy but is independent of the current decoding context, so we augment it with an attention-derived relevance signal computed online. Let $A_l \in \mathbb{R}^{T\times N}$ denote the head-aggregated attention matrix at layer l, where $A_l[\tau, i]$ is the attention weight assigned to cached

token i at decoding step τ. Given a window size S_w, we define the attention score of token i as

$$\mathcal{A}_l(i) \;=\; \mathrm{Mean}\big(A_l[-S_w:,\, i]\big), \qquad (14)$$

where $[-S_w:]$ denotes the most recent S_w decoding steps.

Token eviction at layer l retains the C_l KV entries with the largest importance scores. We define

$$\mathcal{I}_l(i) = \mathcal{A}_l(i)\Delta_l(i), \quad D_l = \mathrm{TopK}(\mathcal{I}_l, C_l), \quad (15)$$

and prune the cache accordingly:

$$\hat{K}_l = K_l[D_l, :], \quad \hat{V}_l = V_l[D_l, :]. \qquad (16)$$

5 Experiments

5.1 Experimental Settings

Models. We evaluate DISCO on LLaMA3.1-8B-Instruct, Mistral-7B-v0.3-Instruct, and Qwen2.5-14B-Instruct, which span diverse architectures and training regimes. Their differing attention and normalization designs yield distinct layer-wise dispersion patterns, providing a rigorous test of DISCO's generality. We also include LLaMA3-OpenBioLLM-8B to assess performance in a biomedical domain setting.

Datasets. We evaluate long-context understanding on LongBench (Bai et al., 2024), which covers diverse tasks with input lengths up to tens of thousands of tokens. Retrieval accuracy under extreme sparsification is assessed using NeedleBench (Li et al., 2024b), a controlled benchmark for exact-token recall. Computational efficiency is measured on standard inference workloads with varying context lengths. To examine domain robustness, we additionally evaluate biomedical question answering and reasoning tasks. Detailed dataset descriptions are provided in Appendix C.

Baseline Methods. We compare DISCO with representative KV cache compression methods. H2O (Zhang et al., 2023) is a token-level eviction method that retains heavy-hitter tokens based on attention contributions. CAKE (Qin et al., 2025) serves as an adaptive KV management method that dynamically adjusts cache retention using model-internal signals. SnapKV (Li et al., 2024c) is a structured compression method. In addition, THINK (Xu et al., 2025) and UNComp (Xiong et al., 2025) are channel or head-level sparsity methods that introduce fine-grained sparsity across attention heads or channels based on query relevance or uncertainty estimation.

5.2 Main Results

Performance on Long-Context Understanding. As shown in Table 1, DISCO consistently achieves the strongest performance across all evaluated models and cache budgets. Notably, its relative gains increase as the memory budget becomes tighter: under the 256L setting, DISCO improves over the best baseline by 1.5–4.0%, while maintaining stable gains of 0.6–3.0% under 1024L. This pattern indicates that dispersion-guided allocation effectively protects layers most sensitive to aggressive KV reduction. The improvements generalize across models and task types, suggesting that dispersion captures an architecture-independent property of Transformer representations. These results align with the information-theoretic analysis in Section 3: by minimizing dispersion loss, DISCO preserves layer-wise information capacity, which is particularly critical for tasks requiring multi-hop reasoning or evidence aggregation, where small losses in representational diversity can disproportionately degrade performance. Additional results are reported in Appendix D.

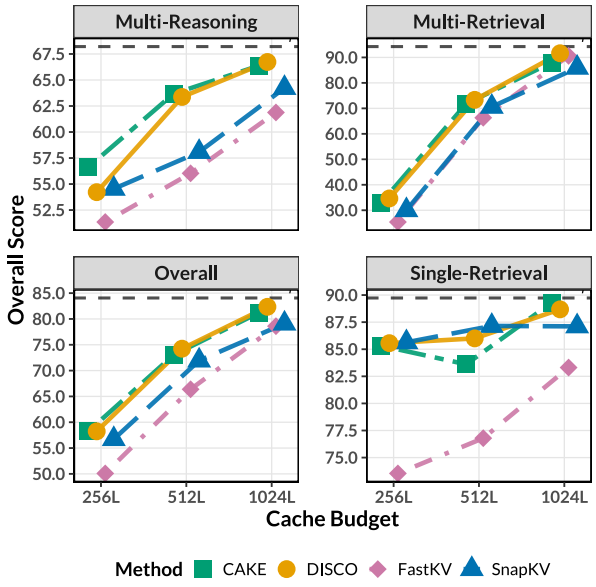


Figure 6: Performance comparison on the NeedleBench benchmark using the LLaMA3.1-8B-Instruct model with various cache budgets. The performance of the Full model (without cache eviction), representing the upper bound, is shown by the black dashed line.

Performance on Retrieval Capability. As shown in Figure 6, while all methods degrade under tighter cache budgets, DISCO demonstrates superior resilience. It matches top baselines in Single-Retrieval tasks and excels in Multi-Retrieval, achieving 34.63 at 256L—significantly

Model	Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
		NrtvQA	Qasper	MF	HotQA	2Wiki	Musi	GovR	QMSum	MTNs	TREC	Trivia	SAMS	PCnt	PR	Lcc	RB-P	
LLaMA3.1-8B (256L)	H2O	25.87	27.32	43.91	52.15	43.08	28.35	22.18	22.44	22.81	47.43	91.19	42.03	6.25	98.95	61.09	**52.11**	39.82
	THINK	29.69	34.29	51.71	54.60	42.46	30.29	**25.20**	23.51	23.50	**71.00**	**92.12**	41.38	**8.38**	**99.50**	59.69	50.99	46.14
	UNComp	29.89	35.43	51.99	53.38	**46.03**	30.36	24.42	23.60	23.29	67.00	91.95	41.51	6.78	**99.50**	**61.23**	50.41	46.06
	SnapKV	29.66	36.33	**52.63**	54.44	43.38	28.73	25.04	23.65	**23.55**	69.50	91.96	41.37	7.88	**99.50**	60.20	50.70	46.16
	CAKE	29.70	38.38	51.40	54.16	45.69	29.87	23.83	**23.78**	23.43	57.00	91.12	41.33	5.95	**99.50**	61.10	51.69	45.50
	DISCO	**31.18**	**40.08**	51.59	**54.82**	45.01	**31.10**	24.68	23.53	23.42	70.50	90.91	**42.75**	6.17	**99.50**	60.05	49.41	**46.54**
LLaMA3.1-8B (1024L)	H2O	29.36	35.87	46.20	54.64	44.93	29.12	27.41	23.54	**26.47**	62.22	92.15	**42.78**	6.40	99.00	62.66	55.10	46.12
	THINK	30.83	44.33	52.54	55.36	45.39	**31.96**	28.70	24.67	26.10	68.5	91.48	42.40	6.50	**99.50**	61.71	55.00	47.81
	UNComp	30.22	44.01	52.04	54.76	**46.76**	30.59	**29.43**	24.66	26.19	70.50	91.76	42.42	**7.03**	**99.50**	62.67	52.74	47.83
	SnapKV	30.85	44.23	52.58	55.09	45.21	31.50	28.70	24.78	26.17	70.50	91.73	42.52	6.50	**99.50**	**63.00**	55.20	48.00
	CAKE	**30.88**	**44.69**	52.16	**55.43**	46.49	30.82	28.62	**24.94**	26.43	69.50	**91.94**	42.26	5.75	**99.50**	62.31	55.96	47.98
	DISCO	31.61	44.56	**52.76**	55.03	46.56	31.74	29.00	24.41	26.42	**71.00**	91.81	42.75	6.00	**99.50**	62.80	**56.37**	**48.27**
Mistral-7B (256L)	H2O	21.25	26.66	35.13	38.82	29.80	18.88	21.00	19.50	18.63	41.00	87.64	38.25	1.50	67.00	44.35	47.29	34.79
	THINK	24.46	31.89	46.92	46.88	30.97	22.55	23.12	22.84	22.46	68.50	89.48	44.10	**5.50**	94.50	**57.31**	56.74	43.01
	UNComp	24.93	29.71	47.89	47.37	31.12	**22.68**	24.43	22.87	22.68	71.00	89.17	43.64	**5.50**	95.00	56.91	55.12	43.13
	SnapKV	24.27	29.36	47.10	47.09	30.66	22.59	24.89	**23.66**	22.11	**72.50**	88.46	**45.14**	**5.50**	94.50	56.76	56.43	43.19
	CAKE	**25.42**	32.53	46.78	47.56	**32.60**	22.04	24.20	23.59	22.68	57.50	88.31	44.19	5.00	96.00	57.16	**57.91**	42.72
	DISCO	24.54	**33.48**	**48.02**	**47.63**	31.45	22.58	**25.06**	23.64	**22.97**	71.00	**89.71**	44.66	5.00	**96.50**	56.67	57.39	**43.77**
Mistral-7B (1024L)	H2O	25.92	33.81	46.57	47.98	32.65	25.24	27.89	23.73	24.81	68.36	87.92	43.15	2.28	71.34	51.82	53.95	41.71
	THINK	26.41	36.45	48.13	50.35	**34.94**	24.98	27.67	24.14	25.42	73.50	**89.11**	46.98	**6.50**	96.50	59.06	59.87	45.62
	UNComp	25.26	35.96	49.02	48.61	34.34	24.89	28.73	**24.85**	25.87	73.50	88.64	47.06	5.00	94.50	59.05	60.10	45.40
	SnapKV	25.74	35.61	48.13	48.90	33.31	24.88	29.23	24.83	**26.32**	73.00	88.99	46.06	5.00	96.50	58.78	60.07	45.33
	CAKE	26.69	35.65	49.11	49.81	34.55	25.02	28.97	24.43	26.11	72.00	**89.11**	46.40	4.50	**97.00**	**59.47**	**60.39**	45.58
	DISCO	**27.62**	**37.11**	**49.64**	**50.82**	34.47	**25.35**	**29.44**	24.56	26.02	**74.00**	88.82	**47.28**	**6.50**	96.50	59.39	59.89	**46.09**
Qwen2.5-14B (256L)	H2O	24.03	29.48	42.10	52.53	52.38	35.84	21.00	19.67	16.27	68.00	83.84	44.41	9.22	94.58	55.85	37.35	42.91
	THINK	25.41	30.48	42.51	57.92	53.68	36.31	21.03	21.25	17.63	70.55	90.48	46.18	9.42	97.38	**57.23**	44.32	45.11
	UNComp	25.36	30.59	42.45	57.85	55.72	36.27	20.99	21.21	17.67	70.50	90.54	**46.24**	9.39	97.42	57.19	44.29	45.23
	SnapKV	24.30	32.21	43.79	58.49	56.21	37.43	**21.21**	21.41	17.81	72.00	**90.69**	45.89	10.13	**98.67**	56.35	44.73	45.71
	CAKE	24.13	33.36	43.72	57.03	55.88	35.92	20.58	**21.81**	17.85	71.00	87.13	45.20	8.97	98.58	56.68	**44.87**	45.17
	DISCO	**25.63**	**35.02**	**45.84**	**58.64**	**57.87**	**37.44**	21.15	21.17	**17.86**	**74.00**	89.59	45.53	**10.46**	97.25	56.51	44.62	**46.16**
Qwen2.5-14B (1024L)	H2O	26.24	39.33	44.74	55.81	55.04	33.34	24.60	20.77	20.55	64.78	87.91	44.26	8.35	94.88	57.80	37.79	44.76
	THINK	26.73	41.52	47.18	**58.91**	57.42	36.68	25.38	22.19	20.81	72.47	**90.89**	47.51	9.51	98.39	60.74	**48.31**	47.79
	UNComp	26.69	41.48	47.22	58.84	57.38	37.61	25.41	22.16	**20.83**	72.50	90.82	47.47	9.48	98.42	60.70	48.26	47.83
	SnapKV	26.22	42.21	47.54	58.90	57.24	**37.82**	25.11	21.65	20.07	72.50	90.79	47.40	10.30	**98.67**	60.35	47.23	47.75
	CAKE	25.97	42.05	**47.89**	58.20	56.41	37.06	25.05	**22.86**	20.68	71.50	90.56	46.22	9.81	98.60	**61.66**	48.15	47.67
	DISCO	**26.89**	**42.49**	47.76	58.86	**57.99**	37.60	**25.69**	21.87	20.73	**74.00**	90.66	**47.86**	**10.67**	98.25	59.90	48.07	**48.08**

Table 1: Performance comparison on 16 LongBench datasets. Best results are in **bold**.

outperforming CAKE (32.73), SnapKV (30.02), and FastKV (25.38). DISCO also maintains competitive performance in Multi-Reasoning across all budgets. Overall, DISCO achieves the highest scores at every budget level (58.24, 74.24, and 82.35 for 256L, 512L, and 1024L, respectively), confirming its robustness and balanced performance across diverse long-context tasks. Detailed extended results are provided in Appendix E.

Performance on System Efficiency. We evaluate the 1024L cache budget setting. As shown in Table 2, DISCO achieves the highest system efficiency under both 8k and 64k inputs. The gains arise not merely from reducing KV cache size, but from where and how KV entries are pruned. By removing tokens from low-dispersion layers and geometrically redundant regions, DISCO reshapes the retained KV distribution, leading to more discriminative attention and fewer ineffective memory accesses. Moreover, DGBA prevents KV overallocation in information-critical layers, alleviating layer-wise computational hotspots. Together, these effects improve memory locality and reduce redundant attention computation, enabling FlashAttention to operate closer to its optimal regime in long-context inference.

Performance on Medical Domain Task. We evaluate DISCO on MedConceptsQA (Shoham and Rappoport, 2024), a domain-specific benchmark pairing long medical references with clinical questions. Its sparse, high-precision terminology and reliance on long-range factual consistency make it sensitive to attention-based KV compression that over-prunes low-attention yet critical con-

Method	Throughput (tokens/s)	Latency (ms/token)	Memory (GB)
8k Context, Compressibility = 1024/8k = 12.5%			
Full w. FlashAttn-2	7.70	0.0480	18.25
THINK	15.85	0.0430	17.08
UNComp	13.73	0.0413	17.05
SnapKV	13.34	0.0408	17.32
CAKE	13.66	0.0449	17.32
DISCO	**17.68**	**0.0394**	**17.01**
64k Context, Compressibility = 1024/64k = 1.56%			
Full w. FlashAttn-2	7.70	0.1581	32.69
THINK	16.46	0.0437	23.96
UNComp	17.10	0.0422	23.89
SnapKV	13.99	0.0410	24.43
CAKE	12.47	0.0449	24.43
DISCO	**17.28**	**0.0397**	**23.87**

Table 2: System efficiency comparison of LLaMA3.1-8B-Instruct using FlashAttention on NVIDIA A100.

cepts. As shown in Figure 7, DISCO consistently outperforms SnapKV and CAKE and narrows the gap to the full-cache baseline across tasks, with the largest gains on harder ICD coding tasks that require aggregating low-frequency concepts over long documents, indicating better preservation of low-redundancy domain facts.

5.3 Ablation Study

Component Effectiveness. Table 3 evaluates the two core components of DISCO on LLaMA-3.1-8B-Instruct under a 256L cache budget. Under fixed token scoring, dispersion-guided budget allocation (DGBA) yields the best performance (46.54 Avg.), outperforming random, uniform, and depth-based allocation, indicating that dispersion provides a more effective signal for distributing KV capacity across layers. Under fixed layer-wise allocation, token-level scoring further shows that neither attention-only nor dispersion-only selection is sufficient; combining attention relevance with dispersion loss (full DPCS) achieves the strongest results across task groups. Together, these ablations confirm that DISCO requires both layer-wise dispersion allocation and dispersion-preserving token selection, and that enforcing a consistent dispersion-based criterion at both levels is critical to its effectiveness.

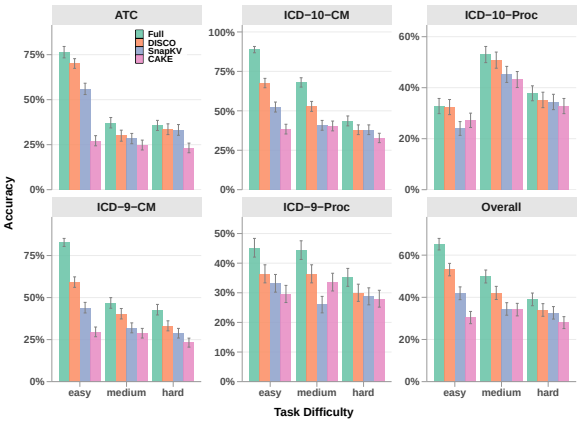


Figure 7: Performance comparison on medical question-answering tasks across different KV cache compression methods.

Setting	Single-QA	Multi-QA	Summ.	Code	Avg.
Layer-wise Budget Allocation (Token Scoring Fixed)					
Random	30.12	27.74	22.01	50.63	45.39
Uniform	30.45	28.02	22.30	51.04	45.52
Pyramid	31.10	28.66	22.75	51.90	45.94
DGBA (Ours)	**31.88**	**29.41**	**23.12**	**53.02**	**46.54**
Token-level Selection (Layer-wise Allocation Fixed)					
Attention-only	30.92	28.78	22.74	51.83	46.23
Dispersion-only	30.51	28.10	22.63	51.18	45.94
Full DPCS (Ours)	**31.18**	**29.41**	**23.12**	**53.02**	**46.54**

Table 3: Top: effect of layer-wise budget allocation with fixed token scoring. Bottom: effect of token-level scoring with fixed layer-wise allocation.

Fusion Strategy and PCA dimension Sensitivity. Table 4 compares fusion strategies for combining key and value dispersion. Multiplicative fusion performs best, improving the average score by 0.5–0.7 points over additive and max fusion, as it requires both retrieval (key) and content (value) signals to be strong rather than allowing either to dominate.

Fusion / d	Single-QA	Multi-QA	Summ.	Code	Avg.
Additive	30.74	28.35	22.41	51.32	46.01
Max	30.21	27.94	22.30	51.10	45.82
Multiplicative	**31.18**	**29.41**	**23.12**	**53.02**	**46.54**
2	30.71	28.92	22.51	51.84	46.02
4	**31.18**	**29.41**	**23.12**	**53.02**	**46.54**
8	31.09	29.33	23.05	52.81	46.46
16	30.96	29.10	22.88	52.54	46.29

Table 4: Effect of fusion strategy and hyperparameters on LLaMA-3.1-8B-Instruct under a 256L cache budget. Multiplicative fusion yields the strongest and most stable performance across task categories.

6 Conclusion

We presented DISCO, which leverages dispersion patterns to guide both layer-wise budget allocation and token eviction in KV cache compression. By recognizing that low-dispersion layers contain more redundancy while high-dispersion tokens carry critical information, DISCO achieves state-of-the-art performance across benchmarks with minimal overhead. Our information-theoretic approach provides a principled framework for KV cache management, demonstrating significant improvements especially in domains requiring high-fidelity information retention.

Limitations

DISCO relies on an offline dispersion estimation to derive layer-wise budgets; while empirically stable across inputs and datasets, models with substantially different architectures or training regimes may require re-estimation. The method allocates budgets at the layer level with shared pruning across heads, which does not explicitly exploit head-level heterogeneity. Token selection uses PCA-based projections and centroid statistics; although lightweight, this adds minor computation compared to purely attention-based heuristics. Finally, our evaluation focuses on autoregressive text generation, and extensions to other settings (e.g., encoder-only or multimodal models) remain future work.

References

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1(1):4.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*.

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)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Amit Ben-Artzy and Roy Schwartz. 2024. Attend first, consolidate later: On the importance of attention in different llm layers. *arXiv preprint arXiv:2409.03621*.

Sébastien Bubeck, Christian Coester, Ronen Eldan, Timothy Gowers, Yin Tat Lee, Alexandru Lupsasca, Mehtaab Sawhney, Robert Scherrer, Mark Sellke, Brian K Spears, and 1 others. 2025. Early science acceleration experiments with gpt-5. *arXiv preprint arXiv:2511.16072*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, and 1 others. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. 2025. When attention sink emerges in language models: An empirical view. In *The Thirteenth International Conference on Learning Representations*.

Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. 2025. OmniKV: Dynamic context selection for efficient long-context LLMs. In *The Thirteenth International Conference on Learning Representations*.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, and 1 others. 2024. Exploring concept depth: How large language models acquire knowledge and concept at different layers? *arXiv preprint arXiv:2404.07066*.

Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim. 2025. Fastkv: Kv cache compression for fast long-context processing with token-selective propagation. *arXiv preprint arXiv:2502.01068*.

Bingli Li and Danilo Vasconcellos Vargas. 2024. Extending token computation for llm reasoning. In *Proceedings of the 2024 8th International Conference on Computer Science and Artificial Intelligence*, pages 367–373.

Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024a. Pre-trained language models for text generation: A survey. *ACM Computing Surveys*, 56(9):1–39.

Mo Li, Songyang Zhang, Yunxin Liu, and Kai Chen. 2024b. Needlebench: Can llms do retrieval and reasoning in 1 million context window? *arXiv preprint arXiv:2407.11963*.

Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. 2021. The heterogeneity hypothesis: Finding layer-wise differentiated network architectures. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2144–2153.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024c. SnapKV:

LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Bingzhe Liu, Gangmuk Lim, Ryan Beckett, and P. Brighten Godfrey. 2024. Kivi: Verification for cluster management. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 509–527, Santa Clara, CA. USENIX Association.

Kanti V Mardia, John T Kent, and Charles C Taylor. 2024. *Multivariate analysis*. John Wiley & Sons.

Jaime Pizarroso, José Portela, and Antonio Muñoz. 2022. Neuralsens: sensitivity analysis of neural networks. *Journal of Statistical Software*, 102:1–36.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. CAKE: Cascading and adaptive KV cache eviction with layer preferences. In *The Thirteenth International Conference on Learning Representations*.

Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. 2024. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*.

Ofir Ben Shoham and Nadav Rappoport. 2024. Medconceptsqa: Open source medical concepts qa benchmark. *Computers in Biology and Medicine*, 182:109089.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*.

Chihiro Taguchi, Seiji Maekawa, and Nikita Bhutani. 2025. Efficient context selection for long-context QA: No tuning, no iteration, just adaptive-k. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20116–20141, Suzhou, China. Association for Computational Linguistics.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Jing Xiong, Jianghan Shen, Fanghua Ye, Chaofan Tao, Zhongwei Wan, Jianqiao Lu, Xun Wu, Chuanyang Zheng, Zhijiang Guo, Min Yang, Lingpeng Kong, and Ngai Wong. 2025. UNComp: Can matrix entropy uncover sparsity? — a compressor design from an uncertainty-aware perspective. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4179–4199, Suzhou, China. Association for Computational Linguistics.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2025. Think: Thinner key cache by query-driven pruning. In *International Conference on Representation Learning*, volume 2025, pages 56691–56709.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. In *MLSys*.

A Related Work

The substantial memory footprint of the Key-Value (KV) cache is a primary bottleneck for deploying Large Language Models (LLMs) in long-context scenarios. This has catalyzed a diverse range of research efforts to optimize KV cache management, which can be broadly classified into three main categories: cache eviction and compression, KV cache quantization, and architectural modifications.

Cache Eviction and Compression strategies focus on reducing the number of tokens stored in the cache. Early and simple methods are recency-based; for example, StreamingLLM (Xiao et al., 2024) uses a sliding window that retains the most recent tokens while discarding the oldest. To preserve important long-term context, more advanced techniques use importance-based eviction. H2O (Zhang et al., 2023), for instance, evicts tokens with the lowest cumulative attention scores. Beyond token-level eviction, recent work explores finer-grained sparsification along channel or head dimensions. THINK (Xu et al., 2025) leverages query-dependent attention structure to dynamically prune key channels that contribute little to the current query, while UNComp (Xiong et al., 2025) models uncertainty via truncated matrix entropy to reveal structured sparsity and compress hidden states and KV caches adaptively across layers and attention heads. Other methods aim for a more structured compression. PyramidKV (Cai et al., 2024) retains recent tokens densely while sparsely

sampling older ones, and SnapKV (Li et al., 2024c) saves compressed "snapshots" of different context segments. These methods aim to intelligently discard or summarize less critical information to stay within a fixed memory budget.

Cache Quantization offers a complementary approach that reduces memory by decreasing the numerical precision of the stored keys and values, rather than removing them. This typically involves converting the 16-bit floating-point (FP16/BF16) cache values into low-bit integer formats, such as INT8 or INT4. A primary obstacle in this process is the presence of extreme outlier values in the activation and cache tensors, which can cause significant performance degradation if handled with naive quantization schemes (Dettmers et al., 2022). Consequently, research has focused on developing more sophisticated methods that are robust to these outliers. Techniques like SmoothQuant (Xiao et al., 2023) mitigate this issue by mathematically smoothing the activation distributions to make them more quantization-friendly. Other approaches, such as those explored in LLM.int8() (Dettmers et al., 2022) and Atom (Zhao et al., 2024), utilize mixed precision or non-uniform quantization to represent outliers with higher precision while aggressively quantizing the remaining values. More recently, methods developed specifically for the KV cache, like KIVI (Liu et al., 2024), have proposed dedicated data structures and fine-grained, per-token quantization strategies to further minimize information loss. These methods aim to achieve significant memory reduction (e.g., 2-4x) with minimal impact on the model's generative quality, making them a crucial component for efficient LLM inference.

B Generalization of Dispersion Patterns Across Different LLMs

This appendix provides comprehensive experimental validation demonstrating that the dispersion-based redundancy patterns observed in LLaMA-3.1-8B-Instruct (Section 2) constitute a universal phenomenon across diverse transformer architectures. Through systematic analysis of layer-wise dispersion scores across models varying in size, architectural design, and training paradigms, we establish that these patterns are not model-specific artifacts but rather fundamental characteristics of how transformer-based language models process information. To ensure robust validation, we conduct our experiments on a diverse set of long-context tasks, utilizing 20 randomly sampled examples from each of the 16 datasets in the LongBench benchmark, totaling 320 evaluation samples per model. Our findings reveal remarkably consistent dispersion profiles and redundancy correlations across multiple model families, suggesting that the computational inefficiencies identified in our primary analysis represent inherent properties of the transformer architecture itself, independent of specific implementation details or training procedures.

B.1 Dispersion Patterns in LLaMA-3.1-8B-Instruct

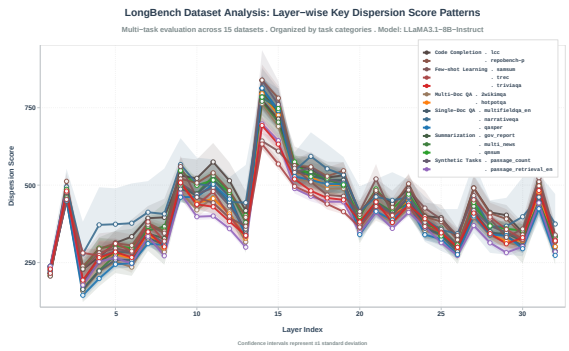
This section characterizes the layer-wise dispersion profiles of LLaMA-3.1-8B-Instruct. Using the Dispersion Score defined in Section 2, we measure how broadly key and value representations are distributed across layers under long-context inputs. The goal is to examine whether dispersion exhibits a structured and stable pattern within a single model and to identify layers that are intrinsically more redundant or information-intensive. We report dispersion for keys and values separately, reflecting their distinct roles in attention, and visualize the resulting layer-wise trends in Figure 8.

B.2 Dispersion Patterns in Mistral-7B-v0.3-Instruct

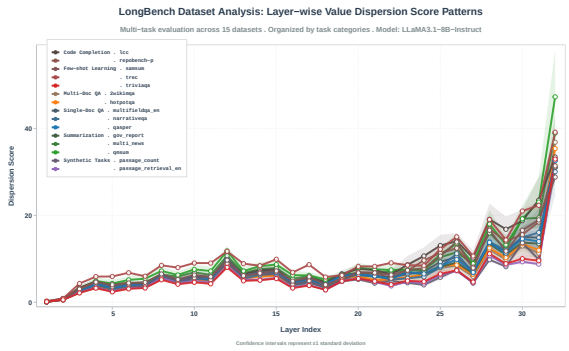
We next analyze the layer-wise dispersion profiles of Mistral-7B-v0.3-Instruct using the same protocol as in Section 2. By computing Dispersion Scores for key and value representations across layers under long-context inputs, we examine whether the structured dispersion patterns observed in LLaMA also emerge in a model with different architecture and attention design. This comparison helps isolate which aspects of dispersion reflect model-intrinsic redundancy rather than idiosyncrasies of a particular model family. The resulting key and value dispersion trends are shown in Figure 9.

B.3 Dispersion Patterns in Qwen2.5-14B-Instruct

We further study the layer-wise dispersion profiles of Qwen2.5-14B-Instruct using the same measurement procedure as in the previous subsections. By computing Dispersion Scores for key and value representations across layers under long-context

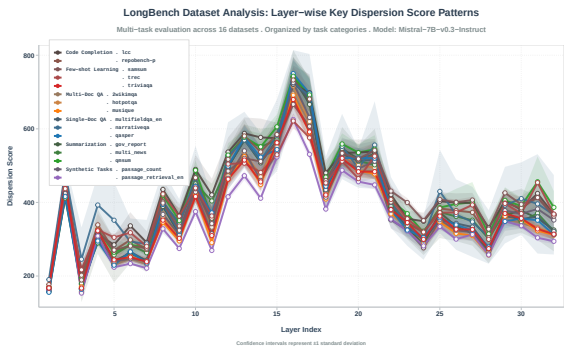


(a)

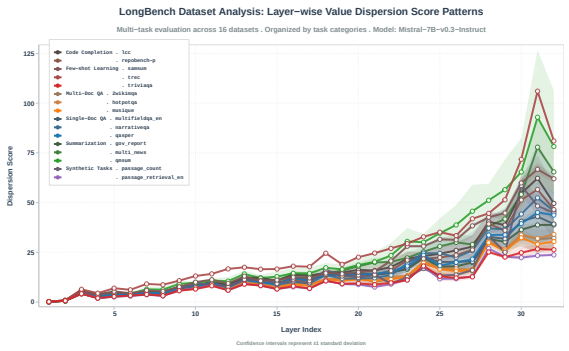


(b)

Figure 8: Layer-wise dispersion analysis of key and value vectors in LLaMA-3.1-8B-Instruct evaluated on LongBench datasets. (a) Key vectors dispersion analysis. The solid line depicts the mean cosine similarity across layers, while the shaded region represents the standard deviation confidence intervals. (b) Value vectors dispersion analysis.



(a)



(b)

Figure 9: Layer-wise dispersion analysis of key and value vectors in Mistral-7B-v0.3-Instruct evaluated on LongBench datasets. (a) Key vectors dispersion analysis. The solid line depicts the mean cosine similarity across layers, while the shaded region represents the standard deviation confidence intervals. (b) Value vectors dispersion analysis.

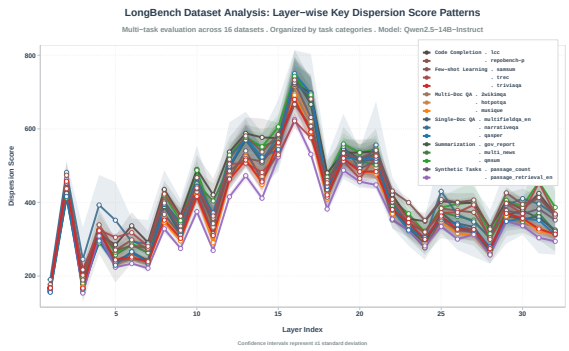
inputs, we test whether the dispersion structure remains consistent in a larger, multilingual model with different training and architectural characteristics. This analysis provides additional evidence on the generality of dispersion as a model-intrinsic redundancy signal beyond a single model family or scale. The resulting key and value dispersion trends are reported in Figure 10.

C Experimental Details

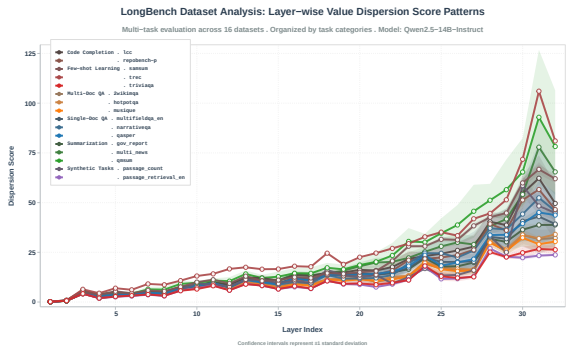
This appendix presents detailed specifications of our experimental configuration, encompassing the experimental settings, model architectures, baseline comparison methods, and evaluation benchmarks utilized throughout our study, thereby providing comprehensive supplementary information to Section 5.1 of the main paper.

C.1 Detailed Experimental Settings

Our experiments were conducted on a high-performance computing cluster equipped with dual Intel Xeon Platinum 8358 processors (64 cores total per node), 8×NVIDIA A100 80GB GPUs, and 1TB DDR4 memory per node, utilizing PyTorch 2.7.1 and CUDA 12.2. The compute nodes feature a dual-socket NUMA architecture with 96MB L3 cache and support for advanced instruction sets including AVX-512. To ensure reproducibility, we implemented comprehensive random seed control (seed=42) across all random number generators, including Python's built-in random functions, NumPy, PyTorch, and CUDA random generators, with PyTorch's deterministic mode enabled to guarantee consistent results across runs. Inference was performed with temperature=1.0, allowing for natural sampling variability while maintaining experimental consistency. All experiments were repeated three times with independent random initializations, and results are reported as averaged values across these runs to ensure statistical reliability. We employed distributed computing with automatic workload balancing across available GPUs and im-



(a)



(b)

Figure 10: Layer-wise dispersion analysis of key and value vectors in Qwen2.5-14B-Instruct evaluated on LongBench datasets. (a) Key vectors dispersion analysis. The solid line depicts the mean cosine similarity across layers, while the shaded region represents the standard deviation confidence intervals. (b) Value vectors dispersion analysis.

plemented rigorous quality control measures, including data integrity checks and comprehensive error handling mechanisms throughout the evaluation pipeline.

C.2 Model Details

We evaluate the proposed method across a range of widely used open-source large language models, encompassing various model architectures and parameter sizes. This diverse selection ensures a thorough assessment of both effectiveness and generalizability. The full list of models is summarized in Table 5.

C.3 Baseline Method Details

We compare DISCO with eight state-of-the-art KV cache compression methods, each representing a distinct technical approach. A summary of these baselines is provided in Table 6.

Model	HuggingFace Repository
LLaMA-3.1-8B-Instruct	`https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`
Mistral-7B-Instruct-v0.3	`https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`
Qwen2.5-14B-Instruct	`https://huggingface.co/Qwen/Qwen2.5-14B-Instruct`
LLaMA3-OpenBioLLM-8B	`https://huggingface.co/aaditya/Llama3-OpenBioLLM-8B`

Table 5: Pretrained large language models used in the experiments.

C.4 Evaluation Benchmark Details

Our evaluation covers four dimensions to ensure comprehensiveness and fairness.

- Long-context understanding: We used the LongBench (Bai et al., 2024) benchmark to evaluate long-context understanding, which includes tasks requiring reasoning over long-range dependencies. In our experiments, we set the input text length to 16k and constrained the KV Cache budget for all compression methods to 4k, resulting in a 25% compression rate. We evaluated six core tasks: Code, QA, Summarization, Few-shot, Math, and Single-doc QA. Detailed information about the datasets used in these tasks is provided in Table 7.

- Retrieval capability: We used the NeedleBench (Li et al., 2024b) benchmark to evaluate the model's ability to retrieve specific information (the "needle") from long, distractor-filled contexts (the "haystack"). The benchmark consists of three distinct tasks: Single-Retrieval, which tests the model's ability to retrieve a single piece of information; Multi-Retrieval, which requires identifying and integrating multiple needles; and Multi-Reasoning, which involves reasoning over the retrieved information. The tasks are available in both English (EN) and Chinese (ZH) to assess cross-lingual performance.

- System efficiency: We evaluate system-level efficiency on a single NVIDIA A100 (80GB) GPU with FlashAttention. We report three metrics: (1) *peak GPU memory* (GB), (2)

Category	Method	Description	Venue & Year
Token-level Eviction	StreamingLLM (Xiao et al., 2024)	Evicts the oldest tokens to preserve recent context when the cache is full.	ICLR 2024
	H2O (Zhang et al., 2023)	Attention-based eviction using cumulative attention weight; evicts tokens with lower scores.	NeurIPS 2023
Adaptive Compression	AdaKV (Feng et al., 2024)	Dynamically adjusts the importance of layers/heads using attention entropy, and allocates budgets accordingly.	ICLR 2024
	FastKV (Jo et al., 2025)	Lightweight scoring mechanism for fast evaluation of token importance to enable dynamic eviction.	arXiv 2025
	CAKE (Qin et al., 2025)	Combines token attention and recency to assign cache sizes dynamically for each head.	ICLR 2025
	THINK (Xu et al., 2025)	Query-aware channel pruning in the key cache; dynamically removes key channels with low contribution to the current query.	ICLR 2025
	UNComp (Xiong et al., 2025)	Models uncertainty via truncated matrix entropy to reveal structured sparsity and compress hidden states and KV cache across layers and heads.	EMNLP 2025
Structured Compression	SnapKV (Li et al., 2024c)	Splits KV cache into multiple "snapshots" and retains key information from each time segment.	ACL 2024
	GemFilter (Shi et al., 2024)	Uses information entropy to identify and retain "gem" tokens contributing most to the output.	arXiv 2024
	PyramidKV (Cai et al., 2024)	Uses a pyramid-shaped retention strategy to prioritize more recent tokens and sparsely sample older ones.	NeurIPS 2024

Table 6: Summary of representative KV cache compression and eviction methods, categorized by strategy type.

decoding throughput (tokens/s), and (3) *per-token decoding latency* (ms/token). All measurements are performed under the same model and context settings (8k/512L and 64k/1024L) for fair comparison.

- Domain-specific performance validation: To assess DISCO's effectiveness in real-world specialized scenarios, we conducted experiments using the Llama3-OpenBioLLM-8B model on the MedConceptsQA dataset (Shoham and Rappoport, 2024). We specifically adapted the dataset by concatenating relevant medical reference materials with each question to generate long-context inputs, thereby creating a challenging evaluation scenario that requires effective information retention across extended medical documents.

D Detailed Experimental Results On LongBench

In this section, we present comprehensive experimental results on LongBench (Bai et al., 2024), a benchmark specifically designed to evaluate long-context understanding. We evaluate the performance of our method under varying cache budget constraints, ranging from 256L to 2048L, across three base models: LLaMA3.1-8B-Instruct (see Appendix D.1), Mistral-7B-Instruct-v0.3 (Appendix D.2), and Qwen2.5-14B-Instruct (Appendix D.3). These results reveal the robustness and adaptability of DISCO across a wide range of memory budgets and model scales.

D.1 Detailed Performance Analysis Across Cache Budgets on LLaMA3.1-8B-Instruct

We first analyze how DISCO behaves as the cache budget varies on LLaMA3.1-8B-Instruct. By sweeping the budget from 256L to 2048L, we quantify the performance–memory trade-off and examine whether DISCO degrades gracefully under tight budgets while approaching the full-cache upper

Dataset	ID	Source	Avg len	Metric	Language	#data
Single-Document QA						
NarrativeQA	1-1	Literature, Film	18,409	F1	English	200
Qasper	1-2	Science	3,619	F1	English	200
MultiFieldQA-en	1-3	Multi-field	4,559	F1	English	150
MultiFieldQA-zh	1-4	Multi-field	6,701	F1	Chinese	200
Multi-Document QA						
HotpotQA	2-1	Wikipedia	9,151	F1	English	200
2WikiMultihopQA	2-2	Wikipedia	4,887	F1	English	200
MuSiQue	2-3	Wikipedia	11,214	F1	English	200
DuReader	2-4	Baidu Search	15,768	Rouge-L	Chinese	200
Summarization						
GovReport	3-1	Government report	8,734	Rouge-L	English	200
QMSum	3-2	Meeting	10,614	Rouge-L	English	200
MultiNews	3-3	News	2,113	Rouge-L	English	200
VCSUM	3-4	Meeting	15,380	Rouge-L	Chinese	200
Few-shot Learning						
TREC	4-1	Web question	5,177	Accuracy (CLS)	English	200
TriviaQA	4-2	Wikipedia, Web	8,209	F1	English	200
SAMSum	4-3	Dialogue	6,258	Rouge-L	English	200
LSHT	4-4	News	22,337	Accuracy (CLS)	Chinese	200
Synthetic Task						
PassageCount	5-1	Wikipedia	11,141	Accuracy (EM)	English	200
PassageRetrieval-en	5-2	Wikipedia	9,289	Accuracy (EM)	English	200
PassageRetrieval-zh	5-3	C4 Dataset	6,745	Accuracy (EM)	Chinese	200
Code Completion						
LCC	6-1	Github	1,235	Edit Sim	Python/C#/Java	500
RepoBench-P	6-2	Github repository	4,206	Edit Sim	Python/Java	500

Table 7: An overview of the dataset statistics in LongBench. Chinese datasets are highlighted. "Source" denotes the origin of the context. "Avg len" (average length) is computed using the number of words for the English (code) datasets and the number of characters for the Chinese datasets. "Accuracy (CLS)" refers to classification accuracy, while 'Accuracy (EM)' refers to exact match accuracy.

bound as more capacity is available.

D.2 Detailed Performance Analysis Across Cache Budgets on Mistral-7B-Instruct-v0.3

We next repeat the cache-budget sweep on Mistral-7B-Instruct-v0.3 to examine whether DISCO exhibits similar robustness under a different architecture and attention design. This analysis highlights how performance evolves from tight to relaxed budgets and whether the same dispersion-guided strategy consistently yields stable gains across models.

D.3 Detailed Performance Analysis Across Cache Budgets on Qwen2.5-14B-Instruct

Finally, we evaluate DISCO on Qwen2.5-14B-Instruct across the same range of cache budgets to assess scalability to a larger, multilingual model. This budget sweep characterizes the performance–memory trade-off at higher model capacity and verifies that dispersion-guided compression remains effective as cache constraints vary.

E Detailed Experimental Results On NeedleBench

In this appendix, we provide more detailed experimental results on the NeedleBench benchmark. The results for LLaMA3.1-8B-Instruct are presented in Table 11. Our method demonstrates strong performance, particularly in the Multi-Retrieval task under tight budget constraints.

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	PR-en	Lcc	RB-P	
Llama3.1-8B-Instruct, $B_{\text{total}} = Full$																	
Full	30.97	45.49	53.78	54.76	47.13	31.42	34.90	25.28	27.48	73.00	91.65	43.80	6.00	99.50	63.38	56.73	49.08
Llama3.1-8B-Instruct, $B_{\text{total}} = 256L$																	
StreamingLLM	23.84	24.26	30.82	45.90	39.88	24.90	21.12	20.46	19.22	45.99	86.95	40.66	6.92	98.74	59.97	48.38	39.88
H2O	25.87	27.32	43.91	52.15	43.08	28.35	22.18	22.44	22.81	47.43	91.19	42.03	6.25	98.95	61.09	**52.11**	39.82
AdaKV	29.14	33.07	50.92	54.22	43.01	30.14	**24.77**	23.26	22.81	**71.00**	91.48	40.98	6.55	**99.50**	60.24	50.99	45.76
GemFilter	15.07	12.37	28.10	39.55	22.26	18.12	23.60	17.58	20.16	59.50	85.57	37.84	1.24	16.50	23.43	24.53	27.84
FastKV	28.74	36.12	52.40	53.26	45.77	30.01	23.84	23.29	23.60	68.00	91.00	42.20	6.38	**99.50**	62.31	49.37	45.99
SnapKV	29.66	36.33	**52.63**	54.44	43.38	28.73	25.04	23.65	23.55	69.50	91.96	41.37	7.88	**99.50**	60.20	50.70	46.16
PyramidKV	**31.48**	33.92	48.76	**57.13**	**46.89**	27.15	24.93	20.88	**25.67**	53.41	**92.85**	43.27	8.14	96.73	**62.58**	49.31	45.19
CAKE	29.70	38.38	51.40	54.16	45.69	29.87	23.83	**23.78**	23.43	57.00	91.12	41.33	5.95	**99.50**	61.10	51.69	45.50
DISCO	31.18	**40.08**	51.59	54.82	45.01	**31.10**	24.68	23.53	23.42	70.50	90.91	**42.75**	6.17	**99.50**	60.05	49.41	**46.54**
Llama3.1-8B-Instruct, $B_{\text{total}} = 512L$																	
StreamingLLM	25.46	27.58	33.25	47.46	39.79	24.63	23.26	20.67	22.79	57.31	87.39	41.97	6.50	97.94	60.77	51.56	41.77
H2O	27.59	28.89	45.12	52.78	44.10	28.93	24.48	22.96	24.62	54.36	91.16	42.40	6.36	99.00	62.36	**54.74**	44.37
AdaKV	30.16	40.11	**54.13**	54.27	44.73	**31.19**	**27.08**	23.82	24.39	64.70	91.9	41.64	6.50	**99.50**	60.54	53.15	46.74
GemFilter	18.52	22.15	37.73	46.88	32.56	24.52	27.05	17.75	23.25	60.50	89.49	40.16	3.53	52.00	27.35	31.02	34.65
FastKV	30.38	41.12	24.37	54.35	**46.69**	30.89	26.78	23.82	24.37	**72.50**	92.04	**42.64**	**7.03**	**99.50**	62.67	52.74	45.74
SnapKV	**30.76**	42.19	52.13	53.87	46.41	30.37	24.99	24.34	24.42	63.95	**92.05**	41.87	6.05	**99.50**	62.62	54.52	46.88
PyramidKV	30.34	42.15	52.17	54.45	45.51	30.60	25.00	24.40	24.72	62.94	91.89	41.85	5.92	**99.50**	61.66	53.92	46.69
CAKE	29.34	42.54	51.63	54.51	46.56	30.50	26.04	**24.73**	**25.10**	62.50	91.93	42.09	6.0	**99.50**	62.57	54.22	46.86
DISCO	30.71	**42.75**	51.48	**54.68**	45.79	30.82	25.57	24.57	24.88	65.50	91.92	41.79	6.05	**99.50**	**62.89**	53.85	**47.05**
Llama3.1-8B-Instruct, $B_{\text{total}} = 1024L$																	
StreamingLLM	26.80	31.07	35.81	47.38	42.09	24.24	25.99	21.43	25.67	63.27	88.24	**42.78**	6.50	88.49	61.41	53.29	42.78
H2O	29.36	35.87	46.20	54.64	44.93	29.12	27.41	23.54	**26.47**	62.22	**92.15**	**42.78**	6.40	99.00	62.66	55.10	46.12
AdaKV	30.87	44.29	52.58	55.33	45.42	**31.93**	28.73	24.63	26.14	68.47	91.52	42.36	6.47	**99.50**	61.74	54.96	47.80
GemFilter	19.34	34.76	46.58	46.82	42.83	27.51	29.89	18.96	25.68	63.00	90.70	42.50	6.50	**99.50**	62.09	56.28	44.56
FastKV	30.22	44.01	52.04	54.76	**46.76**	30.59	**29.43**	24.66	26.19	**70.50**	91.76	42.42	**7.03**	**99.50**	62.67	52.74	47.83
SnapKV	30.85	44.23	52.58	55.09	45.21	31.50	28.70	24.78	26.17	**70.50**	91.73	42.52	6.50	**99.50**	**63.00**	55.20	48.00
PyramidKV	30.25	43.68	52.73	54.96	46.40	31.28	27.64	24.28	26.16	68.00	92.09	42.05	6.04	**99.50**	62.37	55.54	47.69
CAKE	30.88	**44.69**	52.16	**55.43**	46.49	30.82	28.62	**24.94**	26.43	69.50	91.94	42.26	5.75	**99.50**	62.31	55.96	47.98
DISCO	**31.61**	44.56	**52.76**	55.03	46.56	31.74	29.00	24.41	26.42	71.00	91.81	42.75	6.00	**99.50**	62.80	**56.37**	**48.27**
Llama3.1-8B-Instruct, $B_{\text{total}} = 2048L$																	
StreamingLLM	27.45	36.90	37.91	49.38	44.64	24.29	28.73	21.75	27.07	67.55	90.93	42.44	6.14	86.81	62.89	55.33	44.39
H2O	29.68	39.44	48.60	54.18	46.59	29.24	30.02	23.88	27.34	67.96	91.45	43.11	5.94	99.41	63.11	**56.90**	47.30
AdaKV	30.07	45.28	**53.23**	**55.30**	45.88	30.31	30.94	24.50	26.98	**72.50**	**91.65**	43.50	6.18	**99.50**	62.94	56.15	48.43
GemFilter	23.64	41.17	51.39	53.97	45.32	29.24	**32.01**	20.50	26.91	70.00	91.59	42.59	6.88	95.50	47.35	38.91	44.81
FastKV	30.31	45.41	52.79	55.11	45.57	30.46	31.25	**24.82**	27.02	71.50	91.48	**43.66**	6.62	**99.50**	63.17	55.82	48.41
SnapKV	**30.72**	45.07	53.15	55.25	45.33	30.78	30.62	24.45	27.18	70.00	91.48	43.30	**8.21**	**99.50**	63.14	56.06	48.39
PyramidKV	30.13	45.06	52.70	54.78	46.59	**30.89**	30.25	**24.82**	**27.35**	71.00	**91.65**	42.62	6.00	**99.50**	**63.27**	56.44	48.31
CAKE	30.34	45.22	53.04	55.14	46.58	30.40	31.00	24.67	27.12	70.50	**91.65**	43.34	6.00	**99.50**	62.98	56.42	48.37
DISCO	30.60	**45.70**	52.80	54.85	**46.61**	30.71	31.29	24.73	27.23	71.50	91.48	43.21	6.00	**99.50**	**63.27**	56.01	**48.47**

Table 8: Performance comparison over 16 datasets of LongBench on LLaMA-3.1-8B-Instruct. The best result is highlighted in **bold**.

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	PR-en	Lcc	RB-P	
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = Full$																	
Full	28.53	37.99	49.42	51.39	36.18	27.61	34.28	25.36	26.36	76.00	88.59	47.51	5.00	97.00	59.35	60.70	46.95
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 256L$																	
StreamingLLM	18.10	23.41	25.99	26.78	16.25	17.97	18.82	16.56	52.50	84.26	39.42	1.00	**16.50**	44.76	44.99	50.50	31.11
H2O	21.25	26.66	35.13	38.82	29.80	18.88	21.00	19.50	18.63	41.00	87.64	38.25	1.50	67.00	44.35	47.29	34.79
AdaKV	24.71	31.54	47.28	46.63	31.22	22.31	23.41	**24.02**	**23.18**	70.92	88.74	**45.56**	5.34	93.88	55.93	55.06	43.05
GemFilter	8.89	7.64	19.41	26.41	13.56	9.95	23.57	18.29	19.19	48.00	78.84	35.21	2.85	7.25	29.25	32.57	23.80
FastKV	25.15	29.38	**48.12**	47.05	31.34	**22.41**	24.71	23.14	22.95	71.42	88.76	43.21	5.86	94.55	**57.18**	54.74	43.12
SnapKV	24.27	29.36	47.10	47.09	30.66	22.59	24.89	23.66	22.11	**72.50**	88.46	45.14	5.50	94.50	56.76	56.43	43.19
PyramidKV	21.73	26.60	41.46	43.20	29.32	21.47	20.23	19.82	17.46	40.00	87.64	37.11	1.05	69.00	42.55	42.98	35.14
CAKE	**25.42**	32.53	46.78	47.56	**32.60**	22.04	24.20	23.59	22.68	57.50	88.31	44.19	5.00	96.00	57.16	**57.91**	42.72
DISCO	24.54	**33.48**	48.02	**47.63**	31.45	22.58	**25.06**	23.64	22.97	71.00	**89.71**	44.66	5.00	**96.50**	56.67	57.39	**43.77**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 512L$																	
StreamingLLM	19.93	25.83	28.25	36.21	27.60	15.41	21.23	18.67	19.50	64.00	86.45	39.75	1.00	74.50	46.70	47.57	35.79
H2O	23.78	31.63	41.31	43.24	31.07	20.43	26.74	20.41	23.93	67.50	88.84	42.62	1.50	72.00	50.60	49.87	39.72
AdaKV	26.08	35.13	47.61	49.45	33.80	23.85	25.68	**24.49**	24.17	**72.50**	89.14	45.91	5.00	96.00	58.06	57.44	44.64
GemFilter	8.27	13.39	24.46	35.29	20.41	10.13	26.64	19.07	22.80	55.50	84.18	39.11	2.63	29.08	34.66	36.69	28.89
FastKV	25.47	33.53	**49.43**	47.68	34.73	**25.14**	26.42	23.82	24.01	72.00	89.27	46.39	4.50	95.00	**58.74**	58.66	44.67
SnapKV	25.24	33.83	48.72	49.59	32.77	23.88	**27.02**	24.30	**24.52**	72.00	89.49	45.89	5.00	96.00	58.03	**58.91**	44.70
PyramidKV	24.07	33.23	46.74	44.00	31.99	23.20	24.41	21.16	21.28	68.06	88.39	40.25	1.43	87.56	48.64	49.21	40.18
CAKE	**26.16**	33.54	47.76	49.49	**35.37**	23.94	26.08	24.12	24.42	67.50	**89.50**	45.96	5.50	**96.50**	58.21	58.75	44.55
DISCO	25.62	**35.20**	48.51	**50.35**	33.86	23.25	26.64	24.30	24.37	**72.50**	88.86	**46.41**	**6.00**	94.50	58.15	56.80	**44.71**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 1024L$																	
StreamingLLM	21.58	27.23	30.87	36.41	28.94	16.75	23.18	19.92	21.86	67.83	88.29	40.14	1.98	83.31	47.62	49.45	37.84
H2O	25.92	33.81	46.57	47.98	32.65	25.24	27.89	23.73	24.81	68.36	87.92	43.15	2.28	71.34	51.82	53.95	41.71
AdaKV	26.78	36.02	48.55	**50.82**	**35.31**	24.63	28.05	24.48	25.86	74.02	88.67	**47.41**	**6.92**	**97.08**	59.44	60.23	44.52
GemFilter	10.20	19.87	31.39	40.97	26.65	15.23	29.36	19.39	24.90	63.00	87.14	39.51	3.51	67.64	39.21	36.82	34.67
FastKV	25.62	36.41	49.48	48.95	34.81	25.21	29.10	24.47	**26.32**	**74.02**	88.12	47.58	5.42	95.03	59.44	**60.56**	45.66
SnapKV	25.74	35.61	48.13	48.90	33.31	24.88	29.23	**24.83**	**26.32**	73.00	88.99	46.06	5.00	96.50	58.78	60.07	45.33
PyramidKV	25.56	36.08	47.75	44.78	33.17	23.99	26.61	21.84	23.34	70.60	88.46	41.00	1.02	87.09	50.21	50.96	42.03
CAKE	26.69	35.65	49.11	49.81	34.55	25.02	28.97	24.43	26.11	72.00	**89.11**	46.40	4.50	97.00	**59.47**	60.39	45.58
DISCO	**27.62**	**37.11**	**49.64**	**50.82**	34.47	**25.35**	**29.44**	24.56	26.02	74.00	88.82	47.28	6.50	96.50	59.39	59.89	**46.09**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 2048L$																	
StreamingLLM	22.18	28.27	38.00	42.64	29.81	17.38	24.09	20.73	22.50	70.15	91.17	41.63	2.06	82.26	49.38	51.22	39.59
H2O	26.89	35.22	48.91	49.79	33.44	**25.80**	29.56	24.29	25.52	71.74	**92.22**	44.27	2.34	75.46	53.91	56.59	43.50
AdaKV	26.48	37.13	48.61	50.45	33.80	24.85	25.68	24.49	24.17	74.50	89.14	45.91	5.00	**96.00**	58.06	59.44	45.23
GemFilter	15.53	30.43	39.43	42.93	33.06	21.84	**32.11**	21.19	26.36	67.50	87.68	36.20	3.71	90.06	44.94	41.21	39.64
FastKV	26.03	37.44	49.23	50.11	34.20	25.61	31.03	**25.18**	**26.54**	74.50	88.69	47.47	**5.50**	95.50	59.41	60.12	46.04
SnapKV	25.83	37.70	49.09	50.67	34.81	25.20	30.34	25.08	26.26	**75.50**	88.86	47.45	5.00	96.00	59.90	60.14	46.11
PyramidKV	26.08	**37.74**	48.34	45.83	**37.20**	24.04	28.95	22.15	24.40	72.59	88.37	42.33	1.13	84.53	50.09	51.16	42.81
CAKE	**28.29**	36.85	49.11	50.46	35.59	25.40	30.82	24.31	26.47	75.00	88.64	47.21	4.50	95.50	59.44	**60.84**	46.15
DISCO	28.05	36.58	**49.24**	**50.71**	34.59	25.47	30.78	24.74	26.12	75.00	88.64	**47.89**	5.00	95.50	59.62	60.64	**46.16**

Table 9: Performance comparison over 16 datasets of LongBench on Mistral-7B-Instruct-v0.3. The best result is highlighted in **bold**.

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	PR-en	Lcc	RB-P	
Qwen2.5-14B-Instruct, $B_{\text{total}} = Full$																	
Full	28.91	45.78	53.22	62.55	58.19	37.05	16.83	18.05	14.11	76.00	90.69	38.93	9.92	98.67	61.39	49.88	47.51
Qwen2.5-14B-Instruct, $B_{\text{total}} = 256L$																	
StreamingLLM	21.15	24.80	36.77	50.14	48.62	33.20	20.91	18.44	17.63	70.14	84.77	42.19	9.21	92.65	54.82	36.02	41.34
H2O	24.03	29.48	42.10	52.53	52.38	35.84	21.00	19.67	16.27	68.00	83.84	44.41	9.22	94.58	55.85	37.35	42.91
AdaKV	25.21	30.38	42.36	57.82	53.53	36.21	20.93	21.15	17.53	70.45	90.38	46.08	9.37	97.28	**57.18**	44.22	45.01
GemFilter	18.89	27.64	39.41	46.41	43.56	29.95	20.57	18.29	16.19	48.00	78.84	35.21	8.85	78.25	49.25	32.57	36.99
FastKV	25.18	30.42	42.31	57.73	55.48	36.12	20.83	21.06	17.52	70.34	90.37	**46.09**	9.28	97.21	56.98	44.14	45.13
SnapKV	24.30	32.21	43.79	58.49	56.21	37.43	**21.21**	21.41	17.81	72.00	**90.69**	45.89	10.13	**98.67**	56.35	44.73	45.71
PyramidKV	24.33	32.18	43.82	58.52	56.19	37.41	21.23	21.39	17.83	72.03	90.71	45.91	10.11	98.69	56.37	44.71	45.71
CAKE	24.13	33.36	43.72	57.03	55.88	35.92	20.58	**21.81**	17.85	71.00	87.13	45.20	8.97	98.58	56.68	**44.87**	45.17
DISCO	**25.63**	**35.02**	**45.84**	**58.64**	**57.87**	**37.44**	21.15	21.17	**17.86**	**74.00**	89.59	45.53	**10.46**	97.25	56.51	44.62	**46.16**
Qwen2.5-14B-Instruct, $B_{\text{total}} = 512L$																	
StreamingLLM	22.38	25.94	37.89	51.67	49.85	34.52	21.78	19.21	18.49	71.43	86.12	43.64	10.08	94.17	56.29	37.45	42.56
H2O	25.27	30.81	43.54	54.02	53.76	37.19	22.13	20.95	17.64	69.38	85.27	45.73	10.57	95.92	57.21	38.68	44.25
AdaKV	24.78	**37.92**	44.23	57.89	55.34	35.41	24.12	20.67	18.95	74.83	88.72	48.21	**10.65**	97.28	**60.73**	43.87	46.48
GemFilter	20.15	28.97	40.83	47.94	45.02	31.28	21.44	19.56	17.47	49.72	80.31	36.65	9.73	79.68	50.81	33.92	38.34
FastKV	25.45	36.87	45.66	56.56	56.61	36.67	23.38	21.83	19.67	73.50	89.99	46.88	9.30	98.50	59.41	44.65	46.56
SnapKV	25.96	36.47	45.58	56.72	56.61	36.73	22.84	21.89	**20.31**	73.45	90.14	47.05	9.18	96.67	59.26	45.12	46.50
PyramidKV	24.53	37.81	44.92	58.07	55.28	35.96	**24.21**	20.44	19.67	**74.92**	88.76	**48.39**	10.42	97.35	60.58	43.74	46.56
CAKE	**26.88**	37.88	**47.29**	**59.25**	56.60	**37.53**	22.86	**22.18**	19.88	61.50	90.25	46.64	7.60	**98.75**	59.82	**47.69**	46.41
DISCO	25.97	36.87	45.95	58.88	**56.93**	36.22	22.78	21.62	19.50	74.50	**90.27**	47.47	8.49	97.00	58.16	45.14	**46.61**
Qwen2.5-14B-Instruct, $B_{\text{total}} = 1024L$																	
StreamingLLM	23.74	30.12	39.26	53.04	51.19	35.87	22.95	20.48	19.76	72.81	87.55	45.09	11.33	95.62	57.74	38.82	44.09
H2O	26.51	32.14	44.97	55.48	55.12	38.54	23.26	22.22	18.91	70.76	86.70	47.18	**11.84**	97.25	58.57	40.01	45.59
AdaKV	25.31	41.52	46.21	**59.06**	54.73	36.55	22.69	20.04	**22.18**	73.52	**91.99**	46.37	10.31	97.24	59.56	45.83	47.16
GemFilter	21.42	30.23	42.17	49.41	46.37	32.63	22.71	20.83	18.74	51.05	81.78	38.12	10.98	81.15	52.28	35.27	39.70
FastKV	25.69	41.38	46.12	57.74	56.28	35.51	23.31	22.06	20.73	**75.80**	90.22	47.37	8.48	98.32	60.60	**48.42**	47.38
SnapKV	26.22	42.21	47.54	58.90	57.24	**37.82**	25.11	21.65	20.07	72.50	90.79	47.40	10.30	**98.67**	60.35	47.23	47.75
PyramidKV	24.37	**42.75**	47.68	56.49	55.12	36.94	22.18	21.03	22.17	74.85	91.67	46.21	9.74	97.18	59.43	47.28	47.19
CAKE	25.97	42.05	**47.89**	58.20	56.41	37.06	25.05	**22.86**	20.68	71.50	90.56	46.22	9.81	98.60	**61.66**	48.15	47.67
DISCO	**26.89**	42.49	47.76	58.86	**57.99**	37.60	**25.69**	21.87	20.73	74.00	90.66	**47.86**	10.67	98.25	59.90	48.07	**48.08**
Qwen2.5-14B-Instruct, $B_{\text{total}} = 2048L$																	
StreamingLLM	25.08	31.45	40.61	54.51	52.56	37.22	24.22	21.75	21.03	74.18	88.92	46.56	9.60	96.99	59.11	40.19	45.25
H2O	27.86	33.47	46.34	56.95	56.49	**37.89**	24.53	**23.49**	20.18	72.13	88.07	**48.65**	9.11	**98.62**	59.94	41.38	46.57
AdaKV	27.73	42.14	**50.78**	62.68	57.31	37.18	23.26	20.61	21.79	74.15	92.59	46.97	**10.82**	97.84	60.19	46.43	48.28
GemFilter	22.75	31.56	43.54	50.88	47.74	33.98	23.98	22.10	20.01	52.42	83.25	39.59	8.95	82.52	53.65	36.64	40.85
FastKV	26.41	43.52	49.23	61.34	55.86	38.01	22.13	19.38	**22.14**	**75.67**	91.18	48.35	10.07	98.52	58.94	47.89	48.04
SnapKV	27.84	42.19	50.68	**62.71**	57.52	36.73	26.47	22.65	21.39	74.23	**92.74**	47.12	9.34	97.18	60.37	47.54	48.54
PyramidKV	28.14	44.08	49.05	61.86	57.49	37.29	**27.45**	22.30	22.44	72.22	91.04	47.68	9.01	98.55	**60.80**	47.65	48.56
CAKE	**28.81**	43.43	50.39	62.06	57.42	36.93	27.05	23.32	21.72	72.00	90.75	46.83	9.10	98.46	60.71	**48.22**	48.58
DISCO	28.05	**44.11**	50.31	61.46	**57.76**	36.55	26.45	22.66	21.26	73.50	91.84	48.14	9.05	98.33	60.20	47.93	**48.60**

Table 10: Performance comparison over 16 datasets of LongBench on Qwen2.5-14B-Instruct. The best result is highlighted in **bold**.

Method	Single-Retrieval			Multi-Retrieval			Multi-Reasoning			Overall
	ZH	EN	Avg	ZH	EN	Avg	ZH	EN	Avg	
Llama3.1-8B Instruct, $B_{\text{total}} = Full$										
Full	100.00	79.46	89.73	96.65	91.84	94.25	69.22	67.20	68.21	84.06
Llama3.1-8B Instruct, $B_{\text{total}} = 1024L$										
FastKV	99.51	67.11	83.31	90.77	90.42	90.60	62.86	60.90	61.88	78.60
SnapKV	97.91	76.31	87.11	87.51	84.34	85.93	65.75	62.70	**64.23**	79.09
CAKE	**100.00**	**78.58**	**89.29**	86.67	89.24	87.96	**67.95**	64.78	66.37	81.21
DISCO	**100.00**	77.36	88.68	**92.65**	**90.60**	**91.63**	67.29	**66.16**	66.73	**82.35**
Llama3.1-8B Instruct, $B_{\text{total}} = 512L$										
FastKV	88.32	65.26	76.79	59.28	73.33	66.31	51.53	60.52	56.03	66.38
SnapKV	98.00	**76.30**	**87.15**	60.29	80.90	70.60	58.71	57.44	58.08	71.94
CAKE	98.00	69.21	83.61	57.43	**86.15**	71.79	**65.20**	62.14	**63.67**	73.02
DISCO	**99.00**	73.00	86.00	**65.44**	81.23	**73.34**	63.80	**62.93**	63.37	**74.24**
Llama3.1-8B Instruct, $B_{\text{total}} = 256L$										
FastKV	86.28	60.79	73.54	23.90	26.85	25.38	49.83	52.86	51.35	50.09
SnapKV	96.42	**74.85**	**85.64**	31.40	28.64	30.02	56.88	52.19	54.54	56.73
CAKE	**98.00**	72.65	85.33	32.66	32.80	32.73	**58.72**	**54.29**	**56.61**	58.22
DISCO	**98.00**	73.15	85.56	**36.10**	**33.15**	**34.63**	55.28	53.15	54.22	**58.24**

Table 11: Performance of Llama3.1-8B-Instruct on three subtasks of the NeedleBench 32K benchmark. Best results are shown in **bold**.