One Size Does Not Fit All: Token-Wise Adaptive Compression for KV Cache

Anonymous ACL submission

Abstract

Despite the remarkable progress of Large Language Models (LLMs), the escalating memory footprint of the Key-Value (KV) cache remains a critical bottleneck for efficient inference. While dimensionality reduction offers a promising compression avenue, existing approaches typically either necessitate prohibitively expensive pre-training from scratch or suffer from severe performance deterioration under high compression regimes. In this work, we propose DynaKV, a novel post-training framework for low-rank KV cache compression. To the best of our knowledge, DynaKV is the first low-rank method to dynamically allocate compression rates to individual tokens according to their semantic meaning, which allows it to achieve better fidelity at aggressive compression ratios. Extensive experiments demonstrate that our method consistently outperforms existing state-of-the-art compression techniques, achieving significant memory reduction while maintaining competitive generation quality. Furthermore, our approach is orthogonal to sequence-level pruning methods. When integrated with SnapKV, DynaKV retains only 6% of the KV cache while maintaining 94% of the baseline performance on the LongBench benchmark.[1]

1 Introduction

While Large Language Models (LLMs) (Brown et al., 2020) have revolutionized AI, the escalating memory overhead of the Key-Value (KV) cache has emerged as a critical bottleneck as model sizes and context lengths grow. Specifically, the memory footprint of cached states grows linearly with sequence length, rapidly exhausting device memory. This excessive overhead imposes a major challenge on scalability, effectively preventing the deployment of larger-scale models and severely limiting the generation of extended contexts.

[1]We will release our code for reproducibility later.

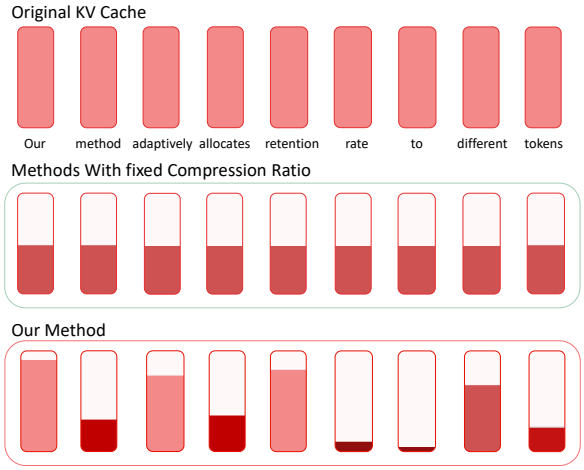

Figure 1: Illustration of the compression strategies. The diagram compares fixed-ratio compression with our proposed dynamic approach. While traditional methods apply a uniform compression rate to all tokens, DynaKV allocates variable storage budgets, assigning different retain rate of KV to different tokens based on their importance.

To mitigate the growing memory footprint of the KV cache, existing methods mainly explore two orthogonal compression dimensions: sequence-level token eviction and feature-level low-rank compression. Token-eviction methods reduce the cached sequence length by selectively retaining a subset of tokens and discarding the rest (Zhang et al., 2023; Xiao et al., 2023; Li et al., 2024; Cai et al., 2024). These methods are often training-free and can effectively reduce prompt-side memory usage, yet their benefits may be limited in long-generation scenarios where newly generated tokens continue to expand the KV cache during decoding. Low-rank compression methods operate along the vector dimension by representing each token's KV states in a compact subspace (Yuan et al., 2023; Chang et al., 2025; Wang et al., 2025; Liu et al., 2024). Since the per-token storage cost is reduced, this line of methods can continuously control cache

growth throughout decoding. As these two lines operate on different dimensions of the KV cache, they are naturally orthogonal and can be combined to achieve more aggressive memory reduction. Our work follows the low-rank compression direction.

Existing low-rank methods face a dilemma between performance and training overhead. Architectural solutions like MLA require training the model from scratch on massive datasets, making them prohibitively expensive to apply to existing pre-trained LLMs. Although methods like MHA2MLA (Ji et al., 2025) allow converting existing MHA models to MLA, they still necessitate Billions of training tokens to make it work. Alternatively, training-free and Post-training compression methods, such as Palu (Chang et al., 2025) and MatryoshkaKV (Lin et al., 2024), Despite their advantage of low adaptation costs, they inherently trade off performance for efficiency. Consequently, the primary challenge for these training-free and post-training approaches lies in sustaining performance at high compression ratios.

A critical observation motivating our work is that natural language exhibits highly non-uniform information density. In language model predictions, the majority of tokens are generated with high confidence (low entropy), indicating substantial computational redundancy while making these predictions. In addition, works like CoM (Song et al., 2025) have shown that different tokens need a variable amount of computation according to their importance. Despite this intrinsic variability, existing low-rank compression paradigms typically enforce a rigid, uniform compression policy across the entire sequence. This “one-size-fits-all" strategy is inherently suboptimal: it allocates equal memory budgets to critical and trivial tokens alike, inevitably wasting resources on the latter or degrading the fidelity of the former. Theoretically, an adaptive strategy that dynamically aligns memory allocation with local semantic importance holds the promise of achieving significantly higher compression rate without compromising the model’s core representational fidelity.

To materialize this potential, we propose **DynaKV**, which, to the best of our knowledge, is the first post-training framework to enable **token-adaptive** compression. Unlike prior works, our method empowers the model to **automatically allocate compression rates across different tokens**. By effectively leveraging this capability, DynaKV successfully bridges the gaps in current research, offering three distinct advantages:

- **Seamless Adaptation to Pre-trained Models:** As a post-training method, DynaKV can be directly applied to existing LLM architectures without structural modifications.
- **Minimal Training Overhead:** Our approach requires only lightweight fine-tuning to calibrate the dynamic gating mechanism (e.g., only 128M tokens for an 8B model), avoiding the prohibitive costs of training from scratch.
- **High Compression with Robust Performance:** By intelligently preserving critical information, DynaKV maintains competitive generation quality even at aggressive compression ratios where uniform methods typically fail.

2 Related Work

Training-free Compression Methods This category of approaches aims to compress KV pairs without updating model parameters, typically by exploiting the inherent low-rank properties of weight matrices or activations. Eigen Attention (Saxena et al., 2024) observes that the standard attention mechanism can be approximated by maintaining a truncated basis of the key-value subspace, effectively projecting KV pairs into a lower-dimensional manifold during inference. Similarly, PALU (Chang et al., 2025) decomposes the linear projection matrices into low-rank components via Singular Value Decomposition (SVD), reconstructing the full cache on-the-fly to reduce memory complexity to the rank size. Addressing inter-layer redundancy, CommonKV (Wang et al., 2025) identifies that attention features often converge across layers. It employs SVD to extract a shared basis for adjacent layers and stores only the layer-specific differences, significantly reducing redundancy with a training-free parameter sharing mechanism.

Post-training Compression Methods Post-training methods introduce lightweight optimization or calibration to adapt the model for higher compression rates. ASVD (Yuan et al., 2023) improves upon standard decomposition by introducing activation-aware scaling, which weighs the singular values by the magnitude of input activations to preserve the accuracy of outlier features during compression. MatryoshkaKV

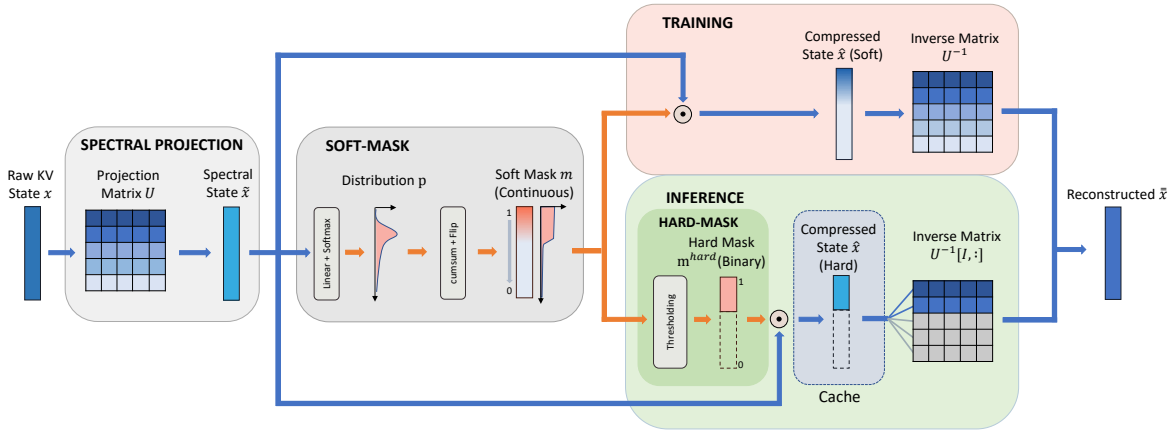

Figure 2: **Overview of the DynaKV framework.** Unlike static methods that use a uniform compression rate, DynaKV employs a token-adaptive masking mechanism to dynamically select and retain critical KV dimensions. This ensures that semantically significant context is preserved while redundancy is minimized, maintaining high performance across both short and long-context tasks.

(Lin et al., 2024) enforces a nested structure within the representation, ensuring that the most critical information is concentrated in the initial dimensions. This allows for elastic truncation, where the cache size can be dynamically adjusted by slicing the embedding vector with minimal performance degradation. Additionally, ReCalKV (Yan et al., 2025) critiques prior methods for neglecting the distinct roles of Keys and Values. It introduces a tailored post-training strategy that applies Head-wise Similarity–aware Reordering (HSR) to Keys for improved grouped SVD, while employing Offline Value Calibration (OVC) to refine Value projection matrices using calibration data to ensure accurate contextual representation.

Architecture-Intrinsic Methods These methods fundamentally alter the attention architecture, necessitating pre-training from scratch or computationally expensive retraining. GQA (Grouped Query Attention) (Ainslie et al., 2023) mitigates the memory bottleneck by grouping multiple query heads to share a single key-value head, offering a balanced trade-off between the quality of Multi-Head Attention (MHA) (Vaswani et al., 2017) and the efficiency of Multi-Query Attention (MQA) (Shazeer, 2019). MLA (Multi-Head Latent Attention) (Liu et al., 2024), introduced in DeepSeek-V2, natively projects KV pairs into a low-dimensional latent space and decouples positional embeddings, maximizing compression capabilities while maintaining model capacity. While MHA2MLA (Ji et al., 2025) attempts to bridge the gap by converting existing MHA models into the MLA format, it relies on matrix decomposition followed by extensive training (e.g., billions of tokens) to recover performance, making the computational cost comparable to partial pre-training.

3 Dynamic Compressing

In this section, we present **DynaKV**, a novel framework for token-adaptive KV cache compression. Our approach is founded on the observation that the information distribution within the KV representations is highly non-uniform and varies significantly across different tokens. The proposed method consists of three key components: (1) a basis transformation via Principal Component Analysis (PCA) to decouple feature correlations, (2) a learnable, differentiable gating mechanism that dynamically determines the optimal retention rank for each token, and (3) a compression-incentivizing training objective.

3.1 KV Projection to Spectral Space

We aim to project $\mathbf{x}$ into a compact spectral space by transforming its coordinate axes, such that the dimensions are strictly ordered by their importance. In this spectral space, the trailing dimensions exhibit minimal significance, allowing us to preferentially prune them during the subsequent compression phase with negligible information loss.

Let $\mathbf{x} \in \mathbb{R}^{d_{kv}}$ denote the pre-RoPE Key and Value states for the t-th token. Formally, we intro-

duce a learnable projection matrix $\mathbf{U} \in \mathbb{R}^{d_{kv} \times d_{kv}}$ to transform the original state:

$$\tilde{\mathbf{x}} = \mathbf{x}\mathbf{U}. \tag{1}$$

In this transformed space, $\mathbf{U}$ is optimized to concentrate the semantic energy of the vector into the leading dimensions, providing a natural inductive bias that ensures the tail dimensions are safe to prune. To establish this property from the outset, we initialize $\mathbf{U}$ as the eigen matrix of PCA derived from offline calibration[2], and subsequently allow it to be updated during training to adaptively align with the compression objective.

3.2 Differentiable Token-Adaptive Compression

To achieve token-adaptive compression, we introduce a learnable gating mechanism that dynamically determines the information retention level for each token. In this section, we first detail how this mechanism is applied during inference to realize physical storage reduction and state reconstruction. Subsequently, we describe the differentiable training process that enables the model to learn these optimal truncation points end-to-end.

Hard Masking at Inference Time During the inference phase, our goal is to minimize KV cache usage by physically discarding redundant spectral components. A key property of our transformed space is that dimensions are strictly ordered by importance, allowing us to prune dimensions starting from the tail end. Accordingly, for each token in each layer, we utilize a binary truncation mask $\mathbf{m}^{hard} \in \mathbb{R}^{d_{kv}}$, which is a vector starting with a contiguous block of $\mathbf{1}$s followed by $\mathbf{0}$s. The specific generation mechanism for $\mathbf{m}^{hard}$ is detailed in the subsequent section. Formally, the truncation is applied to the spectral state via the Hadamard product:

$$\hat{\mathbf{x}} = \tilde{\mathbf{x}} \odot \mathbf{m}^{hard}. \tag{2}$$

In terms of physical storage, we denote the set of retained indices as $\mathcal{I} = \{j \mid m_j^{hard} = 1\}$. Only the components of $\hat{\mathbf{x}}$ indexed by $\mathcal{I}$ are explicitly persisted in the KV cache, thereby realizing the actual memory reduction.

[2]Specifically, we construct a calibration dataset by collecting sufficient large number of consecutive tokens from standard text sources such as Wikipedia. We perform PCA on the covariance matrix of these states and use the resulting eigenvectors (sorted by descending eigenvalues) to initialize $\mathbf{U}$.

When reconstructing the stored KV, since different tokens retain varying numbers of components, this is implemented as a dynamic low-rank reconstruction:

$$\bar{\hat{\mathbf{x}}} = \hat{\mathbf{x}}[\mathcal{I}]\mathbf{U}^{-1}[\mathcal{I}, :]. \tag{3}$$

Here, $\bar{\hat{\mathbf{x}}}$ represents the reconstructed Key or Value vector. Following reconstruction, these states are reshaped into the multi-head format, and Rotary Positional Embeddings (RoPE) are applied to the Key vectors.

Retain Rate To quantify and control the global compression level, we define the *Retain Rate* $\mathcal{R}$ as the average proportion of retained dimensions. Let m_i^{hard} denote the i-th entry of vector $\mathbf{m}^{hard}$. The *Retain Rate* is defined as:

$$\mathcal{R} = \frac{1}{d_{kv}} \sum m_i^{hard}. \tag{4}$$

It is important to note that $\mathcal{R}$ here is specific to a single token within a particular layer. To calculate the model's overall retention rate, we average R across the entire token sequence, all layers, and all batches.

Differentiable Masking at Training Time To generate the binary truncation mask $\mathbf{m}^{hard}$, we optimize a continuous soft-mask $\mathbf{m}$ that serves as a learnable proxy for the discrete truncation operation.

Drawing inspiration from the latent structure modeling in PRPN (Shen et al., 2017) and the ordered representation in ON-LSTM (Shen et al., 2018), we implement a differentiable gating mechanism using a smooth relaxation. For a transformed spectral state $\tilde{\mathbf{x}}$, we first employ a lightweight linear layer to project the features into a probability distribution over possible truncation indices. The trainable parameters $\mathbf{W}$ and $\mathbf{b}$ produce logits which are normalized via Softmax:

$$\mathbf{p} = \text{Softmax}(\tilde{\mathbf{x}}\mathbf{W} + \mathbf{b}). \tag{5}$$

Here, the vector $\mathbf{p}$ represents the probability distribution of the optimal cutoff point across all dimensions. Crucially, we utilize separate sets of projection parameters for Keys and Values to accommodate their distinct redundancy patterns.

To construct a mask that starts at $\mathbf{1}$ (retained) and transitions to $\mathbf{0}$ (discarded), we leverage the cumulative sum operation. Intuitively, if the network predicts a high cutoff probability at index k, the cumulative sum will transition from 0 to 1 at

that index. We apply a reversal operation to invert this sequence:

$$\mathbf{m} = \text{Flip}\left(\text{cumsum}(\mathbf{p})\right). \tag{6}$$

The resulting mask $\mathbf{m}$ is approximately 1 for indices $i < k$ and 0 for $i > k$, providing a smooth gradient flow for backpropagation. This generated soft-mask then serves as a **differentiable gate**, filtering the spectral features via element-wise multiplication:

$$\hat{\mathbf{x}} = \tilde{\mathbf{x}} \odot \mathbf{m}. \tag{7}$$

During training, we retain the continuous values of $\mathbf{m}$ to maintain gradient flow, allowing the model to learn the optimal trade-off between memory usage and performance.

During inference, we discretize this continuous gate to achieve physical compression. Specifically, we obtain the binary mask $\mathbf{m}^{hard}$ by applying a predefined threshold τ (e.g., $\tau = 0.1$) to the learned soft mask:

$$m_i^{hard} = \mathbb{I}(m_i > \tau), \tag{8}$$

where $\mathbb{I}(\cdot)$ is the indicator function. This binary mask $\mathbf{m}^{hard}$ determines the actual truncation point, ensuring that only the informative dimensions are preserved in the KV cache while the rest are discarded.

3.3 Training Objective

To balance the trade-off between model performance and compression efficiency, we propose a composite objective that incorporates both the standard language modeling loss and a regularization term based on the *Retain Rate* $\mathcal{R}$ (as defined in Section 3.2).

Let $\mathcal{L}_{CE}$ denote the standard cross-entropy loss over the target tokens. The final loss function $\mathcal{L}$ is formulated as:

$$\mathcal{L} = \mathcal{L}_{CE} + \alpha \cdot \mathcal{R}^2, \tag{9}$$

where α is a tunable hyperparameter governing the magnitude of the compression penalty. Crucially, to ensure valid gradient flow during optimization, the $\mathcal{R}$ in this objective is the averaged overall retain rate computed using the continuous differentiable mask $\mathbf{m}$.

We perform continue pre-training using this composite objective. By varying α, we can explicitly control the equilibrium between the KV cache budget and generation quality, obtaining models with varying degrees of compression.

3.4 Storage for Variable-rate KV Compression

To support token-wise variable-rate KV compression, we store the compressed KV states of each sample in a compact contiguous cache. Since different tokens retain different numbers of dimensions, padding them to a common size would introduce unnecessary memory overhead. Instead, we concatenate the retained KV components of all tokens into a single continuous tensor and maintain an index array that records the starting position of each token's compressed representation. This allows each token's compressed KV state to be precisely located without storing a dense padded cache.

We further implement a customized attention kernel that operates directly on this compact representation. Given the continuous KV buffer and the token-wise starting indices, the kernel retrieves and reconstructs the required KV features during attention computation, without explicitly materializing the full dense KV cache.

4 Experiments

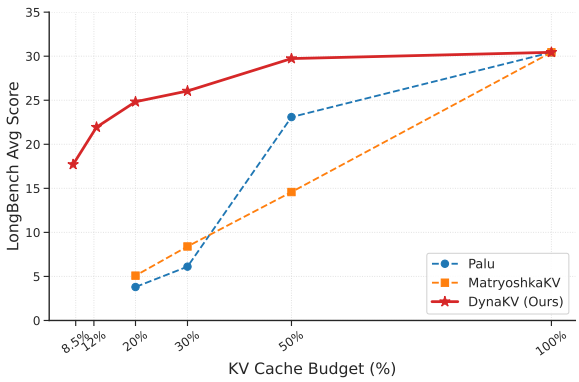

Figure 3: LongBench average scores under varying KV cache budgets. The plot compares DynaKV with baseline methods (Palu and MatryoshkaKV) across different compression rates.

4.1 Experimental Settings

Models and Benchmarks We evaluate DynaKV on LLaMA-3-8B (Touvron et al., 2023) and Qwen3-8B-Base (Yang et al., 2025). For long-context understanding, we employ LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024). To assess general capabilities, we test on standard short-context benchmarks including ARC-C, ARC-E (Clark et al., 2018), PIQA (Bisk et al., 2020), Winogrande (Sakaguchi et al., 2021), and HellaSwag (Zellers et al., 2019). Additionally, we measure

Method	Rate	Single-Doc QA			Multi-Doc QA			Summarization			Few-Shot			Synthetic		Code		Avg
		NarQA	Qasp	MF-en	Hotp	2Wiki	Musiq	GovR	QMSum	MNews	TREC	TrivQA	SAM	PasCnt	PasRet	LCC	Repo	
LLaMA-3-8B																		
Full Cache	100%	5.00	18.87	20.39	9.59	11.49	6.12	27.43	23.82	2.12	75.0	90.78	44.77	2.00	9.17	72.03	68.53	30.44
Palu	50%	7.83	16.50	17.21	9.08	9.76	**5.27**	13.49	7.00	2.31	**72.00**	83.72	26.03	**1.89**	3.45	47.13	46.69	23.09
MatryoshkaKV	50%	1.30	13.17	11.23	3.16	7.33	2.21	12.98	7.48	15.87	48.00	19.71	10.53	1.33	3.45	38.19	37.29	14.58
DynaKV	47%	**19.96**	**18.84**	**29.13**	**9.12**	**11.62**	5.22	**26.75**	**22.35**	**17.95**	71.50	**87.95**	**41.32**	0.55	**5.53**	**55.47**	**52.47**	**29.73**
Palu	30%	0.70	5.69	4.57	1.33	2.02	0.67	7.11	3.14	5.39	18.50	4.79	1.24	1.29	**4.58**	19.09	17.59	6.11
MatryoshkaKV	30%	0.51	6.08	5.69	1.47	2.47	1.02	7.37	7.84	9.79	9.00	4.81	4.76	**1.54**	3.69	**36.49**	31.84	8.40
DynaKV	29%	**18.18**	**18.10**	**29.39**	**8.97**	**9.58**	**4.40**	**25.73**	**21.13**	**10.81**	**62.50**	**88.03**	**40.38**	1.00	3.33	33.52	**41.70**	**26.05**
Palu	20%	0.11	2.52	2.16	0.24	0.65	0.44	6.40	2.74	1.65	1.00	2.33	0.48	**2.16**	**4.33**	17.75	15.81	3.80
MatryoshkaKV	20%	0.71	4.41	1.92	0.68	1.28	0.95	3.18	8.51	5.04	0.00	0.92	7.21	0.94	3.24	24.07	18.35	5.09
DynaKV	20%	**19.80**	**16.48**	**24.85**	**7.95**	**9.42**	**3.98**	**22.96**	**21.04**	**10.29**	**51.00**	**84.93**	**37.98**	0.87	4.16	**34.80**	**46.70**	**24.83**
DynaKV	12%	15.48	15.34	19.26	7.11	9.85	4.29	18.83	19.45	10.73	40.50	74.40	29.01	2.64	3.22	37.78	43.22	**21.94**
DynaKV	8.5%	2.63	13.46	15.82	5.18	8.53	3.74	16.22	17.38	18.41	30.50	35.54	22.64	2.95	4.43	43.67	42.31	**17.71**
Qwen3-8B-Base																		
Full Cache	100%	18.55	47.41	54.30	57.15	44.54	31.22	32.87	25.01	27.28	75.0	92.21	46.23	7.00	89.00	72.72	63.07	48.97
Palu	50%	**13.96**	16.37	28.93	9.11	10.42	5.51	15.98	8.93	10.89	65.5	83.90	19.89	3.08	3.82	20.31	25.16	21.36
DynaKV	48%	10.11	**23.49**	**31.08**	**11.49**	**11.63**	**7.05**	**30.51**	**24.17**	**26.76**	**72.5**	**91.68**	**41.79**	**3.39**	**56.00**	**38.41**	**34.13**	**32.14**
Palu	30%	1.08	6.09	9.39	3.16	5.31	2.03	11.65	3.75	8.58	41.5	12.17	3.03	**2.96**	2.95	17.21	19.99	9.43
DynaKV	30%	**2.51**	**19.20**	**21.23**	**11.44**	**11.59**	**8.25**	**26.29**	**23.90**	**25.85**	**69.5**	**88.16**	**38.67**	2.28	**5.67**	**32.13**	**34.72**	**26.34**
Palu	20%	0.79	3.73	4.20	1.61	1.70	1.39	10.72	3.53	7.15	9.5	4.19	3.26	**3.33**	0.65	**17.24**	15.15	5.51
DynaKV	18%	**2.68**	**19.20**	**16.60**	**9.95**	**10.21**	**4.23**	**20.89**	**21.21**	**22.73**	**60.0**	**79.98**	**36.16**	2.06	**5.00**	32.42	**36.40**	**23.73**

Table 1: Detailed performance on LongBench categorized by task type. We report the score for each of the 16 subsets and the overall average (Avg). **Bold** indicates the best performance under comparable compression rates.

Method	Rate	ARC-C	ARC-E	PIQA	Wino	Hella	Avg
LLaMA-3-8B							
Full Cache	100%	45.76	70.02	80.85	71.51	74.68	68.56
Palu	50%	39.32	63.84	77.58	**67.80**	69.00	63.51
MatryoshkaKV	50%	38.64	60.49	78.02	63.14	67.36	61.53
DynaKV	47%	**44.75**	**67.37**	**80.36**	67.64	**71.70**	**66.36**
Palu	30%	29.15	27.51	71.60	60.62	57.19	49.21
MatryoshkaKV	30%	32.88	49.74	73.67	55.80	56.80	53.78
DynaKV	29%	**42.71**	**59.96**	**79.82**	**66.54**	**70.33**	**63.87**
Palu	20%	29.49	28.57	65.72	54.70	46.47	44.99
MatryoshkaKV	20%	27.46	45.15	68.50	53.83	45.31	48.05
DynaKV	20%	**38.98**	**59.44**	**78.56**	**65.51**	**67.89**	**62.08**
Qwen3-8B-Base							
Full Cache	100%	44.41	71.60	79.76	70.40	72.28	67.69
Palu	50%	32.88	49.74	71.38	62.43	60.74	55.43
DynaKV	48%	**42.37**	**70.19**	**78.78**	**64.96**	**68.18**	**64.90**
Palu	30%	27.12	37.74	61.97	53.75	40.04	44.12
DynaKV	30%	**40.00**	**50.79**	**76.71**	**63.14**	**67.08**	**59.54**
Palu	20%	26.44	31.57	56.91	50.99	32.39	39.66
DynaKV	18%	**39.32**	**59.96**	**75.41**	**58.72**	**62.44**	**59.17**

Table 2: Zero-shot performance on short-context benchmarks. "Avg" denotes the average score across these five tasks. **Bold** indicates the best performance under comparable compression rates (excluding Full Cache).

perplexity (PPL) on C4 (Raffel et al., 2020) and PG-19 (Rae et al., 2019).

Baselines We compare our method with Palu (Chang et al., 2025), a low-rank compression approach for KV cache. Specifically, we adopt the J-LRD variant for evaluation. We also compare against MatryoshkaKV (Lin et al., 2024), which employs a nested embedding structure for static truncation, serving as a fixed-rank baseline.

Implementation Details Our model is continued pre-trained on the RedPajama-V2-sample dataset (Weber et al., 2024) for 128 million tokens within 8 GPU-hours. By varying $\alpha \in \{19, 32, 50\}$, we obtain models with different compression rates. The average retain ratio of our models is measured on the WikiText-2 dataset (Merity et al., 2016). All experiments are performed on NVIDIA H200 GPUs. MatryoshkaKV is excluded from the Qwen3-8B-Base model due to compatibility issues. The extreme variance in Qwen3's specific dimensions causes the inverse Cayley transform applied in MatryoshkaKV to produce NaN values, preventing the successful construction of the compressed basis.

4.2 Main Results

Performance on Short-Context Tasks As shown in Table 2, DynaKV consistently outperforms both Palu and MatryoshkaKV across all settings, and the performance gap widens significantly under aggressive compression regimes. Specifically, at a 20% retention rate on LLaMA-3-8B, baseline methods suffer severe degradation, where Palu and MatryoshkaKV drop to 44.99 and 48.05, respectively. In contrast, DynaKV sustains a robust average of 62.08, surpassing the strongest baseline by over 14.03 points. A similar trend is observed on Qwen3-8B; DynaKV dominates Palu (59.17 vs. 39.66 at ∼18% rate), validating that our token-adaptive strategy effectively preserves critical semantic information even when the memory budget is strictly limited.

Method	Rate	Retrieval (Single)			Retrieval (Multi)					QA		Reas.	Agg.		Avg
		S-1	S-2	S-3	MK-1	MK-2	MK-3	MQ	MV	Squad	Hotpot	VT	CWE	FWE	
LLaMA-3-8B															
Full Cache	100%	50.0	50.0	50.0	49.8	49.8	49.0	49.9	49.9	37.0	28.3	49.7	49.2	44.7	46.7
Palu	50%	**50.0**	49.5	33.3	45.8	**48.5**	34.3	36.4	22.8	27.5	23.0	**45.4**	28.7	32.7	36.8
MatryoshkaKV	50%	49.8	46.0	24.8	16.0	1.8	0.0	7.2	3.8	7.2	11.0	0.1	6.9	3.9	13.7
DynaKV	47%	**50.0**	**50.0**	**48.0**	**46.8**	47.0	**43.3**	**46.5**	**46.3**	**36.3**	**28.5**	30.2	**36.9**	**41.5**	**42.4**
Palu	30%	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.8	2.0	0.0	0.0	7.8	1.5
MatryoshkaKV	30%	39.5	16.0	0.0	4.2	0.0	0.0	0.1	0.4	2.5	2.2	0.0	2.4	6.7	5.7
DynaKV	29%	**50.0**	**50.0**	**49.3**	**47.8**	**47.5**	**35.0**	**44.9**	**41.3**	**34.3**	**25.5**	**29.8**	**19.8**	**37.8**	**39.4**
Palu	20%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0
MatryoshkaKV	20%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.2	0.0	0.1	2.2	0.2
DynaKV	20%	**49.8**	**50.0**	**47.0**	**46.0**	**38.8**	**23.5**	**39.1**	**34.9**	**27.5**	**24.3**	**33.6**	**8.1**	**39.8**	**35.6**
Qwen3-8B-Base															
Full Cache	100%	100.0	100.0	100.0	98.8	95.0	92.0	93.9	98.0	76.3	55.3	98.2	76.6	93.8	90.6
Palu	50%	100.0	99.3	98.0	85.3	63.0	37.0	84.5	81.6	39.0	33.3	31.4	25.6	76.0	65.7
DynaKV	48%	**100.0**	**100.0**	**98.8**	**88.3**	**77.5**	**66.5**	**90.1**	**91.9**	**70.3**	**59.5**	**97.7**	**45.5**	**78.5**	**81.9**
Palu	30%	96.8	47.0	9.5	32.3	2.3	0.0	31.6	25.8	6.3	8.3	0.0	0.6	1.7	20.2
DynaKV	30%	**100.0**	**99.8**	**99.0**	**79.3**	**75.3**	**49.8**	**77.5**	**78.7**	**51.3**	**39.0**	**92.3**	**24.4**	**42.3**	**69.9**
Palu	20%	9.0	0.0	0.0	1.5	0.0	0.0	1.9	2.3	0.5	1.3	0.0	0.0	0.2	1.3
DynaKV	18%	**100.0**	**95.8**	**91.5**	**57.3**	**15.8**	**0.3**	**62.8**	**70.6**	**38.8**	**35.3**	**40.8**	**18.1**	**40.8**	**51.3**

Table 3: Performance on Ruler benchmark. Scores are averaged across context lengths from 4k to 32k. **Bold** indicates the best performance under comparable compression rates.

Method	Rate	C4	PG-19
LLaMA-3-8B			
Full Cache	100%	9.49	9.26
Palu	50%	15.20	14.93
MatryoshkaKV	50%	22.14	20.35
DynaKV	47%	**10.73**	**10.46**
Palu	30%	48.51	53.41
MatryoshkaKV	30%	43.87	38.87
DynaKV	29%	**11.46**	**11.34**
Palu	20%	113.90	127.88
MatryoshkaKV	20%	137.59	94.64
DynaKV	20%	**12.51**	**13.09**
Qwen3-8B-Base			
Full Cache	100%	11.71	9.77
Palu	50%	17.73	19.26
DynaKV	48%	**12.97**	**11.59**
Palu	30%	47.58	50.78
DynaKV	30%	**14.30**	**15.30**
Palu	20%	123.51	149.38
DynaKV	18%	**16.01**	**18.71**

Table 4: Perplexity (PPL) evaluation on C4 and PG-19 datasets. Lower is better. **Bold** indicates the best performance under comparable compression rates.

Performance on Long-Context Tasks Figure 3 illustrates the performance trends on LongBench (Bai et al., 2024). While methods with fixed compression rate degrade rapidly as the memory budget decreases, DynaKV maintains robust understanding even at extreme compression: it achieves a score of 17.71 with only 8.5% cache budget, surpassing Palu's performance at a much larger 30% budget (6.11). This advantage is further amplified on the RULER benchmark (Hsieh et al., 2024) (Table 3), where baselines fail completely at a 30% retention rate (scoring < 6.0), while DynaKV retains a competitive 39.4. Similar gains on Qwen3-8B confirm that dynamic capacity allocation effectively prevents context collapse under strict memory constraints.

Perplexity Finally, we quantify fundamental information loss via perplexity (PPL) measurements on the C4 and PG-19 datasets (Table 4). The results highlight a clear distinction in stability. Baselines like Palu and MatryoshkaKV experience catastrophic PPL surges as the retention rate decreases. For instance, at a 20% retention rate on LLaMA-3-8B, Palu's PPL on C4 spikes to 113.90 and Matryoshka to 137.59, indicating severe language degradation. In contrast, DynaKV maintains a low PPL of 12.51 at the same 20% retention rate. This result verifies that our method effectively preserves core linguistic capabilities even under aggressive compression.

End-to-End Efficiency and Memory Reduction We evaluate the practical decoding efficiency of different KV compression methods on LLaMA-3-8B using a single NVIDIA RTX 4090 GPU. For each run, we generate 100 tokens and measure the average decoding time per token; the reported throughput is averaged over 100 repeated runs. As shown in Figure 4, DynaKV achieves 29.84 tokens/s, slightly outperforming the uncompressed baseline while clearly surpassing Palu and MatryoshkaKV. This result indicates that our flattened ragged KV layout and customized attention kernel effectively support token-adaptive compression

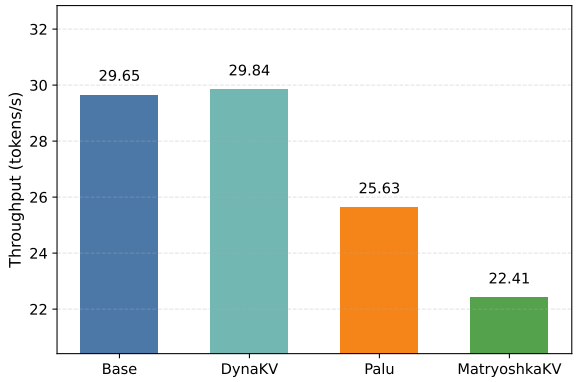

Figure 4: Decoding throughput comparison on LLaMA-3. Throughput is measured by generating 100 tokens and averaging the per-token decoding time over 100 repeated runs.

Model	KV Cache Size
LLaMA-3-8B	1024.00 MB
DynaKV 47%	493.26 MB
DynaKV 29%	307.71 MB
DynaKV 20%	197.87 MB

Table 5: KV cache memory usage of LLaMA-3-8B and DynaKV under different retention rates.

without introducing noticeable decoding overhead.

We further measure the actual GPU memory footprint of the KV cache under different compression rates. Specifically, we randomly sample an 8k-token text segment from WikiText and record the memory occupied by the KV cache after processing the sequence. As shown in Table 5, the full-cache LLaMA-3-8B baseline requires 1024.00 MB for the KV cache. In comparison, DynaKV substantially reduces the cache memory usage: DynaKV 47% requires only 493.26 MB, while DynaKV 29% and DynaKV 20% further reduce the KV cache size to 307.71 MB and 197.87 MB, respectively. These results confirm that DynaKV achieves real memory savings in practical inference.

4.3 Integration with Token-Eviction Methods

DynaKV compresses the KV cache along the feature/channel dimension, which is orthogonal to sequence-level token eviction methods such as SnapKV (Li et al., 2024). Since the two methods operate on different dimensions of the KV cache, they can be naturally combined for more aggressive memory reduction. To verify this compatibility, we integrate DynaKV with SnapKV and evaluate the combined method on LLaMA-3-8B using LongBench. As shown in Table 6, the combined approach remains highly robust under extreme compression. Even with a strict 6% overall KV budget, it preserves 94% of the original performance. These results demonstrate that DynaKV can effectively complement token-eviction mechanisms, enabling substantial memory savings while maintaining strong long-context capability.

Configuration	Base	Budget	LB Avg
Full Cache	8192	100.0%	30.44
SnapKV (2048)	2048	25.0%	29.95
Snap(2048) + DynaKV 47%	2048	~11.8%	29.19
Snap(2048) + DynaKV 30%	2048	~7.5%	25.60
SnapKV (1024)	1024	12.5%	29.56
Snap(1024) + DynaKV 47%	1024	~5.9%	28.88
Snap(1024) + DynaKV 30%	1024	~3.7%	25.08

Table 6: Synergistic performance of DynaKV integrated with SnapKV on LongBench (LLaMA-3-8B). **Snap(L)+DynaKV** R**%** denotes applying SnapKV to retain L tokens, followed by DynaKV compression at rate $R\%$.

5 Conclusion

In this work, we introduce DynaKV, a novel post-training framework for low-rank KV cache compression. To the best of our knowledge, DynaKV is the first approach to move beyond the rigid "one-size-fits-all" compression paradigm by implementing a **token-adaptive** strategy. By dynamically allocating compression rates based on the semantic significance of individual tokens, our method effectively resolves the dilemma between compression ratio and model performance that limits existing techniques.

Extensive experiments validate that DynaKV can be seamlessly adapted to pre-trained LLMs with minimal training overhead, achieving significant memory reduction while maintaining robust generation quality where uniform compression methods fail. Furthermore, we demonstrate that our dimensionality reduction approach is orthogonal to sequence-level pruning, allowing for synergistic integration with methods like SnapKV to push the boundaries of memory efficiency. We hope this work inspires further exploration into adaptive, semantics-aware compression strategies for scalable LLM deployment.

Limitations

A limitation of this work is that our evaluation is restricted to models with relatively moderate context lengths. Since DynaKV requires post-training calibration, extending it to long-context models entails additional computational cost. Due to time and resource constraints, we have not yet trained or evaluated DynaKV on models with very long context windows, such as LLaMA-3.1 with 128K context length. As a result, its effectiveness under extreme long-context scenarios remains an open direction. We will include experiments on longer-context models and benchmarks in future work.

References

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 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.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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*.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. 2025. Palu: Kv-cache compression with low-rank projection. In *The Thirteenth International Conference on Learning Representations*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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? *arXiv preprint arXiv:2404.06654*.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. 2025. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms. *arXiv preprint arXiv:2502.14837*.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, Tianqi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. 2024. Matryoshkakv: Adaptive kv compression via trainable orthogonal projection. *arXiv preprint arXiv:2410.14731*.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen attention: Attention in low-rank space for kv cache compression. *arXiv preprint arXiv:2408.05646*.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron Courville. 2017. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2018. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*.

Kaitao Song, Xiaohua Wang, Xu Tan, Huiqiang Jiang, Chengruidong Zhang, Yongliang Shen, Cen Lu, Zihao Li, Zifan Song, Caihua Shan, and 1 others. 2025. Chain-of-model learning for language model. *arXiv preprint arXiv:2505.11820*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. 2025. Commonkv: Compressing kv cache with cross-layer parameter sharing. *arXiv preprint arXiv:2508.16134*.

Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, and 1 others. 2024. Redpajama: an open dataset for training large language models. *Advances in neural information processing systems*, 37:116462–116492.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Xianglong Yan, Zhiteng Li, Tianao Zhang, Haotong Qin, Linghe Kong, Yulun Zhang, and Xiaokang Yang. 2025. Recalkv: Low-rank kv cache compression via head reordering and offline calibration. *arXiv preprint arXiv:2505.24357*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. 2023. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

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.

A Compression Rate Allocation Analysis

Figure 5 and Figure 6 visualizes the token-wise and layer-wise compression rate allocation of DynaKV on a sample sentence: *"I keep telling myself that I am going to start working on it tomorrow, but to be honest, I am just struggling with chronic procrastination."* The bar chart and heatmap reveal three distinct characteristics of our adaptive compression strategy:

- **Dominance of Attention Sinks:** First, we observe a significant retention bias towards the initial token ($< BOS >$). As shown in Figure 6, it consistently maintains the highest average retention rate ($\sim$0.75) across all layers. This aligns with the "attention sink" phenomenon observed in previous studies (Xiao et al., 2023), where models allocate disproportionate attention mass to initial tokens to stabilize inference. DynaKV successfully identifies and preserves this critical anchor despite its lack of semantic content.

- **Semantic-Adaptive Allocation:** Second, the method demonstrates a clear ability to distinguish between high-value semantic tokens and low-information functional ones. Rare or complex words such as *"chronic"* and *"procrastination"* (tokenized as *"procrast"* and *"ination"*) are assigned significantly higher retention rates. In contrast, common functional tokens and stopwords—such as *"that"*, *"to"*, *"be"*, and *"just"*—are aggressively compressed, showing much lower retention rates. This confirms that DynaKV effectively allocates limited memory budget to tokens that contribute most to the semantic density of the context.

- **Hierarchical Layer-wise Distribution:** Third, Figure 6 reveals a distinct hierarchical pattern in compression intensity. The lower layers exhibit generally higher retention

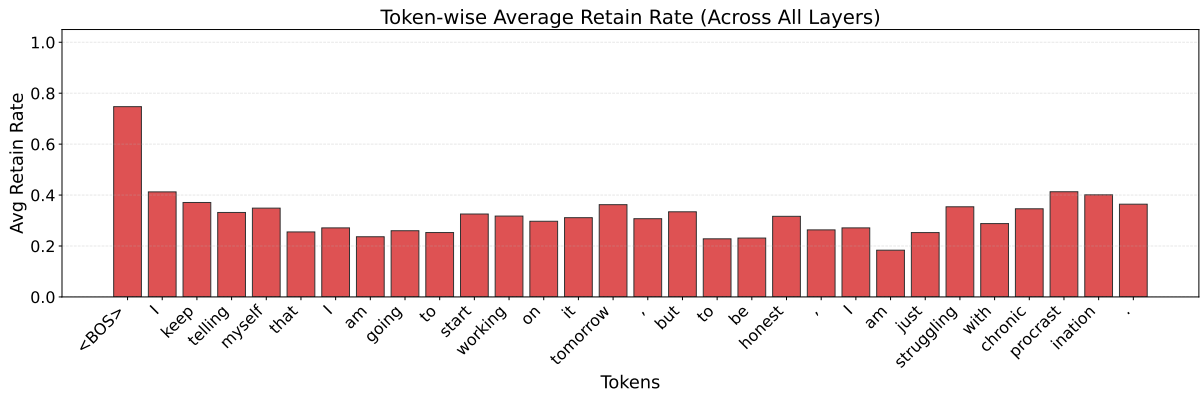

Figure 5: Token-wise average retention rate across all layers. The bar height represents the budget allocated to each token, highlighting the high retention of the sink token and semantic tokens compared to functional ones.

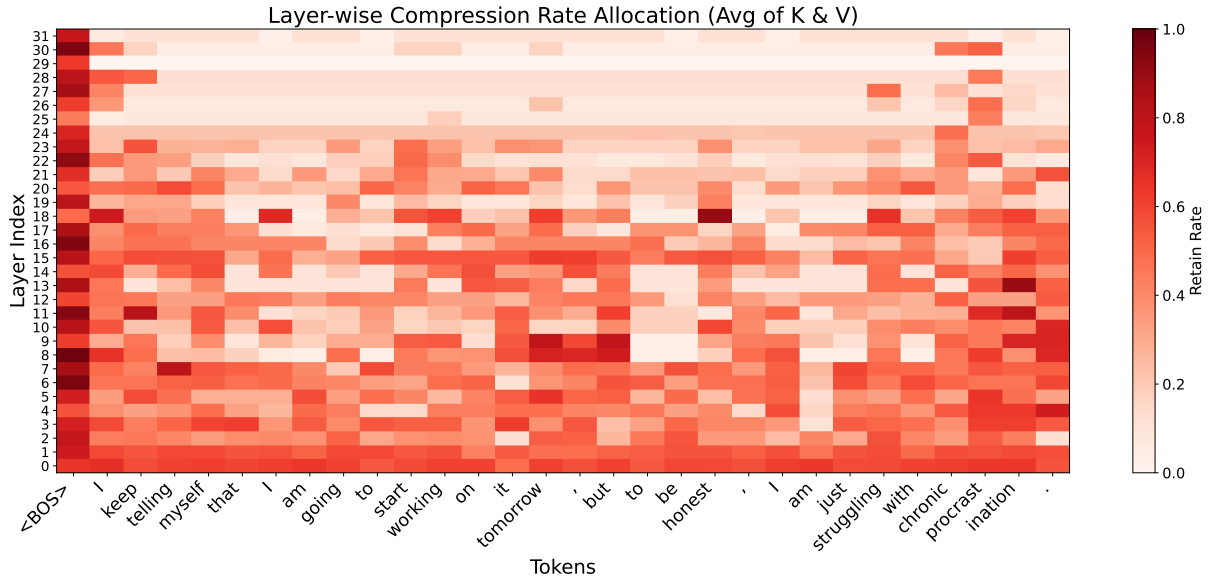

Figure 6: Layer-wise compression rate allocation heatmap. The x-axis denotes input tokens and the y-axis denotes model layers (0–31). Darker red indicates a higher retention rate, visualizing the dynamic distribution of memory budgets across different network depths.

rates, suggesting that early layers prioritize preserving a broader range of syntactic information. Conversely, the deeper layers show a shift towards lighter colors, indicating higher compression rates. This suggests that as the model abstracts information in higher layers, it requires fewer tokens to maintain context, allowing DynaKV to compress more aggressively with low performance loss.

B Discuss For Artifacts And Packages

All existing artifacts used in this work are employed consistently with their intended research and evaluation purposes. The pre-trained language models are used only as backbone models for studying KV-cache compression and efficient inference. The training and calibration corpora are used solely for post-training calibration and retention-ratio measurement. The evaluation benchmarks are used only for their intended purpose of measuring model capabilities, including long-context understanding, perplexity, and short-context reasoning performance. Baseline implementations are used only for academic comparison under their released licenses and are properly cited.

We use OpenCompass for evaluation. For LLaMA-3-8B, we set the maximum context length to 8192 tokens, while for Qwen3-8B-Base we use a 32K context length. All evaluations are conducted with batch size 1, and all other hyperparameters follow the default or recommended settings of the corresponding benchmarks.

Artifact	Type	License / Terms	Usage in This Work
Meta LLaMA-3-8B	Model	Meta Llama 3 Community License Agreement	Used as one of the backbone LLMs for evaluating DynaKV.
Qwen3-8B-Base	Model	Apache License 2.0	Used as one of the backbone LLMs for evaluating DynaKV.
RedPajama-V2-sample	Training dataset	Data follows Common Crawl Foundation Terms of Use; processing code is Apache License 2.0	Used for continued pre-training / post-training calibration.
WikiText-2	Calibration / measurement dataset	Creative Commons Attribution-ShareAlike License	Used to measure the average retention ratio of DynaKV models.
C4	Perplexity dataset	ODC-BY; also subject to Common Crawl Terms of Use	Used for perplexity evaluation.
PG-19	Perplexity dataset	Apache License 2.0; texts are derived from Project Gutenberg books	Used for perplexity evaluation.
LongBench	Benchmark	MIT License	Used for long-context understanding evaluation.
RULER	Benchmark	Apache License 2.0	Used for long-context capability evaluation.
ARC-Challenge / ARC-Easy	Benchmark	CC BY-SA 4.0	Used for short-context zero-shot evaluation.
PIQA	Benchmark	Not specified / unknown in the public dataset card	Used for short-context physical commonsense evaluation.
WinoGrande	Benchmark	Dataset: CC-BY; codebase: Apache License 2.0	Used for short-context commonsense evaluation.
HellaSwag	Benchmark	MIT License	Used for short-context commonsense evaluation.
Palu	Baseline implementation	MIT License	Used as a low-rank KV-cache compression baseline.
MatryoshkaKV	Baseline implementation	No explicit license found in the public code repository	Used as a fixed-rank / trainable projection KV-cache compression baseline.
SnapKV	Baseline implementation	Apache License 2.0	Used to evaluate compatibility with token-eviction methods.

Table 7: Licenses and terms of use for the artifacts used in this work. We report the license or usage terms provided by the official model, dataset, benchmark, or code repositories.

C Information About Use Of AI

We used AI assistants, including ChatGPT, during the preparation of this work. The assistance was limited to language polishing, grammar checking, LaTeX formatting, code debugging, and improving the clarity of explanations. All technical ideas, experimental designs, implementation decisions, results, analyses, and final claims were developed, verified, and approved by the authors. We did not use AI assistants to generate experimental results, fabricate citations, or replace human judgment in the research process.