DynamicKV: Task-Aware Adaptive KV Cache Compression for Long Context LLMs

Anonymous ACL submission

Abstract

Efficient KV cache management in LLMs is crucial for long-context tasks like RAG and summarization. Existing KV cache compression methods enforce a fixed pattern, neglecting task-specific characteristics and reducing the retention of essential information. However, we observe distinct activation patterns across layers in various tasks, highlighting the need for adaptive strategies tailored to each task's unique demands. Based on this insight, we propose **DynamicKV**, a method that dynamically optimizes token retention by adjusting the number of tokens retained at each layer to adapt to the specific task. DynamicKV establishes global and per-layer maximum KV cache budgets, temporarily retaining the maximum budget for the current layer, and periodically updating the KV cache sizes of all preceding layers during inference. Our method **retains only** 1.7% **of the KV cache size while achieving** $\sim 85\%$ **of the Full KV cache performance** on LongBench. Notably, even under extreme compression (0.9%), **DynamicKV surpasses state-of-the-art (SOTA) methods by 11% in the Needle-in-a-Haystack test** using Mistral-7B-Instruct-v0.2. The code will be released.

1 Introduction

Large Language Models (LLMs) (Achiam et al., 2023; Radford, 2018; Radford et al., 2019) are exerting a considerable influence in the field of natural language processing (NLP), driving advancements in summarization, translation, code generation, etc. (Chiang et al., 2023; Zhong et al., 2023; Peng et al., 2023; Lu et al., 2024; Wang et al., 2024). Recent developments in LLMs (Liu et al., 2024b) have been scaled up to handle long contexts, with LlaMA3 (Dubey et al., 2024) processing up to 32K tokens and InternLM (Cai et al., 2024) handling 1M tokens. However, scaling LLMs to handle extended contexts inherently incurs a substantial delay due to the quadratic complexity of attention mechanisms with increasing context length. A widely adopted solution to alleviate these delays is caching the key and value (KV) states of previous tokens (Waddington et al., 2013). Despite this optimization, handling long sequences still demands substantial memory (*e.g.,* maintaining a KV cache for 100K tokens in LlaMA2-7B (Touvron et al., 2023) consumes over 50 GB of memory).

To address this issue, recent studies have explored the optimization of KV caching, including KV cache quantization (Kang et al., 2024; Hooper et al., 2024), token dropping (Zhang et al., 2024b; Xiao et al., 2023), architectural improvements to Transformers (Sun et al., 2024), KV cache fusion (Nawrot et al., 2024), and hierarchical sharing and constraints(Liu et al., 2024a; Brandon et al., 2024). Existing KV cache compression methods enforce a fixed pattern (as shown in Figure 1), such as a hierarchical pyramid structure (Zhang et al., 2024a) or a structure similar to FastGen's fixed internal pattern (Ge et al., 2023), or they fix the length of the KV cache to selectively retain tokens across different layers (Zhang et al., 2024b; Li et al., 2024). However, LLMs require different numbers of layers when handling different types of tasks. For example, for knowledge-based question-answering tasks, only the first few layers are needed to achieve high accuracy, while for complex reasoning tasks (*e.g.,* mathematics and code generation), more layers are often required to achieve higher accuracy (Elhoushi et al., 2024). Thus, we raise a question: *Do different types of tasks all follow a fixed pattern?*

To examine this question, we aim to systematically investigate the design principles of the KV cache compression across different tasks. Inspired by Zhang et al. (2024a), we first investigate how information flow is aggregated through attention mechanisms across different layers in four types of tasks, including single- and multi-document QA, summarization, synthetic tasks and code completion. We find that the attention distribution varies

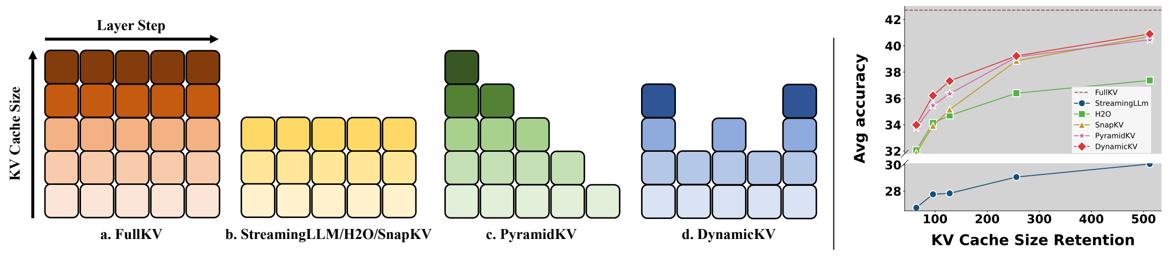


Figure 1: **Comparison of DynamicKV with traditional methods in maintaining KV cache size across layers.** Left: the structure difference: (a) Retain all KV cache. (b) Fixed KV cache for each layer (e.g., StreamingLLM, H2O, SnapKV). (c) Hierarchically decreasing pyramid KV cache retention. (d) Ours DynamicKV: layer-aware adaptive KV cache retention. Right: average accuracy on different KV cache retention.

for different types of tasks. For example, in summarization tasks, the upper layers require a small KV cache size, while code completion tasks need larger KV cache sizes in the upper layers. This implies that for code completion tasks, upper layers require maintaining a larger KV cache size, in contrast to PyramidKV (Zhang et al., 2024a), where the KV cache size decreases as the layer depth increases.

Building on this insight, we propose a task-aware adaptive KV cache compression method, named DynamicKV. Specifically, we first calculate an attention score for the most recent few tokens and all other tokens, which in RAG (Lewis et al., 2020) can be viewed as calculating the relevance of the most recent query to the retrieved text. Then, we preset a temporary storage to hold the temporary KV cache states and gradually calculate the size of the final retained temporary storage at each k layer by calculating the size of the correlation mean. It should be noted that at each update, the value is gradually normalized, and the retained temporary storage at each layer is always smaller than the previous one. This temporary storage is determined by the number of tokens that need to be retained, and its size is much smaller than the original cache, thus imposing minimal memory overhead.

We validate our DynamicKV on 16 datasets from LongBench (Bai et al., 2023), demonstrating robust performance across multiple models, including LlaMA-3-8B-Instruct (Dubey et al., 2024), Qwen-2-7B-Instruct (Yang et al., 2024), Mistral-7b-Instruct-v0.2 (Jiang et al., 2023), InternLM-2.5-7b-Chat-1M (Cai et al., 2024). Our DynamicKV exhibits superior overall effectiveness compared to conventional fixed-pattern methods (Zhang et al., 2024b; Li et al., 2024; Nawrot et al., 2024). Notably, DynamicKV can retain full performance while utilizing only 6.9% of the tokens, and in extreme scenarios, it preserves 90% of the performance with just 1.7% of the tokens. Furthermore, experiments on the Needle in a Haystack benchmark revealed that DynamicKV significantly outperforms state-of-the-art (SOTA) methods.

2 Related Work

Potential patterns of attention in LLMs. The Transformer architecture (Vaswani, 2017) becomes a cornerstone in NLP by stacking multiple layers to progressively refine input data. BERT (Devlin, 2018), a model based on this architecture, Jawahar et al. (2019) demonstrates that intermediate layers encode a rich hierarchy of linguistic information: from surface-level features at the bottom, through syntactic features in the middle, to semantic features at the top. This indicates that models are capable not only of understanding lexical information but also of grasping more complex linguistic structures.

For decoder-only LLMs, Fan et al. (2024) observes that not all layers are necessary for simple tasks, as intermediate layers can often achieve comparable performance to the final layer. Techniques like (Elhoushi et al., 2024), which involve increasing dropout in lower layers during training, allow the model to exit computation early, reducing resource consumption.

To optimize model inference efficiency, especially in terms of KV cache compression, Brandon et al. (2024) proposes cross-layer attention(CLA), which can reduce the KV cache size by at least half by sharing cross-layer attention, significantly lowering memory usage. Ada-KV (Feng et al., 2024) visualizes attention distributions across all layers have also shown that attention patterns dynamically evolve as the layers progress. Inspired by these findings, we aim to dynamically select and adjust the number of tokens to retain per layer, combining

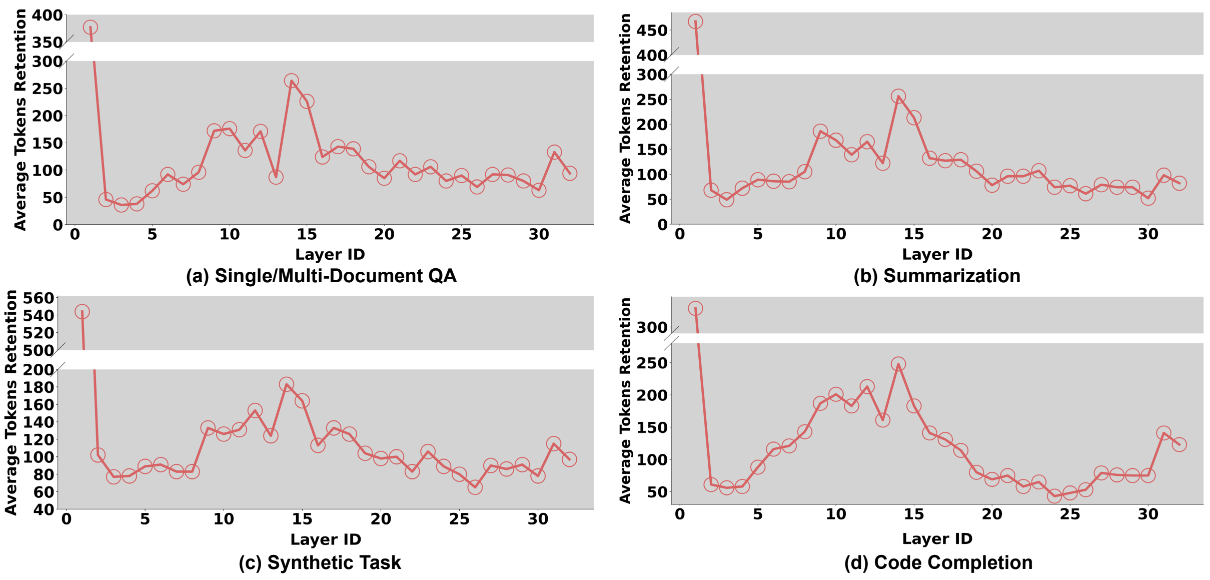


Figure 2: **Average token retention across layers in LlaMA for different tasks**, including (a) *Single/Multi-Document QA*, (b) *Summarization*, (c) *Synthetic Task*, and (d) *Code Completion*. There is a sharp decrease in token retention after the first layer, followed by varying patterns of fluctuation. Peaks are observed around Layer 15 and towards the final layers.

inter-layer redundancy identification with efficient KV cache management. This approach aims to maintain high-quality output while improving inference efficiency.

Token drop strategies in KV cache compression. Token drop is a strategy designed to reduce memory usage by selectively retaining the most influential tokens in the KV cache during the inference phase of LLMs. Due to its plug-and-play nature, the token drop method can often be applied to different models without incurring any additional costs. FastGen (Ge et al., 2023) evicts unnecessary contexts and discards non-special tokens based on the recognized structure of attention modules by effectively analyzing the token information within attention patterns. Scissorhands (Liu et al., 2024c) exploits the hypothesis of the persistence of importance, suggesting that tokens with significant influence at one point will continue to impact future generations. By using attention scores as a metric and applying a Least Recently Used (LRU) cache eviction strategy, it discards non-critical tokens to optimize memory usage. StreamingLLM (Xiao et al., 2023) leverages the characteristics of attention sinks in LLMs to focus on streaming processing with dynamic adjustment of the KV cache. H2O (Zhang et al., 2024b) proposes a scoring function based on accumulated attention scores for greedily evicting KV pairs during generation. SnapKV (Li et al., 2024) primarily achieves compression by selectively targeting key positions for each attention head. PyramidKV (Zhang et al., 2024a) identified the phenomenon of massive activation and adopted a hierarchical structure to optimize the number of KV cache entries retained at each layer. Although the PyramidKV approach considers the varying information density across different layers, its pyramidal pattern does not generalize across multiple models or tasks. LazyLLM (Fu et al., 2024) utilizes dynamic token pruning and an Aux Cache mechanism, allowing the model to select different subsets of tokens from the context at various generation steps, even reviving tokens pruned in previous steps. Ada-KV (Feng et al., 2024) breaks from the conventional approach of uniform budget allocation across attention heads within layers, and optimizes the eviction loss upper bound, leading to improved performance under various memory constraints when integrated with SnapKV and PyramidKV.

3 Observation

To systematically investigate the attention mechanism across layers in LLMs for long-context inputs, we conduct a fine-grained analysis of four tasks: single- and multi-document question answering (QA), summarization, synthetic tasks, and code completion. The main target is to investigate the distribution of attention in these various tasks, thereby enhancing our understanding of how the model aggregates dispread information within long-context inputs to generate accurate responses.

In particular, we focus our analysis on LlaMA (Dubey et al., 2024), visualizing the distribution and behavior of attention across layers to gain deeper insights into its internal mechanisms. Inspired by Zhang et al. (2024a), we calculate the average attention scores between the most recent tokens and all other tokens. Based on these scores, we then identify the top-k (128 multiplied by the number of layers) tokens with the highest attention across all layers, resulting in a layer distribution map denoted as Figure 2.

We observe a significant drop in the KV cache size requirement at the lower layers across the four tasks, indicating that only a small KV cache is needed in these layers. In contrast, the upper layers show a clear upward trend, suggesting that larger KV cache sizes are necessary, particularly in the code completion task, where complex reasoning is required. This phenomenon underscores that tasks involving complex reasoning demand larger KV cache sizes in the upper layers.

4 DynamicKV

During inference, the quadratic complexity of attention calculation results in a significant computational and memory burden, especially when processing long contexts. DynamicKV addresses this issue by focusing on inter-layer attention in large language models (LLMs), determining the appropriate size of KV cache to retain per layer through efficient awareness of inter-layer attention.

Rather than relying on a fixed retention pattern, such as pyramid-shape or average retention all layers, DynamicKV employs a progressive algorithm that dynamically adjusts token retention during the prefill phase. This dynamic retention strategy accelerates the decoding stage while maintaining minimal impact on overall memory usage.

Specifically, we first define layer $l \in \mathbb{R}^L$ and head $h \in \mathbb{R}^H$ in LLMs. For the calculation of attention scores, we use weights $W_Q \in \mathbb{R}^{N \times N}$, $W_K \in \mathbb{R}^{N \times N}$, and $W_V \in \mathbb{R}^{N \times N}$, with the input query embedding denoted as $X \in \mathbb{R}^{N \times M}$, N is the dimension of the hidden size, and M is the length of input tokens. Traditional token drop methods often consider the most recent tokens as the important ones for producing output information, as they retain relevant information needed for generating answers. We refer to these tokens collectively as the *current window*, with the window size denoted as ws. In the prefill phase, we adopt the method from Li et al. (2024), Zhang et al. (2024a), where the attention score is calculated by averaging over the current window and previous tokens, followed by pooling. The formula is as follows:

$$A_{l,h} = \text{pooling}(\frac{1}{ws}\sum_{i=1}^{ws} \text{Attention}(X_i, W_Q, W_K)), \tag{1}$$

where pooling helps better understand the context and $A_{l,h}$ denotes the attention score for the l-th layer and the h-th head. This approach allows us to effectively pool the attention scores, ensuring that key tokens are retained based on their relevance to both the current window and previous context.

Next, we set a fixed retention budget. Specifically, to ensure a fair comparison with other methods, we introduce the average retention length per layer, denoted as wt, and a scaling ratio, r_{max}. The calculation formula is as follows:

$$bs = (wt - ws) \times r_{max}, \tag{2}$$

where bs represents the size of retained KV cache across all layers. Next, we design a layer-aware progressive dynamic KV cache compression method. The prefill phase of LLMs involves a hierarchical forward process, where for each layer, we retain a KV cache of length bs when computing A. Additionally, every m layers, we perform an update across the current and all previous layers. Specifically, for each layer, we use a top-k strategy to retain the largest bs values from A_l, where A_l represents the attention scores of layer l. The formula for this process is as follows:

$$A'_l = \text{TopK}(A_l, bs). \tag{3}$$

Next, we extract the indices in the original A_l that correspond to the values in A'_l. The KV cache at these indices is retained as the compressed KV cache. Specifically, the retained KV cache is defined as:

$$\text{KV}'_l = \text{KV}_l[A'_l.\text{indices}], \tag{4}$$

where A'_l.indices represents the indices of the top-k values in A_l. This ensures that the KV cache is compressed efficiently, retaining only the most important tokens for each layer while minimizing memory usage.

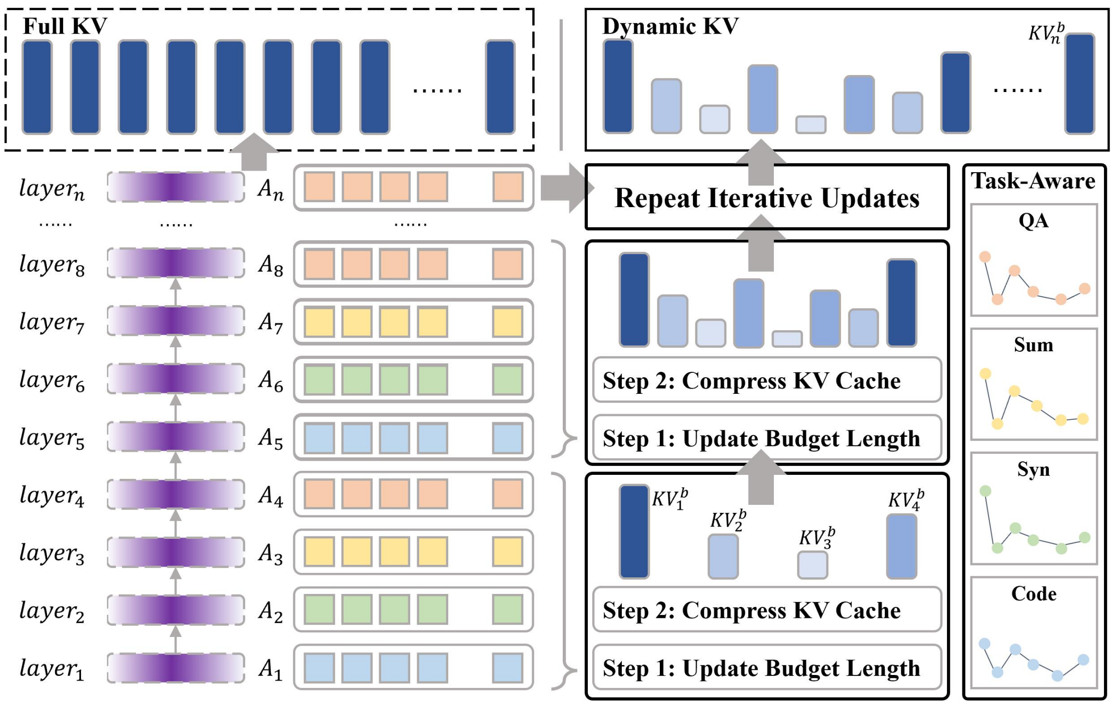


Figure 3: **Overview of our DynamicKV structure and KV cache compression comparison.** Left: Layer-wise KV cache retention mechanism in transformer architectures. Right: Our proposed DynamicKV framework employs stage-wise dynamic updating to maintain KV cache within predefined memory budgets, with task-specific visualization showing KV cache preservation patterns across layers.

To ensure that the memory required for hierarchical transmission remains small, the KV cache of each layer is initially compressed as described above. For every m layer, we extract A and perform a unified normalization across the completed layers, updating them layer by layer to ensure consistency across the entire hierarchy.

First, we fix the final size of the KV cache to be retained, which is calculated as $(wt - ws) \times H \times l$, where H is the number of heads and l is the number of layers. Then, for each layer, the attention score A is used to compute the length to retain for each layer C_l via a top-k strategy. The retention lengths for the first m layers are then normalized to obtain a budget length Z, ensuring that the retention is distributed effectively across layers. The specific formula is as follows:

$$I = \frac{\mathrm{TopK}_{indices}(A, (wt - ws) \times H \times l)}{(L \times M \times l)} \tag{5}$$

$$C_l = \mathrm{Norm}(\mathrm{Count\_Elements}\,(I)) \tag{6}$$

$$Z = \left[\frac{bs \times t}{max(C_l)} \;\; for\, t \;\in\; C_l \right] \tag{7}$$

$$r = \frac{\sum Z}{(wt - ws) \times L}, Z = \left[\frac{k}{r} \;\; for\; k \;\in\; Z \right] \tag{8}$$

The KV cache is further updated layer by layer based on this normalized budget, progressively refining the retained information to align with the overall compression strategy. The above process can be expressed as Algorithm 1.

5 Experiments

We conduct comprehensive comparative and ablation experiments to verify the effectiveness of our DynamicKV. In Section 5.1, we introduce the models, datasets and baselines used in our experiments. Section 5.2 provides a performance comparison between DynamicKV and baseline approaches. Next, in Section 5.3, we present the results of DynamicKV on the Needle in Haystack Task. Finally, in Section 5.4, we conduct an ablation study on the parameters of our method to validate its feasibility.

5.1 Implementation details

Models and Context Length. We utilize the official checkpoints of recently released models from huggingface including LlaMA-3-8B-Instruct (Dubey et al., 2024), Qwen-2-7B-Instruct (Yang et al., 2024), Mistral-7B-Instruct-v0.2 (Jiang et al., 2023), and InternLM-2.5-7B-Chat-1M (Cai et al., 2024) as our base models, which support context lengths of 8k, 32k, 32k, and 1M tokens respectively.

Datasets. LongBench is a comprehensive benchmark for evaluating the contextual understanding capabilities of LLMs. For our comparative experiments, we use 16 English datasets from this benchmark, specifically NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al., 2021), MultiFieldQA-en, HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019), TREC (Li and Roth, 2002), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019), PassageCount, PassageRetrieval-en, LCC (Guo et al., 2023), and RepoBench-P (Liu et al., 2023). These cover key long context application scenarios such as *Single-Document QA*, *Multi-Document QA*, *Summarization*, *Few-shot Learning*, *Synthetic Tasks*, and *Code Completion*. Additionally, for the experiment on the Needle in Haystack task, we test the models across their maximum length ranges [8k, 32k, 1M] using the PaulGrahamEssays dataset.

Baselines. We evaluate the recent fixed-pattern token-dropping methods, including: (1) **StreamingLLM**, which utilizes attention sinks and rolling KV caches to retain the most recent tokens. (2) **H2O**, which employs a Heavy Hitter Oracle for KV cache eviction. (3) **SnapKV**, which selects important tokens for each attention head through clustering. (4) **PyramidKV**, which introduces a pyramid pattern where layers select important tokens in a monotonically decreasing manner.

5.2 Comparative experiments on LongBench

With the total KV cache size fixed at 128 and 512, we compare the performance retention of StreamingLLM, H2O, SnapKV, PyramidKV, and our proposed method, DynamicKV, relative to FullKV. As shown in Table 1, DynamicKV demonstrates stable improvements even while maintaining an extremely low KV cache size relative to the total context (128: 1.7%; 512: 6.9%). Specifically, with the cache size of 128, DynamicKV outperforms the best alternative by 0.3%, 0.97%, 1.68%, and 0.79% on LLaMA, Mistral, Qwen, and InternLM, respectively, retaining 90%, 87%, 78%, and 83% of the overall performance. Moreover, with a cache size of 512, DynamicKV surpasses the highest-performing method by 0.43%, 0.19%, 0.69%, and 0.53% on the same models, retaining 97%, 96%, 96%, and 89% of FullKV's performance. The data in the table clearly demonstrate DynamicKV's effectiveness under extreme compression, achieving nearly FullKV-level performance with just 6.9% of the cache size. The experimental results show that DynamicKV can improve the effect of complex tasks such as *code completion* more obviously based on maintaining PyramidKV performance, and greatly improve the performance upper limit of lower KV cache size.

5.3 Visualization on Needle-in-Haystack Task

The needle-in-a-haystack test involves inserting key information at random positions within a long context and setting answers to evaluate whether LLMs can accurately detect critical information in extensive contexts. To further illustrate the effectiveness of our approach in compressing the KV cache, we conduct additional experiments using Mistral on the needle-in-a-haystack task, focusing on maintaining an optimal size for the KV cache.

As shown in Figure 4, we insert information at various positions in the Paul Graham Essays dataset and extract answers by prompting the model to generate responses. The green blocks indicate that the response matches the contents of the needle, but the colour change from yellow to red indicates that the response is more irrelevant to the needle.

We test a fixed KV cache size of 64 using FullKV, StreamingLLM, H2O, SnapKV, PyramidKV, and the DynamicKV method. The results indicate that DynamicKV maintains 90% of the model's performance even under extreme compression, improving accuracy by 57%, 37%, 41%, and 11% compared to the other methods, respectively. Additionally, the figure shows that with a context length of up to 7000, the extreme compression of DynamicKV nearly achieves full scores, and even beyond 7000, it shows significant improvements compared to other approaches. This finding illustrates that DynamicKV has a distinct advantage in hierarchical token selection and confirms that the number of critical tokens contained at different layers is always dynamic.

5.4 Ablation Study

In this study, we investigate the performance of the DynamicKV mechanism across varying key-value cache sizes. The results, as shown in Table 2, reveal a consistent improvement in performance with an increase in the cache size for all evaluated models. For the LlaMA-3-8B-Instruct, the perfor-

Model	Size	Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		
			NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	Avg.
			18409	3619	4559	9151	4887	11214	8734	10614	2113	5177	8209	6258	11141	9289	1235	4206	–
LlaMA-3-8B -Instruct	–	FullKV	25.16	31.81	39.59	43.09	36.15	21.77	28.62	23.34	26.33	75.00	90.50	42.36	5.20	69.25	59.04	53.93	41.95
	128	StreamingLLM	17.85	9.50	23.09	37.84	29.02	16.77	17.91	20.42	20.16	44.00	73.00	30.00	5.80	69.50	48.38	49.31	32.03
		H2O	21.58	12.54	28.49	37.13	32.36	18.88	20.23	22.16	21.14	39.00	86.62	39.19	5.50	69.50	57.39	54.46	35.39
		SnapKV	21.71	12.37	32.38	37.44	30.48	19.50	19.06	21.36	20.07	45.5	87.74	38.15	5.50	68.85	57.42	54.61	35.76
		PyramidKV	22.26	16.65	30.73	38.97	29.28	19.19	19.92	22.06	20.87	68.00	88.95	38.23	5.92	69.50	57.20	51.54	37.45
		ours	22.10	14.93	32.94	41.06	27.98	21.18	20.03	22.06	21.28	65.50	89.61	38.70	5.13	69.50	58.01	54.00	**37.75**
	512	StreamingLLM	19.03	12.78	28.67	37.83	29.97	16.55	20.30	20.94	24.56	61.00	75.43	30.82	5.86	69.50	51.93	49.98	34.70
		H2O	22.84	16.80	32.36	41.43	34.07	19.30	22.28	22.81	23.69	41.00	90.46	40.19	5.54	69.50	57.52	55.43	37.20
		SnapKV	24.62	22.78	37.88	42.96	34.82	20.65	22.63	22.54	23.93	70.00	90.39	40.30	5.74	69.50	60.27	55.85	40.30
		PyramidKV	24.48	23.51	36.14	42.33	31.95	20.73	23.37	23.01	24.37	72.50	90.43	40.54	5.88	69.50	59.25	54.87	40.18
		ours	24.78	24.76	36.84	44.13	33.25	20.82	23.00	22.76	24.14	72.50	90.39	40.76	5.78	69.50	61.40	56.91	**40.73**
Mistral-7B -Instruct-v0.2	–	FullKV	26.63	32.99	49.34	42.77	27.35	18.77	32.87	24.24	27.10	71.00	86.23	42.96	2.75	86.98	56.93	54.49	42.71
	128	StreamingLLM	16.58	14.76	30.36	28.13	21.76	11.98	18.26	19.02	19.16	43.50	74.12	28.50	2.50	31.81	43.65	41.19	27.83
		H2O	21.66	21.64	38.60	30.96	20.63	13.02	20.65	22.61	22.08	39.00	82.19	39.75	3.16	79.98	51.25	48.20	34.71
		SnapKV	20.11	21.28	42.98	37.51	22.31	14.43	19.19	21.89	21.01	48.00	83.77	40.44	2.51	66.99	51.64	48.57	35.16
		PyramidKV	22.11	22.52	43.04	33.57	22.98	15.69	20.56	22.52	21.36	65.50	83.84	40.03	2.89	67.26	51.51	46.42	36.36
		ours	22.05	23.65	43.08	36.03	22.60	15.23	21.35	23.11	22.19	68.00	84.79	41.02	4.20	70.11	52.45	47.41	**37.33**
	512	StreamingLLM	19.05	17.21	36.82	30.64	21.84	10.56	24.47	19.84	25.48	62.00	72.82	29.49	2.71	19.25	46.15	42.55	30.06
		H2O	22.33	25.75	44.09	32.76	22.88	14.96	23.53	22.96	24.53	41.50	85.53	41.54	3.39	86.20	55.11	50.81	37.37
		SnapKV	24.95	27.97	49.04	39.93	25.18	17.64	24.14	23.69	24.47	67.50	86.04	41.14	2.90	86.98	56.73	53.11	40.71
		PyramidKV	23.49	28.79	48.71	41.00	25.64	16.35	24.79	23.52	24.49	69.50	86.20	42.58	3.53	81.81	55.45	51.67	40.47
		ours	25.63	29.11	48.41	39.85	26.62	16.72	24.73	23.72	24.83	70.50	86.74	43.01	3.20	83.57	55.40	52.35	**40.90**
Qwen2-7B -Instruct	–	FullKV	25.14	42.35	45.04	14.80	14.13	9.23	36.35	23.79	26.51	76.50	89.16	45.23	6.50	75.50	60.30	60.78	40.71
	128	StreamingLLM	19.25	23.63	26.51	14.00	15.30	7.46	18.07	19.30	18.30	47.00	77.92	31.57	6.50	17.00	42.52	41.94	26.64
		H2O	20.33	30.43	34.22	13.61	13.37	7.81	20.72	21.66	18.44	40.00	86.94	42.17	7.00	70.50	53.45	53.76	33.40
		SnapKV	22.26	31.62	38.95	16.05	17.71	7.66	18.91	21.41	18.21	46.00	87.61	42.01	6.50	63.50	54.87	53.03	34.14
		PyramidKV	20.50	31.70	39.95	18.54	18.54	8.85	19.24	20.47	18.18	60.00	87.98	39.71	7.00	49.00	48.77	47.91	33.52
		ours	22.77	35.57	42.62	14.80	16.35	8.31	21.41	21.97	19.56	58.00	88.18	40.93	6.50	70.00	53.58	52.50	**35.82**
	512	StreamingLLM	20.47	26.97	32.64	14.31	14.39	6.82	25.70	19.31	24.88	66.00	76.56	32.11	8.00	15.50	46.58	44.20	29.65
		H2O	22.88	34.28	41.40	13.30	14.60	8.31	23.69	22.07	22.72	39.50	88.75	43.91	6.00	72.00	58.83	57.83	35.63
		SnapKV	23.86	38.61	44.65	15.60	14.62	9.13	24.56	22.39	23.07	70.00	89.31	43.32	5.00	72.00	58.67	60.74	38.47
		PyramidKV	24.47	37.60	43.51	14.48	12.83	8.99	23.59	22.30	22.41	74.00	89.21	43.40	6.50	74.00	57.67	56.14	38.19
		ours	24.66	40.44	45.30	15.42	13.89	8.46	25.51	22.77	22.92	74.00	89.27	43.18	7.00	74.00	60.38	59.33	**39.16**
InternLM-2.5-7B -Chat-1M	–	FullKV	22.42	27.61	39.98	40.92	33.48	26.68	33.01	25.18	26.28	72.50	86.76	39.76	2.91	100.00	55.86	57.95	43.21
	128	StreamingLLM	17.91	13.02	24.31	24.27	16.01	11.29	17.29	20.62	18.06	48.5	67.53	21.93	0.82	87.39	43.45	42.79	29.70
		H2O	16.16	17.71	27.94	26.83	17.83	17.81	13.99	22.59	16.9	39.50	81.87	32.15	1.32	96.50	48.30	47.27	32.79
		SnapKV	19.65	17.44	35.29	27.36	18.58	19.79	12.76	22.42	16.31	48.00	80.23	31.35	0.95	95.00	49.47	48.22	33.93
		PyramidKV	18.80	17.35	33.48	31.16	20.05	19.02	14.65	22.02	17.40	69.50	80.87	32.02	1.23	95.00	47.13	44.73	35.28
		ours	17.93	19.89	34.15	31.50	19.03	20.60	15.14	22.41	18.15	70.00	83.09	32.44	0.86	95.50	49.33	47.16	**36.07**
	512	StreamingLLM	17.58	15.86	26.55	26.68	16.69	11.01	25.96	21.33	25.57	65.00	67.16	21.71	0.95	87.56	43.58	42.76	32.25
		H2O	15.33	19.84	32.41	27.88	20.10	21.13	16.91	22.99	21.49	41.00	84.38	34.76	1.23	96.50	48.46	50.00	34.65
		SnapKV	16.86	23.28	36.24	32.14	19.89	23.21	17.69	23.18	22.44	71.00	84.05	34.34	1.00	96.50	50.32	53.34	37.84
		PyramidKV	17.62	21.08	37.52	32.21	21.31	22.03	19.37	24.06	22.22	73.00	83.94	34.61	1.05	95.50	50.45	49.72	37.86
		ours	17.77	23.87	37.74	32.98	21.13	20.85	19.13	23.49	22.48	75.00	84.89	36.70	0.91	95.50	50.70	51.08	**38.39**

Table 1: **Performance comparison on the LongBench dataset** for full KV cache, previous methods (StreamingLLM, H2O, SnapKV, PyramidKV), and our DynamicKV method, with KV cache sizes of 128 and 512, using models including LLaMA3-8B-Instruct, Mistral-7B-Instruct-v0.2, QWen2-7B-Instruct, and InternLM-2.5-Chat-1M. Bold indicates the best performance.

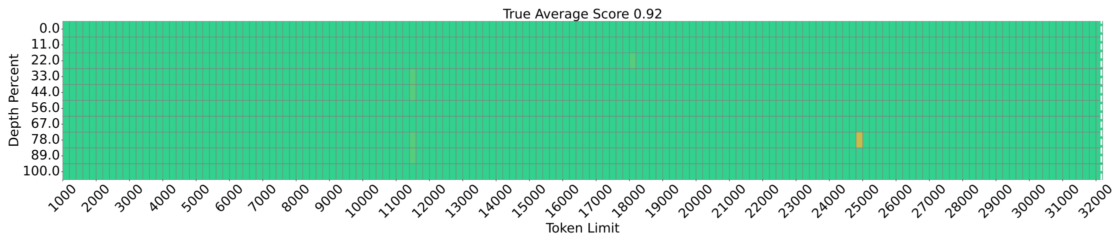
KV size	LlaMA-3-8B-Instruct	Mistral-7B-Instruct-v0.2	Qwen2-7B-Instruct	InternLM2.5-7B-Chat-1M
64	34.93	33.95	32.67	33.67
96	36.70	36.22	34.85	35.31
128	37.75	37.33	35.82	36.07
256	39.83	39.23	36.98	37.29
512	40.73	40.90	39.16	38.39
1024	41.22	41.48	39.72	38.86

Table 2: **Performance of DynamicKV** with **different KV cache size**.

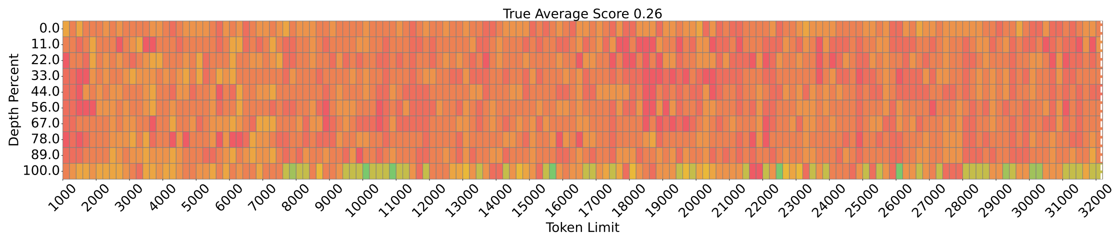
mance metric improved from 34.93 to 41.22 as the key-value cache size was increased from 64 to 1024. This improvement is also applicable to other models. These findings underscore the effectiveness of the DynamicKV cache in leveraging KV cache compression to maintain the capabilities of long context. Notably, a larger cache capacity is generally associated with superior performance. Nonetheless, it is essential to strike a balance when selecting the cache size, taking into account the practical constraints related to storage and computational resources.

☞ More analyses in Appendix

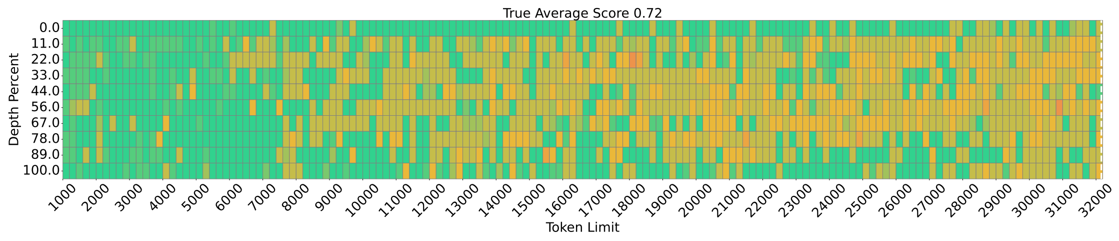
In addition to the above discussions, we conduct more related analyses and show them in Appendix: (1) DynamicKV is effectively scalable to other models on the Needle-in-Haystack task (Appendix A.3); (2) Enhancing generation efficiency



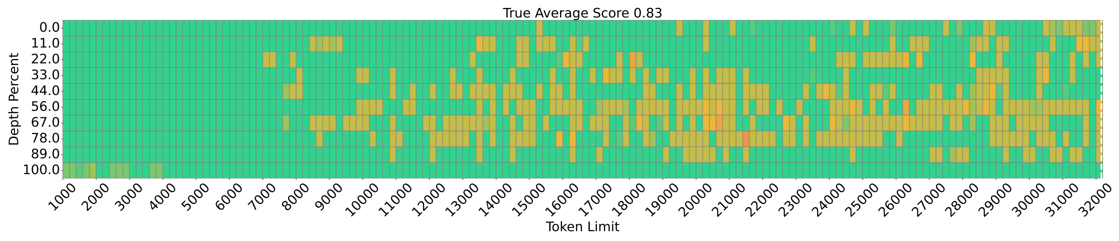
(a) FullKV



(b) StreamingLLM



(c) PyramidKV



(d) DynamicKV

Figure 4: **Performance Comparison on the Needle in a Haystack Task** Using Mistral-7B-Instruct-v0.2.

with 22.5-129% higher token throughput compared to fullKV and superior memory optimization saving up to 11.2% GPU memory while maintaining stable performance (Appendix A.4); (3) Task-Aware adaptability that dynamically adjusts token processing across transformer layers based on input complexity(Appendix A.5). Please refer to the Appendix for more details.

6 Conclusion

In this study, we analyze the intrinsic patterns exhibited by large language models (LLMs) when processing long-context inputs across different task types. Our empirical findings reveal significant variations in the distribution of attention across these task types. Based on this observation, we introduce DynamicKV, a novel layer-aware KV cache compression approach that dynamically adjusts the KV cache size across layers. We evaluate the effectiveness and generalizability of DynamicKV through experiments on 16 datasets from the LongBench benchmark, demonstrating its broad applicability and performance benefits. From the results, we mainly conclude that: (1) a wave-like pattern is followed in complex reasoning tasks (e.g., *code completion* tasks); (2) a pyramid-like pattern is followed in *Synthetic* and *Summarization* tasks; (3) The dynamic hierarchical adaptive DynamicKV approach is capable of formulating a relatively appropriate KV cache retention strategy in accordance with diverse tasks. Particularly, in the circumstance of maintaining an extremely small KV cache size, the effect is significantly enhanced.; In the future, we hope that there is a more suitable method to perform KV cache compression without increasing the computation.

Limitations

Our work has several potential limitations. First, given the limited computational budget, we only validate our DynamicKV on models Scaling up to super-large model sizes (e.g., 70B), and applying DynamicKV to more cutting-edge model architectures will be more convincing model architectures. Second, although we have conducted experiments on multiple tasks including single- and multi-document QA, summarization, synthetic tasks, and code completion, the generalization ability of DynamicKV to other tasks or datasets has not been fully explored. Future work will focus on expanding the application scope of DynamicKV to more diverse tasks and datasets.

Ethics and Reproducibility Statements

Ethics We take ethical considerations seriously and follow the guidelines outlined by the ACL Ethics Policy. The DynamicKV method is designed to optimize long-context inference in LLMs, without the need for collecting sensitive or private information. All datasets used in the experiments are publicly available and widely adopted by the research community, ensuring transparency and accessibility. We do not foresee any significant ethical concerns related to the development and use of the DynamicKV method.

Reproducibility To ensure reproducibility, we provide detailed descriptions of our experimental setup, including model configurations, datasets, and performance metrics. Furthermore, **we have provided our code in the Supplementary Material**. We hope that the provided resources will support further advancements in efficient LLM inference and memory management.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. 2024. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Optimizing kv cache eviction in llms: Adaptive allocation for enhanced budget utilization. *arXiv preprint arXiv:2407.11550*.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does bert learn about the structure of language? In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

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

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. Minicache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024b. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024c. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Qingyu Lu, Baopu Qiu, Liang Ding, Kanjian Zhang, Tom Kocmi, and Dacheng Tao. 2024. Error analysis prompting enables human-like translation evaluation in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 8801–8816, Bangkok, Thailand. Association for Computational Linguistics.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. 2024. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*.

Keqin Peng, Liang Ding, Qihuang Zhong, Li Shen, Xuebo Liu, Min Zhang, Yuanxin Ouyang, and Dacheng Tao. 2023. Towards making the most of chatgpt for machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5622–5633.

Alec Radford. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Daniel Waddington, Juan Colmenares, Jilong Kuang, and Fengguang Song. 2013. Kv-cache: A scalable high-performance web-object cache for manycore. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 123–130. IEEE.

Shuai Wang, Liang Ding, Li Shen, Yong Luo, Zheng He, Wei Yu, and Dacheng Tao. 2024. Improving code generation of llms by uncertainty-aware selective contrastive decoding. *arXiv preprint arXiv:2409.05923*.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, et al. 2024a. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2023. Can chatgpt understand too? a comparative study on chatgpt and fine-tuned bert. *arXiv preprint arXiv:2302.10198*.

A Appendix

A.1 Model Details

All the model structures and details in our experiment are shown in Table 3.

A.2 Dataset Details

The data sources, average length, evaluation metrics, language, and data volume of the LongBench(Bai et al., 2023) dataset's subdatasets are shown in Table 4.

A.3 Need in a HayStack

The experimental results are presented in Table 5, which illustrates the performance of our DynamicKV approach across various models, including LlaMA and Qwen, in Needle in a Haystack task.

A.4 Efficiency Experiments

We evaluate the efficiency of DynamicKV against the standard method (FullKV) under varying input/output lengths. All experiments are conducted with a fixed context window ($m = 128$), measuring Time-to-First-Token (TTFT), Time-Per-Output-Token (TPOT), end-to-end latency, and GPU memory usage. The results are summarized in Table 6.

Key observations include:

- **Short Sequences (8k/2k):** DynamicKV improves TPOT by 22.5% (27.63→33.85 tok/s) while slightly increasing TTFT by 6% (0.66s→0.70s), achieving 18.2% lower total latency (74.79s→61.21s) with 638MB memory reduction.
- **Long Sequences (32k/8k):** The advantages amplify significantly, with DynamicKV delivering 129% higher TPOT (11.65→26.69 tok/s), 56% lower latency (706.56s→310.56s), and 11.2% memory savings (31213MB→27713MB).
- **Scalability:** FullKV shows superlinear TPOT degradation (11.65 tok/s at 32k inputs), while DynamicKV maintains stable throughput through on-demand computation, demonstrating better adaptability to long-context generation.

The experiments demonstrate that dynamic KV caching trades marginal initial latency for substantially better sustained generation speed and memory efficiency, particularly beneficial for long-text generation tasks (>2k output tokens).

A.5 Task-Aware Sensitivity Study

The presented Table 7 offers a comprehensive insight into the token processing statistics per layer for 32 transformer layers using the DynamicKV method in Mistral-7B-Instruct, as applied to various datasets from LongBench. This detailed analysis reveals significant variability in how tokens are processed at different layers depending on the specific task.

In particular, our approach demonstrates that certain tasks, such as those represented by the Multi-News and GovReport datasets, exhibit exceptionally high token processing counts even in early layers, indicating a potentially greater complexity or information density in these tasks. Conversely, datasets such as TriviaQA and SAMSum show relatively lower initial processing demands, but with significant spikes in later layers, suggesting evolving complexity throughout the document or dialogue.

Moreover, the data highlights that the DynamicKV mechanism adeptly adjusts to the unique demands of each dataset, optimizing performance and resource allocation dynamically across all layers. This adaptability is crucial for handling the diverse range of tasks and input lengths encountered in real-world applications, ensuring efficient processing without compromising on understanding or accuracy.

Configuration	**LlaMA-3-8B-Instruct**	**Mistral-7B-Instruct-v0.2**	**Qwen2-7B-Instruct**	**InternLM2.5-7B-Chat-1M**
Hidden Size	4,096	4,096	3,584	4096
# Layers	32	32	28	32
# Query Heads	32	32	28	32
# KV Heads	8	8	4	8
Head Size	128	128	128	128
Intermediate Size	14,336	14,336	18,944	14336
Embedding	False	False	False	False
Vocabulary Size	128,256	32,000	151,646	92,544

Table 3: Configuration of Models.

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

Table 4: An overview of the dataset statistics in LongBench.

Model	**StreamingLLM**	**H2O**	**SnapKV**	**PyramidKV**	**DynamicKV**
LlaMA-3-8B-Instruct	0.29	0.46	0.80	0.89	0.9
Qwen-2-7B-Instruct	0.22	0.41	0.84	0.86	0.87

Table 5: Configuration of Models.

Input Len	**Output Len**	**Method**	**TTFT (s)**	**TPOT (tok/s)**	**Latency (s)**	**Memory (MB)**
8k	2k	FullKV	0.66	27.63	74.79	20055
8k	2k	Dynamickv	0.70	33.85	61.21	19417
16k	4k	FullKV	1.45	19.55	209.56	23859
16k	4k	Dynamickv	1.49	33.02	125.52	22051
32k	8k	FullKV	3.52	11.65	706.56	31213
32k	8k	Dynamickv	3.58	26.69	310.56	27713

Table 6: Efficiency Comparison Between FullKV and DynamicKV

Dataset	**Layer-wise (1 - 32) Tokens**															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
NarrativeQA	93	75	111	51	71	75	91	108	97	154	138	143	196	126	156	201
	173	133	202	200	158	128	115	86	123	109	117	76	122	120	142	206
Qasper	157	66	92	43	59	70	77	111	93	137	132	160	187	122	147	186
	184	132	183	181	160	135	122	77	119	117	128	69	127	113	179	231
MultifieldQA-en	245	79	109	50	76	75	92	106	90	169	138	164	205	115	155	196
	174	116	167	169	145	116	103	74	97	90	108	67	116	116	154	220
HotpotQA	145	75	115	55	73	73	92	102	88	146	128	140	190	106	140	208
	172	129	194	185	146	123	109	80	105	99	125	75	127	122	173	256
2WikiMQA	198	74	110	53	69	74	91	105	89	159	133	153	196	109	143	215
	178	132	181	179	139	112	96	76	102	94	120	70	118	117	161	250
MuSiQue	108	71	108	54	72	71	89	98	85	143	124	135	189	104	138	216
	178	128	206	198	148	130	113	84	112	100	126	76	134	128	173	257
GovReport	332	98	113	48	73	78	88	136	125	150	176	145	142	113	136	164
	212	129	148	161	145	115	79	62	103	78	94	57	99	121	138	238
QMSum	156	85	119	51	65	63	78	116	81	146	126	136	185	105	142	218
	187	121	204	201	165	134	121	85	122	105	122	70	115	121	144	207
MultiNews	459	94	115	50	67	71	88	101	99	120	126	147	168	130	156	190
	202	125	147	160	134	107	78	60	101	84	95	55	102	117	146	202
TREC	214	70	112	47	69	97	93	129	98	156	142	215	220	141	213	115
	143	110	166	103	105	91	90	67	84	89	139	78	129	96	192	283
TriviaQA	105	62	97	52	73	89	86	106	102	178	139	178	198	140	189	199
	168	118	159	139	144	103	90	66	109	102	131	71	127	124	196	256
SAMSum	126	64	103	48	71	96	93	152	118	157	130	174	173	150	189	179
	168	132	177	135	135	108	98	84	101	104	120	87	114	114	168	228
PassageCount	91	72	106	48	54	58	80	111	96	119	153	143	166	147	157	225
	214	147	191	203	178	106	96	72	116	116	118	70	122	130	148	243
PassageRetrieval-en	152	92	129	70	72	75	92	121	108	128	162	149	142	139	145	175
	179	145	180	172	134	113	95	87	112	109	124	91	108	118	140	238
LCC	186	59	104	44	64	80	98	128	112	159	169	161	177	173	162	163
	175	133	170	147	113	102	83	69	94	80	114	84	138	130	178	247
RepoBench-P	106	63	106	46	66	89	96	132	119	168	177	169	179	183	176	189
	197	138	177	148	113	102	80	71	94	80	106	83	132	133	157	221

Table 7: Per-layer token processing statistics across 32 transformer layers by DynamicKV in Mistral-7B-Instruct-v0.2 model on LongBench datasets.

Algorithm 1 DynamicKV in Prefill Phase

```
1: Input: initial budget K/V cache list $K^b$, $V^b$, radio max $r_{max}$, update interval $m$, mean token length $wt$, window size $ws$, sequence length $S$, head dimention $hd$, input embedding of window size $X^{ws} \in \mathbb{R}^{ws*d}$, initial budget Attention list computed by window token and others $A^b$,
2: Output: Compressed K/V cache $K^c$, $V^c$
3: $bs = (wt - ws) \times r_{max}$
4: def Update_Buffer_Length($A$, $l$):
5:     $A^{gather} \leftarrow$ cat(([$A$ for $l$ in (1, $l$)]), 0).view(-1)
6:     $cnts \leftarrow$ Count_Elemnets(topk($A^{gather}$, k=$(wt - ws) * H * l$).indices / ($L * S$)) / $l$
7:     Compute the $norm$ of $cnts$, range in (0, 1)
8:     $BL \leftarrow$ [int(($bs * t$ / max($norm$))) for $t$ in $norm$]
9:     $r \leftarrow$ sum($BL$) / (($wt - ws$)$*L$)
10:    $BL \leftarrow$ [int($k/r$) for $k$ in $BL$]
11:    Return $BL$
12: for $l \leftarrow 1$ to $L$ do
13:   Compute full KV states $K^s$, $V^s$
14:   for $h \leftarrow 1$ to $H$ do
15:     /* compute the Attention between window size token and other all token */
16:     $A_{l,h} \leftarrow$ softmax(($X^{ws} W_h^Q$) $\cdot$ $K_h^T$).mean(dim=-2).pooling(dim=-1)
17:   end for
18:   Append $A_l$ to $A^b$ /* current $A_l$ shape is [$H$, $S$] */
19:   /* calculate current layer buffer KV cache */
20:   indices $\leftarrow$ $A_l$.topk($bs$, dim=-1).indices.unsqueeze(-1).expand(-1, -1, $hd$)
21:   $K_l^b \leftarrow$ cat(($K^s$[:,:$-ws$,:].gather(dim=-2, indices),$K^s$[:,$-ws$:,:]), dim=-2)
22:   $V_l^b \leftarrow$ cat(($V^s$[:,:$-ws$,:].gather(dim=-2, indices),$V^s$[:,$-ws$:,:]), dim=-2)
23:   /* gradually compress*/
24:   if $l$ % $m$ == 0 then
25:     $Bl \leftarrow$ Update_Buffer_Length($A_l$, $l$)
26:     /* update the buffer K/V Cache*/
27:     for $i \leftarrow 1$ to $l$ do
28:       $K_i^b \leftarrow$ cat(($K_l^b$[:,:$Bl_i$,:], $K_l^b$[:,$-ws$:,:]), dim=-2)
29:       $V_i^b \leftarrow$ cat(($V_l^b$[:,:$Bl_i$,:], $V_l^b$[:,$-ws$:,:]), dim=-2)
30:     end for
31:   end if
32: end for
33: Update the K/V Cache $K^c, V^c$ from $K^b, V^b$
```