Task-Aware Dynamic KV Cache Eviction via Error-Driven Importance Estimation for Efficient LLM Inference

Anonymous ACL submission

Abstract

The increasing context length in Large Language Models (LLMs) introduces significant memory overhead due to the rapid growth of the Key-Value (KV) cache. Recent works have explored KV cache eviction methods for KV cache reduction in LLMs. However, these methods typically rely on simplistic calculations for KV cache importance and apply rigid cache allocation strategies across layers, resulting in notable performance degradation. To address these limitations, we propose an Error-Driven Importance Estimation (EDIE) method that rigorously quantifies token criticality based on the output error of the attention, and build upon it a Task-Aware Dynamic Allocation (TADA) mechanism that optimizes the layer-specific allocation of KV cache capacity based on the task complexity and layer importance. Experiments show consistent accuracy gains on LongBench tasks, surpassing prior methods across cache budgets. Notably, on the Needle-in-a-Haystack task, our method achieves up to 15.7% absolute accuracy gain under extreme cache constraints.

1 Introduction

Large Language Models (LLMs) have achieved significant success and have been widely applied across various domains, such as text generation (Wu, 2024), machine translation (Zhao et al., 2024), and code generation (Wang and Chen, 2023). The context size of LLMs represents the length of context they can process, which is crucial for various real-world applications, such as long-document summarization and multi-turn conversations. Consequently, the context size of LLMs has been continuously increasing, expanding from 2048 tokens in models like GPT-3 (Dettmers et al., 2022) and LLaMA1 (Touvron et al., 2023) to 128K tokens in LLaMA3.1 (Grattafiori et al., 2024).

LLMs generate text sequentially (Vaswani et al., 2017). Modern LLMs utilize the KV cache (Shi et al., 2024), where key and value vectors from previously processed tokens are stored and reused during inference. However, the KV cache is proportional to the context length, growing rapidly as the context length increases. This leads to substantial requirements for GPU memory and bandwidth, which can negatively impact the inference efficiency of LLMs.

An approach to addressing the excessive size of the KV cache is the KV cache eviction method. This method reduces the KV cache size by pruning the key-value pairs of less important tokens and retaining only a limited number of key-value pairs of important tokens in an attention layer. The KV cache eviction method encompasses two interdependent research components: (1) determining the importance criteria of tokens; (2) deciding the allocation scheme of KV cache capacity.

Recent works (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024; Cai et al., 2024; Yang et al., 2024; Feng et al., 2024; Guo et al., 2024a) have focused on identifying which tokens should be retained in the attention layer to improve efficiency. These methods often prioritize tokens based on heuristics, such as their position at the beginning of the sequence, their presence within the recent decoding window, large accumulated attention scores (AAS) or large products of the accumulated attention scores and value norms (PAASVN). Although these metrics offer intuitive insights, they typically consider only one aspect of token importance, such as position, attention score, or value norm, and thereby undermine the combined effects of these elements. We argue that assessing token importance through the attention output error introduced by token removal provides a more systematic and holistic evaluation.

With regard to allocation schemes, several works (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024; Cai et al., 2024; Yang et al., 2024; Feng et al., 2024) have been proposed to determine how many pairs

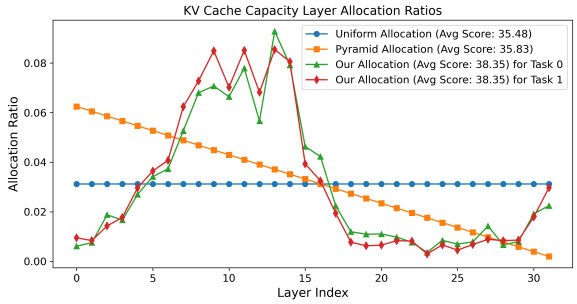


Figure 1: KV cache capacity layer allocation methods. In contrast to uniform and pyramid allocation, our allocation is capable of adapting the layer allocation ratio according to task differences, thereby achieving higher average score across 16 tasks in the LongBench benchmark.

of key-value should be retained in different layers. These methods typically adopt a fixed allocation strategy that distributes the KV capacity either uniformly or in a pyramid-shaped manner across layers. However, such approaches lack flexibility and fail to account for the distinct roles and computational characteristics of different layers. Since each layer processes information in a unique manner, applying such allocation schemes across all layers is an oversimplification that could hinder the performance. In addition to layer-wise differences, KV caches are typically maintained independently for each attention head, suggesting that allocation decisions should also be made at the head level rather than uniformly across the model. Moreover, the nature of the task should guide the retention of KV pairs, as assuming all tasks require the same memory budget for each layer in LLM can lead to inefficient or suboptimal cache utilization.

To address these limitations, we propose a task-aware dynamic allocation strategy that adaptively distributes KV cache capacity along three dimensions—layer, head, and task—guided by importance scores estimated from token removal error. An illustration of our strategy is shown in Figure 1, our allocation strategy yields allocation patterns that differ from those of uniform and pyramid allocations. It adapts its allocation based on the task nature and layer error, demonstrating high flexibility and performance.

In summary, we propose: (1) A systematic importance estimation method derived from the perspective of minimizing attention output error, which precisely characterizes the importance of tokens; (2) A task-aware dynamic allocation strategy for KV cache capacity that builds upon EDIE, adapting to different tasks, attention layers, and attention heads.

We conducted extensive experiments on the LLaMA3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 models, evaluating our method in 16 long-context tasks from the LongBench and Needle-In-A-Haystack benchmarks. By accurately estimating the importance of tokens and dynamically allocating KV cache capacity, our method achieves state-of-the-art results. Code available soon.

2 Related Works

Recent studies have investigated methods for assessing the importance of KV cache entries and pruning less critical key-value pairs of tokens in LLMs. H_2O (Zhang et al., 2023) employ the accumulated attention scores to estimate the importance of the tokens and dynamically retains a combination of recent and important tokens in the KV cache. StreamingLLM (Xiao et al., 2024) addresses long-context inference bottlenecks by preserving both recent tokens and initial attention sinks in the KV cache. SnapKV (Li et al., 2024) introduces a KV cache compression method that identifies critical tokens through the accumulated attention scores of the observation tokens. VATP (Guo et al., 2024a) combines attention scores and value vector norms to estimate the importance of the tokens. These methods lack rigorous theoretical derivation and cannot identify important keys and values well, leading to significant performance degradation.

Other recent studies have investigated methods for determining the KV cache capacity. PyramidKV (Cai et al., 2024) and PyramidInfer (Yang et al., 2024) introduce a pyramid-shaped KV cache compression method that dynamically allocates larger cache budgets to lower layers (for dispersed information) and smaller ones to higher layers. AdaKV (Feng et al., 2024) dynamically allocates the KV cache capacity of different attention heads in the same layer based on their mean attention scores. DynamicKV (Zhou et al., 2024) dynamically adjusts the number of retained tokens per layer based on attention scores for each task. (He et al., 2025) allocate cache based on semantic differentiation of attention heads. These methods either rely on predefined KV cache allocation across layers or dynamic KV cache allocation based on attention scores , rather than using error-guided strategy to assess the impact of token removal across layers, tasks, and heads.

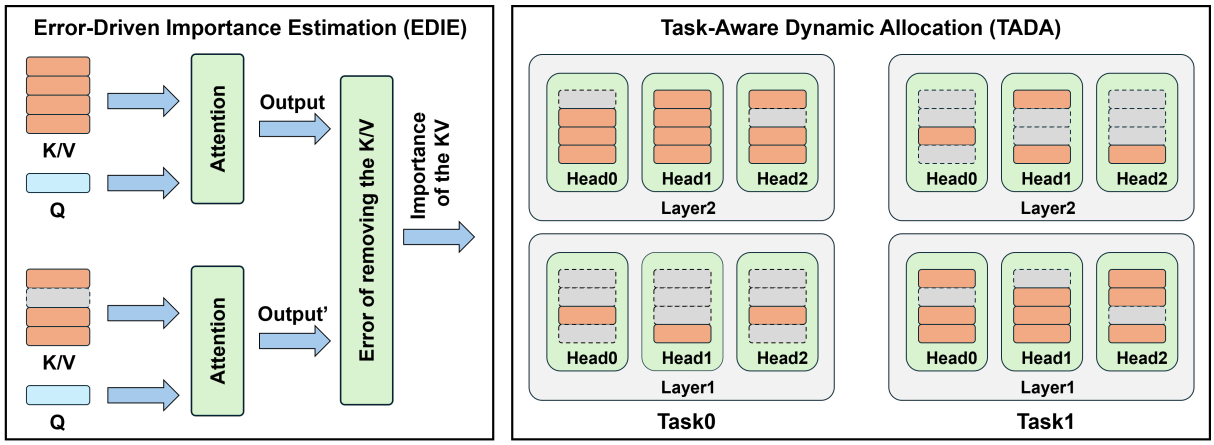


Figure 2: Method overview. Left side shows our error-driven importance estimation method that rigorously quantifies token importance based on the output error of the attention. Right side shows our task-aware dynamic allocation strategy for KV cache capacity that adaptively distributes KV cache capacity along layer, head, and task dimensions.

3 Method

3.1 Preliminary

The output of an attention head at position j is defined as follows:

$$a_i^j = \frac{exp(\boldsymbol{q}_j \boldsymbol{k}_i^T / \sqrt{d_h})}{\sum_i exp(\boldsymbol{q}_j \boldsymbol{k}_i^T / \sqrt{d_h})} \tag{1}$$

$$o_j = \sum_i a_i^j \boldsymbol{v}_i \tag{2}$$

where $\boldsymbol{q}_j$ is the query vector of token j, $\boldsymbol{k}_i$ is the key vector of token i, d_h is the attention head dimension, a_i^j is the attention score between query token with index j and previous token with index i, and $\boldsymbol{v}_i$ is the value vector of token in index i. Traditional methods typically measure importance I based on attention scores (Zhang et al., 2023) or a combination of attention scores and value states (Guo et al., 2024b) using the following equations:

$$I_i^{AAS} = \sum_{t-w<j\leq t} a_i^j \tag{3}$$

$$I_i^{PAASVN} = \sum_{t-w<j\leq t} a_i^j \left\|\boldsymbol{v}_i\right\|_1 \tag{4}$$

In Equation (4), w represents the window size for the observation window, and t represents the latest token in the input sequence.

3.2 Error-Driven Importance Estimation

An overview of the importance estimation is shown on the left of Figure 2. Here we derive the importance of tokens from the error of the attention output step-by-step. We define the importance of a token as the error of the attention output introduced by its removal. If the key-value pair of m-th token is removed, the new attention output $o'_j(m)$ thus become:

$$\hat{a}_i^j(m) = \frac{exp(\boldsymbol{q}_j \boldsymbol{k}_m^T / \sqrt{d_h})}{\sum_{i\neq m} exp(\boldsymbol{q}_j \boldsymbol{k}_m^T / \sqrt{d_h})} = \frac{a_i^j}{1 - a_m^j} \tag{5}$$

$$o'_j(m) = \sum_{i\neq m} \hat{a}_i^j(m) \boldsymbol{v}_i = \sum_{i\neq m} \frac{a_i^j}{1 - a_m^j} \boldsymbol{v}_i \tag{6}$$

where $\hat{a}_i^j(m)$ is the attention score between query token with index j and previous token with index i when the key-value pair of m-th token is removed. Previous methods overlook the normalization of Softmax and thus do not consider the denominator term $1 - a_m^j$ in the formula, which affects the effectiveness of the methods. The L_1 norm of the token removal error $E_m^j = ||o_j - o'_j(m)||_1$ due to the removal of the m-th token is:

$$E_m^j = \frac{a_m^j}{1 - a_m^j} \left\|\boldsymbol{v}_m - o_j\right\|_1 \tag{7}$$

Figure 3 presents the different token importance estimation scores versus token removing error. The most important tokens with the highest attention output errors cannot be effectively estimated using either AAS or PAASVN. They both miss a substantial number of important tokens.

Computing directly using the error function in Equation (7) for all pairs of i and j incurs overhead $O(wLD)$, where w is the size of the observation window, L is the sequence length, D is the hidden size. Instead, we can approximate it by its upper error bound, with only $O(LD)$ overhead. Using

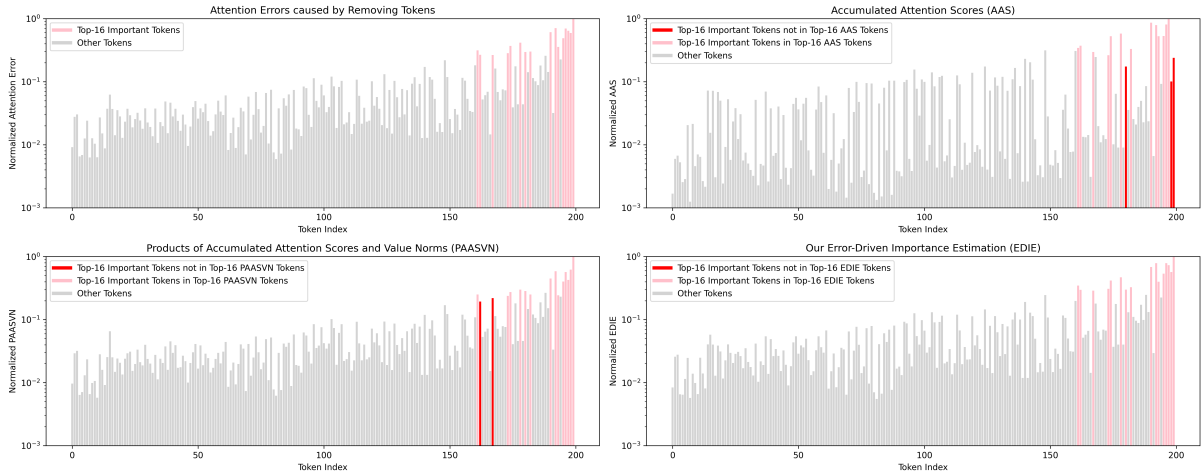


Figure 3: Token importance estimation from previous work vs. token removing error. Top-left figure presents the top-16 most important tokens selected based on the output error caused by token removal. Top-right and bottom-left figures show the tokens determined by the AAS method and the PAASVN method respectively. Bottom-right figure presents the tokens selected based on our EDIE method. The AAS method and PAASVN method both miss important tokens, while our EDIE method find all important tokens.

the triangle inequality, the upper bound of the error is:

$$E_m^j \leq U_m^j = \frac{a_m^j}{1 - a_m^j}(\|\boldsymbol{v}_m\|_1 + \|o_j\|_1) \quad (8)$$

where U_m^j is the upper bound of the error of the attention head at step j when removing the key and value of token m. We introduce α in the denominator to prevent overflow and excessively large values. The revised formula is:

$$\tilde{U}_m^j = \frac{a_m^j}{1 + \alpha - a_m^j}(\|\boldsymbol{v}_m\|_1 + \|o_j\|_1) \quad (9)$$

The final form of the upper bound error $\tilde{U}_m^j$ is presented as a multi-factor approach, which includes the attention score, the L_1 norm of the value state, the L_1 norm of the output state and attention renormalization when a token is removed.

For the final importance estimation, we adopt a weighted observation window approach, in contrast to previous works that assign equal weights to all tokens within the window (as shown in Equations 3–4). This is because not all tokens in the window contribute equally. Tokens with larger attention score tend to carry more information and should be more heavily weighted. To measure this weight, we use the maximum attention score of each token:

$$c_j = \max_{w < i \leq t-2w} a_i^j \quad (10)$$

c_j is the weight for the token j within the observation window. Note that attention scores for tokens at the beginning of the input or those close to the observation tokens tend to be large, while more informative scores reflecting detailed context are often found in the middle of the sequence. Thus we manually exclude these regions when determining the maximum attention score. We found that excluding the $(i \leq w)$ and $(i > t - 2w)$ reaches the best practice in experiment.

The final importance calculation accumulates weighted $\tilde{U}_i^j$ in the observation window w:

$$I_i = \sum_{t-w<j\leq t} c_j \tilde{U}_i^j \quad (11)$$

I_i is the importance of the i-th token. This importance is directly used for estimating the importance of each token within each attention head. For grouped query attention (GQA), we compute the importance of the i-th token by summing the values of I_i across different heads that belong to the same key-value group.

3.3 Task-aware dynamic allocation of KV Cache Capacity

Having established the importance estimation based on token removal error, we first use it to estimate layer-wise error, which in turn is used to assess task difference. Together, these components inform the final allocation of KV cache across layers, heads, and tasks. An overview is shown on the right side of Figure 2.

For layer-wise error, we consider the direct output error introduced by pruning a subset of tokens

within the layer. However, since each layer contributes differently to the final output of the model, we also account for its indirect impact on the overall model error, reflecting the layer's significance within the network. To estimate the direct output error of the layer itself, we consider the output error of the layer, which can be approximated by the sum of the upper bounds I_i on the error of tokens outside the observation window across all attention heads. The layer internal error equation follows:

$$E_{internal} = \sum_{h} \sum_{i \leq t-w} I_i^h \tag{12}$$

For the indirect impact, we quantify the influence of each layer's output error to the model output by analyzing the angular change in the hidden states. We consider the cosine dissimilarity between the residual R from previous layer and the sum of the residual and the current layer output $(R + O)$ to obtain the final output error of the model:

$$E_{layer}^l = (1 - \frac{R^l \cdot (O^l + R^l)}{\|R^l\|_2 \, \|O^l + R^l\|_2}) E_{internal}^l \tag{13}$$

Since the absolute values of hidden states are normalized through layer normalization, the directional information of the hidden states becomes the primary focus. Therefore, the impact of the output error of each layer to the final model output is reflected in its ability to alter the direction of the hidden states. Thus, we employ cosine dissimilarity to quantify the influence of each layer's output error on the model output.

Unlike previous methods that treat all inputs uniformly, we incorporate task context into the allocation process, allowing KV cache capacity to be dynamically distributed based on task-specific characteristics. We reuse the sum of the layer-wise errors across all layers to define the difficulty for a given context:

$$D_k = \sum_{l} E_{layer}^{l,k} \tag{14}$$

where k indicates the k-th input context. Note that computing D_k directly requires summing the errors across all layers, which is impractical as it would require completing the full forward pass in advance. To address this, we use an equivalent but more computationally efficient task difficulty term – $\bar{D}$, which calculates the average value of D:

$$\bar{D}_k = \frac{1}{k} \sum_{i<k} D_i \tag{15}$$

Similarly, we use the average value of E_{layer}^l as the layer difficulty of the layer in the task:

$$\bar{E}_{layer}^{l,k} = \frac{1}{k} \sum_{i<k} E_{layer}^{l,i} \tag{16}$$

The final number of KV cache capacity N_k^l assigned to layer l for context k is then determined by distributing the total KV cache capacity C proportionally to each layer's difficulty:

$$N_k^l = \frac{\bar{E}_{layer}^{k,l}}{\bar{D}_k} C \tag{17}$$

$$\sum_{l} N_k^l = C \tag{18}$$

After determining the per-layer capacity, we proceed to select which KV entries to retain at the head level. Specifically, we preserve the top N_k^l token positions with the highest importance scores from Equation(11), regardless of which heads they belong to. This importance-based selection naturally supports uneven allocation between heads, as some heads may retain more tokens than others. For GQA, only a single KV cache is maintained for each key-value group. Our method thus enables dynamic KV cache allocation across layers, heads, and tasks based on token importance.

4 Experiments

4.1 Settings

Datasets: To assess the effectiveness of our method in real-world scenarios, we evaluate our methods on 2 open-source datasets. We adopt LongBench (Bai et al., 2023), which comprises 16 English tasks, each containing between 150 and 500 samples, and covers a diverse range of long-text applications including question answering, text summarization, etc. We also evaluate our method on the Needle-in-a-Haystack task (Liu et al., 2024) to assess key information identification and in-context retrieval over long sequences.

Models&Baselines: We use two open-source LLMs: LLaMA3.1-8B-Instruct(Grattafiori et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) for experiments. We compare our works with previous works: StreamingLLM(Xiao et al., 2024), PyramidKV (Cai et al., 2024), and AdaKV (Feng et al., 2024). Since AdaKV+SnapKV outperforms AdaKV+PyramidKV in its original paper, we refer to AdaKV+SnapKV simply as AdaKV in

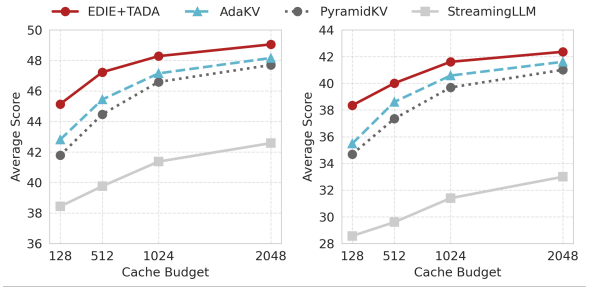
Table 1: Detailed results of LLaMA-3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 on LongBench.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shotLearning			Synthetic		Code		Ave. Score
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
LLaMA-3.1-8B-Instruct, B=FULL																	
Full Cache	30.22	45.37	55.80	55.97	45.00	31.26	35.12	25.38	27.20	72.50	91.64	43.57	9.41	99.50	62.88	56.43	49.20
LLaMA-3.1-8B-Instruct, B=128																	
StreamingLLM	22.24	20.87	31.72	44.02	37.55	24.54	18.76	21.09	18.48	40.50	84.41	38.82	8.00	**99.50**	57.02	47.29	38.43
PyramidKV	25.70	24.69	47.74	52.87	40.57	27.23	20.02	22.38	19.74	44.50	88.81	40.30	7.22	**99.50**	57.25	49.90	41.78
AdaKV	24.90	24.41	49.95	53.15	41.73	28.55	20.54	**23.21**	20.28	50.50	89.49	**40.71**	7.45	99.00	58.74	**52.40**	42.81
EDIE+TADA	**28.93**	**35.19**	**54.16**	**56.47**	**45.90**	**30.65**	**21.19**	23.08	**20.94**	**56.50**	**91.11**	38.87	**9.25**	**99.50**	**59.83**	50.71	**45.14**
LLaMA-3.1-8B-Instruct, B=512																	
StreamingLLM	25.51	25.78	34.19	45.01	35.91	24.93	23.61	21.26	23.57	57.50	87.86	41.44	6.98	96.50	60.85	51.02	41.37
PyramidKV	28.71	39.89	52.86	54.00	44.20	**31.22**	24.74	23.73	24.28	66.00	91.07	41.42	8.39	**99.50**	61.99	53.44	46.59
AdaKV	29.07	40.16	52.44	53.90	43.05	31.10	**25.75**	**24.39**	**24.85**	69.00	**92.34**	**42.05**	7.98	**99.50**	63.43	**55.32**	47.15
EDIE+TADA	**29.70**	**43.35**	**56.95**	**57.62**	**48.65**	31.13	25.34	24.17	24.80	**70.50**	91.73	41.76	**8.50**	**99.50**	**64.48**	54.31	**48.28**
Mistral-7B-Instruct-v0.2, B=FULL																	
Full Cache	26.74	32.84	50.00	43.45	27.77	18.49	32.91	24.64	26.99	71.00	86.23	43.32	2.94	86.31	57.39	54.32	42.83
Mistral-7B-Instruct-v0.2, B=128																	
StreamingLLM	18.02	13.32	27.41	30.49	21.81	11.85	15.49	19.42	17.84	44.00	80.74	37.35	**3.57**	22.93	49.07	43.74	28.57
PyramidKV	20.78	19.76	42.92	36.31	22.37	13.91	18.84	21.84	20.42	46.50	84.55	40.23	2.79	65.46	51.49	46.83	34.69
AdaKV	20.10	20.14	44.20	37.32	23.90	15.29	19.15	22.27	20.71	50.00	84.20	39.64	3.09	67.11	52.26	48.25	35.48
EDIE+TADA	**21.03**	**23.75**	**47.81**	**38.94**	**26.13**	**16.37**	**20.90**	**23.66**	**22.07**	**64.00**	**85.54**	**40.70**	3.54	**73.22**	**54.55**	**51.39**	**38.35**
Mistral-7B-Instruct-v0.2, B=512																	
StreamingLLM	21.12	15.99	30.82	30.39	22.32	10.92	21.43	19.98	22.88	61.50	82.11	41.89	3.21	17.32	53.43	47.05	31.40
PyramidKV	21.81	24.33	48.67	39.31	25.14	17.51	23.22	23.16	23.87	66.00	85.20	41.96	3.08	85.71	55.10	50.98	39.69
AdaKV	23.62	27.34	48.70	39.81	26.42	17.36	23.42	23.26	24.50	67.50	86.46	42.04	3.04	**86.64**	56.11	53.05	40.58
EDIE+TADA	**25.84**	**30.21**	**49.05**	**42.43**	**26.43**	**18.37**	**25.26**	**23.69**	**25.18**	**71.00**	**86.47**	**42.85**	**3.18**	85.93	**56.15**	**53.69**	**41.61**

the results that follow. The full KV cache is also presented to assess performance degradation.

Parameters: Our method adopts an observation window size w of 32 and a max-pooling kernel size of 7, following the configuration settings in AdaKV which also applies a similar window approach. The α in Equation(9) is by default set to 0.1. The range of i in Equation(10) is by default set to w to $t-2w$.

4.2 LongBench



(a) LLaMA3.1-8B-Instruct (b) Mistral-7B-Instruct-v0.2

Figure 4: Results of LLaMA-3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 on LongBench.

The results are presented in Figure 4 and Table 1. By combining EDIE and TADA, our method demonstrates consistent superiority across both the LLaMA3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 architectures under varying KV cache budgets (128/256/512/1024), achieving higher average scores than baselines on the LongBench benchmark. Under KV cache budget=128, our method achieves SOTA performance on 13/16 LongBench tasks for both LLaMA3.1-8B-Instruct and 15/16 LongBench tasks for Mistral-7B-Instruct-v0.2 models, and achieves near SOTA performance on the remaining tasks. Our method also demonstrates consistent performance superiority in other evaluated cache budgets, which conforms its effectiveness to different budgets.

4.3 Needle-In-A-Haystack

We primarily compared the results with AdaKV in Figure 5. The vertical axis shows the insertion depth of the needle, and the horizontal axis shows the context length. Each grid cell reflects a test result: green for successful retrieval, yellow for failure. Our approach significantly outperforms AdaKV for both LLaMA3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 models under different KV cache budgets (128/512), demonstrating its effectiveness in long-text processing. For LLaMA3.1-8B-Instruct, when the budget is set to 128, our approach improves the accuracy from 72.6 to 88.3 compared to AdaKV, demonstrating its superior performance. Notably, on both models, our approach achieves accuracy comparable to AdaKV

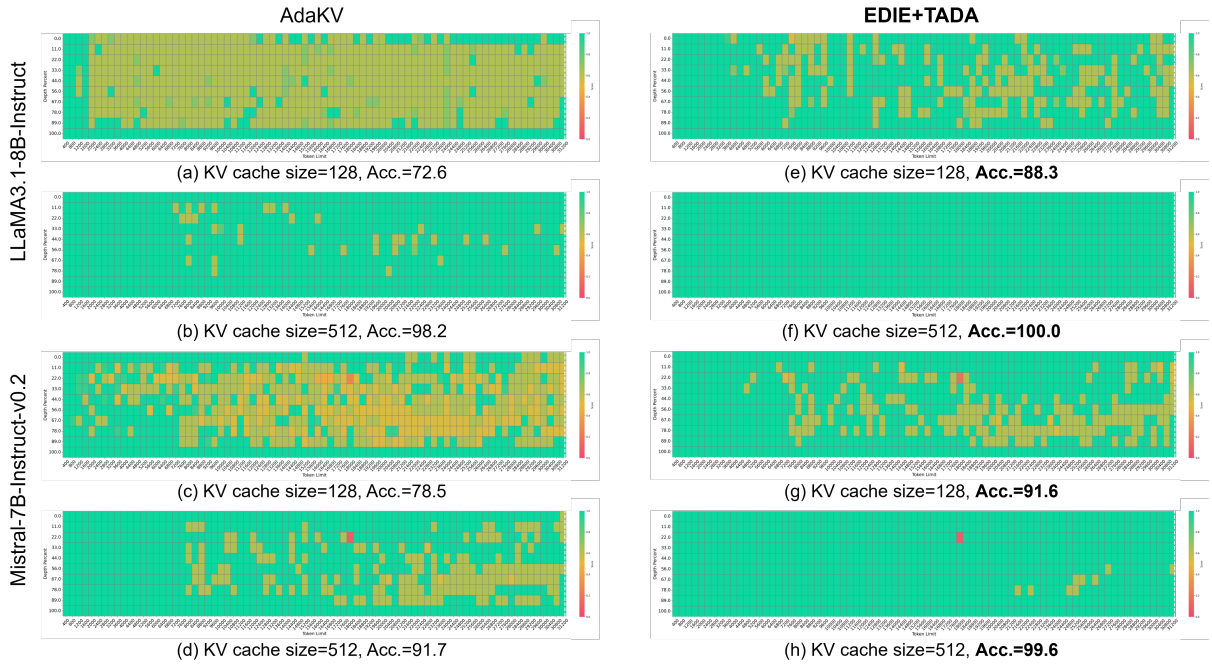


Figure 5: The evaluation results from Needle-In-A-Haystack across 128 and 512 cache sizes for LLaMA3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 in context size of 32k.

while using only a quarter of its KV cache budget. This demonstrates our method's ability to allocate KV cache capacity more effectively across different layers and more precisely identify important tokens for KV cache storage.

4.4 Ablation Study

We conducted ablation experiments on EDIE, TADA and the hyperparameter α using the Mistral-7B-Instruct-v0.2 model with a budget of 128. The results are presented in Table 2. Compared to the baseline, the EDIE method alone, when combined with Uniform and Pyramid allocation, improves the model's accuracy on LongBench and Needle-in-a-Haystack, demonstrating its effectiveness. The results also indicate that EDIE is not sensitive to the hyperparameter α, and setting it to 0.1 yields optimal performance. Finally, applying the TADA method on top of EDIE further enhances the performance, confirming the effectiveness of TADA.

To examine the observation window inclusion range used in our weighting strategy as in Equation (10), we conducted ablation experiments using various definitions of the weight coefficient c_j. As shown in Table 3, removing token weights altogether ("None") already offers a strong baseline, but incorporating attention-based weighting improves performance on both tasks. Notably, assigning weights using only tokens in the mid-sequence range $(w < i \leq t - 2w)$, where attention scores are more likely to reflect contextually relevant information, achieves the best results. This confirms that not all tokens contribute equally and limit the observation range for maximum attention leads to more reliable importance estimation.

To assess the role of cosine dissimilarity in Equation (13) for modeling the influence of each layer's output on the final prediction, we compare model performance with and without its inclusion.

Table 2: The Ablation study for EDIE and TADA with Mistral-7B-Instruct-v0.2 in the budget of 128. The values 0.05, 0.1, and 0.2 are used as the hyperparameter α in the EDIE method. The **tw** refers to the token weighting mechanism in EDIE, which represents the weighting coefficient c_j in Equation (11). The **cd** denotes the cosine dissimilarity employed in the TADA method in Equation (13).

Method	LongBench Avg Score	Needle in A Haystack
AAS + Uniform	35.48	78.5
PAASVN + Uniform	36.12	82.2
$\text{EDIE}_{0.1,w/\ tw}$ + Uniform	37.49	89.0
$\text{EDIE}_{0.1,w/o\ tw}$ + Uniform	36.73	83.1
$\text{EDIE}_{0.1,w/\ tw}$ + Uniform	37.49	89.0
$\text{EDIE}_{0.05,w/\ tw}$ + Uniform	37.42	88.9
$\text{EDIE}_{0.2,w/\ tw}$ + Uniform	37.43	88.9
$\text{EDIE}_{0.1,w/\ tw}$ + Uniform	37.49	89.0
$\text{EDIE}_{0.1,w/\ tw}$ + Pyramid	37.61	87.0
$\text{EDIE}_{0.1,w/\ tw}$ + $\text{TADA}_{w/o\ cd}$	37.37	92.6
$\text{EDIE}_{0.1,w/\ tw}$ + $\text{TADA}_{w/\ cd}$	38.35	91.6

Table 3: The Ablation study for the weight c_j of tokens within the observation window with Mistral-7B-Instruct-v0.2 in the budget of 128.

c_j	LongBench Avg Score	Needle in A Haystack
$None$	37.60	86.2
$\max_i(a_i^j)$	36.20	85.0
$\max_{i\leq t-2w}(a_i^j)$	36.32	85.7
$\max_{i>w}(a_i^j)$	37.16	89.3
$\max_{w<i\leq t-2w}(a_i^j)$	38.35	91.6

For LLaMA3.1-8B-Instruct, it substantially boosts 8.6% Needle-in-a-Haystack accuracy, with only a minimal drop of 0.06 in LongBench, suggesting better selective retrieval. In contrast, Mistral-7B-Instruct-v0.2 sees a notable LongBench gain but a slight drop in Needle accuracy. Given the complexity of LongBench, a 0.98 improvement is particularly meaningful, suggesting enhanced robustness on more challenging & diverse tasks. These results show that cosine dissimilarity benefits models differently—enhancing in-context retrieval for LLaMA3.1 and overall robustness for Mistral. Overall, it supports more accurate layer importance assessment for adaptive KV cache allocation.

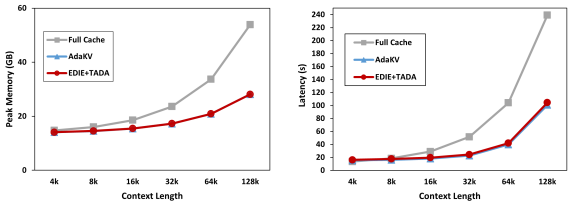
Table 4: The Ablation study for the cosine dissimilarity (cd) in Equation(13) with LLaMA3.1-8B-Instruct and Mistral-7B-Instruct-v0.2 in the budget of 128.

Model	TADA	LongBench Avg Score	Needle in A Haystack
LLaMA3.1	w/o cd	45.22	79.7
	w/ cd	45.16	88.3
Mistral-v0.2	w/o cd	37.37	92.6
	w/ cd	38.35	91.6

4.5 Computational Efficiency

We evaluated the computational efficiency of the method using the Mistral-7B-Instruct-v0.2 model, tested on one AMD Instinct MI250 GPU. The context size ranged from 4K to 128K, with the number of generation tokens set to 256 and the batch-size set to 1. The results are presented in Figure 6. Compared to Full Cache, our method significantly reduces memory overhead and inference latency for longer sequences. Compared to AdaKV, our method does not introduce additional memory overhead or inference latency.

Our method seamlessly adapt to multi-batchsize setup. We benchmarked the efficiency of both full cache method and our method under an 64GB GPU memory constraint using the Mistral-7B-Instruct-v0.2 model. As shown in Table 5, our method consistently supports larger maximum batch sizes across all context lengths. It also achieves higher throughput at longer context lengths, indicating better utilization of computational resources. Overall, these results demonstrate that EDIE+TADA offers superior efficiency for high-throughput, long-sequence inference workloads.



(a) Memory Overhead (b) Runtime Efficiency

Figure 6: Computational Efficiency for Mistral-7B-Instruct-v0.2 on the AMD Instinct MI250 GPU.

Table 5: Comparison of maximum batch size and throughput on Mistral-7B-Instruct-v0.2 under 64GB GPU memory.

	Full Cache		EDIE+TADA	
Context Length	Max Batch Size	Throughput (tokens/s)	Max Batch Size	Throughput (tokens/s)
4k	32	56.64	64	127.15
8k	16	27.97	32	69.41
16k	8	13.22	16	34.42
32k	4	6.08	8	15.98
64k	2	2.77	4	6.95
128k	1	1.07	2	2.74
256k	-	-	1	0.95

5 Conclusion

In this study, we propose a task-aware KV cache eviction strategy for large language models, where dynamic allocation of KV cache capacity is guided by error-driven importance estimation. Our method precisely quantifies the importance of KV cache entries via estimating token removal error, and achieves full dynamic allocation of the KV cache capacity by introducing task difference. Experiments show our approach consistently outperforms existing methods, achieving new SOTA results on LongBench and Needle-In-A-Haystack benchmarks. Our results highlight a promising direction for reducing memory overhead in sigle/multi-batch LLM inference while preserving top accuracy in long context understanding.

6 Limitations

While our method demonstrates strong efficiency and achieves state-of-the-art performance on long-context benchmarks, it has not yet been integrated into optimized attention kernels such as FlashAttention. Incorporating our KV cache reduction strategy into such kernels could potentially yield further improvements in both speed and memory efficiency, which we leave for future work.

References

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*.

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

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

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

Aaron Grattafiori, Abhimanyu Dubey, and Abhinav Jauhri .et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024a. Attention score is not all you need for token importance indicator in KV cache reduction: Value also matters. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166, Miami, Florida, USA. Association for Computational Linguistics.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024b. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166.

Xingyang He, Jie Liu, and Shaowei Chen. 2025. Task-kv: Task-aware kv cache optimization via semantic differentiation of attention heads. *arXiv preprint arXiv:2501.15113*.

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*.

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.

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

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *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.

Jianxun Wang and Yixiang Chen. 2023. A review on code generation with llms: Application and evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, pages 284–289. IEEE.

Yonghui Wu. 2024. Large language model and text generation. In *Natural Language Processing in Biomedicine: A Practical Guide*, pages 265–297. Springer.

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

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3258–3270.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Haofei Zhao, Yilun Liu, Shimin Tao, Weibin Meng, Yimeng Chen, Xiang Geng, Chang Su, Min Zhang, and Hao Yang. 2024. From handcrafted features to

llms: A brief survey for machine translation quality estimation. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. 2024. Dynamickv: Task-aware adaptive kv cache compression for long context llms. *arXiv preprint arXiv:2412.14838*.

A Implementation Details

Consistent with AdaKV, our method adopts a window size of 32, utilizing mean pooling with a kernel size of 7. AdaKV is configured with a floor ratio of 0.2, a window size of 32, and also employs mean pooling with a kernel size of 7. The keys and values of tokens within the recent window size are unconditionally retained and are also counted toward the budget.

B Dataset Details

We select the English subset from Longbench (Bai et al., 2023). Table 6 shows the information on 16 tasks that we use in the experiments.

Dataset	Avg len	Metric	#data
NarrativeQA	18,409	F1	200
Qasper	3,619	F1	200
MultiFieldQA-en	4,559	F1	150
HotpotQA	9,151	F1	200
2WikiMultihopQA	4,887	F1	200
MuSiQue	11,214	F1	200
GovReport	8,734	Rouge-L	200
QMSum	10,614	Rouge-L	200
MultiNews	2,113	Rouge-L	200
TREC	5,177	Accuracy (CLS)	200
TriviaQA	8,209	F1	200
SAMSum	6,258	Rouge-L	200
PassageCount	11,141	Accuracy (EM)	200
PassageRetrieval-en	9,289	Accuracy (EM)	200
LCC	1,235	Edit Sim	500
RepoBench-P	4,206	Edit Sim	500

Table 6: Dataset statistics in LongBench, including key metrics. 'Avg len' represents the average number of words for English datasets (or code). 'Accuracy (CLS)' denotes classification accuracy, while 'Accuracy (EM)' refers to exact match accuracy.

C System Configuration Details

All experiments are conducted on a server equipped with two AMD EPYC™ 73F3 16-Core processors, 1024 GB of system memory, and 8 AMD Instinct™ MI250 GPUs. The software environment includes ROCM™ 6.3.2 and Ubuntu 22.04.5 LTS. We use PyTorch 2.4.0 as the deep learning framework, along with HuggingFace Transformers 4.44.2 and FlashAttention 2.6.3.

D Detailed Experiment Results

Table 7 presents the detailed results of LLaMA-3.1-8B-Instruct on LongBench. Table 8 presents the detailed results of Mistral-7B-Instruct-v0.2 on LongBench. The EDIE+TADA method achieves the highest average score across 16 tasks under all four KV cache budget settings (128, 256, 512, and 1024).

Figure 7 shows the evaluation results from the Needle-In-A-Haystack benchmark across KV cache budgets of 64, 128, 256, and 512 for LLaMA3.1-8B-Instruct with a context length of 32k. Our method consistently outperforms AdaKV across all budget settings, achieving substantial improvements. Notably, at a budget of 256, our approach reaches an accuracy of 99.9%, compared to only 86.4% achieved by AdaKV. Figure 8 presents the evaluation results from the Needle-In-A-Haystack benchmark across cache sizes of 64, 128, 256, and 512 for Mistral-7B-Instruct-v0.2 with a context length of 32k. Our method consistently outperforms AdaKV across all cache size settings, demonstrating significant improvements. Specifically, at a cache size of 128, our approach achieves an accuracy of 91.6%, whereas AdaKV attains only 78.5%.

Table 7: Detailed results of LLaMA-3.1-8B-Instruct on LongBench.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shotLearning			Synthetic		Code		Ave. Score
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Full Cache	30.22	45.37	55.80	55.97	45.00	31.26	35.12	25.38	27.20	72.50	91.64	43.57	9.41	99.50	62.88	56.43	49.20
B=128																	
StreamingLLM	22.24	20.87	31.72	44.02	37.55	24.54	18.76	21.09	18.48	40.50	84.41	38.82	8.00	**99.50**	57.02	47.29	38.43
PyramidKV	25.70	24.69	47.74	52.87	40.57	27.23	20.02	22.38	19.74	44.50	88.81	40.30	7.22	**99.50**	57.25	49.90	41.78
AdaKV	24.90	24.41	49.95	53.15	41.73	28.55	20.54	**23.21**	20.28	50.50	89.49	**40.71**	7.45	99.00	58.74	**52.40**	42.81
EDIE+TADA	**28.93**	**35.19**	**54.16**	**56.47**	**45.90**	**30.65**	**21.19**	23.08	**20.94**	**56.50**	**91.11**	38.87	**9.25**	**99.50**	**59.83**	50.71	**45.14**
B=256																	
StreamingLLM	22.71	23.79	31.80	43.43	36.55	25.55	21.29	20.68	20.67	46.00	87.11	40.82	7.20	**99.50**	59.89	49.19	39.76
PyramidKV	25.53	33.15	51.44	55.03	42.42	28.62	22.57	23.37	22.33	56.50	91.19	41.28	6.97	**99.50**	60.36	51.18	44.47
AdaKV	26.11	33.39	51.44	54.94	42.15	29.54	23.01	23.85	22.88	63.50	91.57	40.94	8.00	**99.50**	61.95	**54.33**	45.44
EDIE+TADA	**28.23**	**39.82**	**56.52**	**57.89**	**48.21**	**30.27**	**23.60**	**24.01**	**23.27**	**65.00**	**91.89**	**41.42**	**8.49**	**99.50**	**63.94**	53.66	**47.23**
B=512																	
StreamingLLM	25.51	25.78	34.19	45.01	35.91	24.93	23.61	21.26	23.57	57.50	87.86	41.44	6.98	96.50	60.85	51.02	41.37
PyramidKV	28.71	39.89	52.86	54.00	44.20	**31.22**	24.74	23.73	24.28	66.00	91.07	41.42	8.39	**99.50**	61.99	53.44	46.59
AdaKV	29.07	40.16	52.44	53.90	43.05	31.10	**25.75**	**24.39**	**24.85**	69.00	**92.34**	**42.05**	7.98	**99.50**	63.43	**55.32**	47.15
EDIE+TADA	**29.70**	**43.35**	**56.95**	**57.62**	**48.65**	31.13	25.34	24.17	24.80	**70.50**	91.73	41.76	**8.50**	**99.50**	**64.48**	54.31	**48.28**
B=1024																	
StreamingLLM	24.97	30.22	37.06	46.57	39.14	25.24	26.01	21.08	25.72	63.50	88.87	42.28	6.98	89.00	61.30	53.40	42.58
PyramidKV	29.62	43.66	54.10	55.06	44.22	31.30	27.27	24.30	25.68	68.50	91.27	41.96	7.73	**99.50**	63.13	55.85	47.70
AdaKV	29.23	44.09	53.82	54.80	44.01	31.40	**28.86**	24.73	**26.04**	**72.50**	91.72	**42.56**	7.82	**99.50**	63.22	**56.33**	48.16
EDIE+TADA	**31.14**	**45.72**	**57.36**	**57.71**	**48.39**	**31.42**	27.22	**24.84**	25.71	72.00	**92.04**	42.32	**8.47**	**99.50**	**65.28**	55.70	**49.05**

Table 8: Detailed results of Mistral-7B-Instruct-v0.2 on LongBench.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shotLearning			Synthetic		Code		Ave. Score
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Full Cache	26.74	32.84	50.00	43.45	27.77	18.49	32.91	24.64	26.99	71.00	86.23	43.32	2.94	86.31	57.39	54.32	42.83
B=128																	
StreamingLLM	18.02	13.32	27.41	30.49	21.81	11.85	15.49	19.42	17.84	44.00	80.74	37.35	**3.57**	22.93	49.07	43.74	28.57
PyramidKV	20.78	19.76	42.92	36.31	22.37	13.91	18.84	21.84	20.42	46.50	84.55	40.23	2.79	65.46	51.49	46.83	34.69
AdaKV	20.10	20.14	44.20	37.32	23.90	15.29	19.15	22.27	20.71	50.00	84.20	39.64	3.09	67.11	52.26	48.25	35.48
EDIE+TADA	**21.03**	**23.75**	**47.81**	**38.94**	**26.13**	**16.37**	**20.90**	**23.66**	**22.07**	**64.00**	**85.54**	**40.70**	3.54	**73.22**	**54.55**	**51.39**	**38.35**
B=256																	
StreamingLLM	19.08	15.30	28.27	31.87	22.26	11.37	18.08	19.37	19.99	51.00	80.92	39.62	**3.57**	15.90	51.74	45.22	29.60
PyramidKV	20.39	22.63	46.67	38.64	23.73	15.82	21.34	22.30	22.01	57.50	83.63	40.49	2.99	75.84	53.56	50.03	37.35
AdaKV	20.97	23.71	47.52	38.48	24.77	15.76	21.89	22.94	22.70	63.50	86.25	40.75	2.46	80.24	54.42	51.46	38.61
EDIE+TADA	**23.00**	**25.90**	**49.31**	**39.43**	**24.91**	**16.46**	**23.40**	**23.54**	**23.73**	**68.50**	**86.50**	**41.74**	2.83	**83.34**	**54.98**	**52.62**	**40.01**
B=512																	
StreamingLLM	21.12	15.99	30.82	30.39	22.32	10.92	21.43	19.98	22.88	61.50	82.11	41.89	3.21	17.32	53.43	47.05	31.40
PyramidKV	21.81	24.33	48.67	39.31	25.14	17.51	23.22	23.16	23.87	66.00	85.20	41.96	3.08	85.71	55.10	50.98	39.69
AdaKV	23.62	27.34	48.70	39.81	26.42	17.36	23.42	23.26	24.50	67.50	86.46	42.04	3.04	**86.64**	56.11	53.05	40.58
EDIE+TADA	**25.84**	**30.21**	**49.05**	**42.43**	**26.43**	**18.37**	**25.26**	**23.69**	**25.18**	**71.00**	**86.47**	**42.85**	**3.21**	85.93	**56.15**	**53.69**	**41.61**
B=1024																	
StreamingLLM	22.15	18.64	31.03	32.94	22.45	11.93	23.89	20.60	25.48	64.00	84.71	41.59	**3.49**	22.15	53.72	49.19	33.00
PyramidKV	24.12	29.44	48.83	40.30	25.94	**19.42**	25.29	23.52	25.77	68.00	86.30	41.62	2.84	86.07	56.06	52.57	41.01
AdaKV	25.11	29.98	49.27	40.62	27.05	18.54	25.85	23.46	26.08	68.50	86.30	42.77	2.92	**88.27**	56.88	**54.16**	41.61
EDIE+TADA	**26.99**	**32.11**	**49.46**	**42.74**	**27.26**	19.11	**28.51**	**23.76**	**26.60**	**71.00**	**86.54**	**43.38**	2.82	86.14	**57.47**	53.92	**42.36**

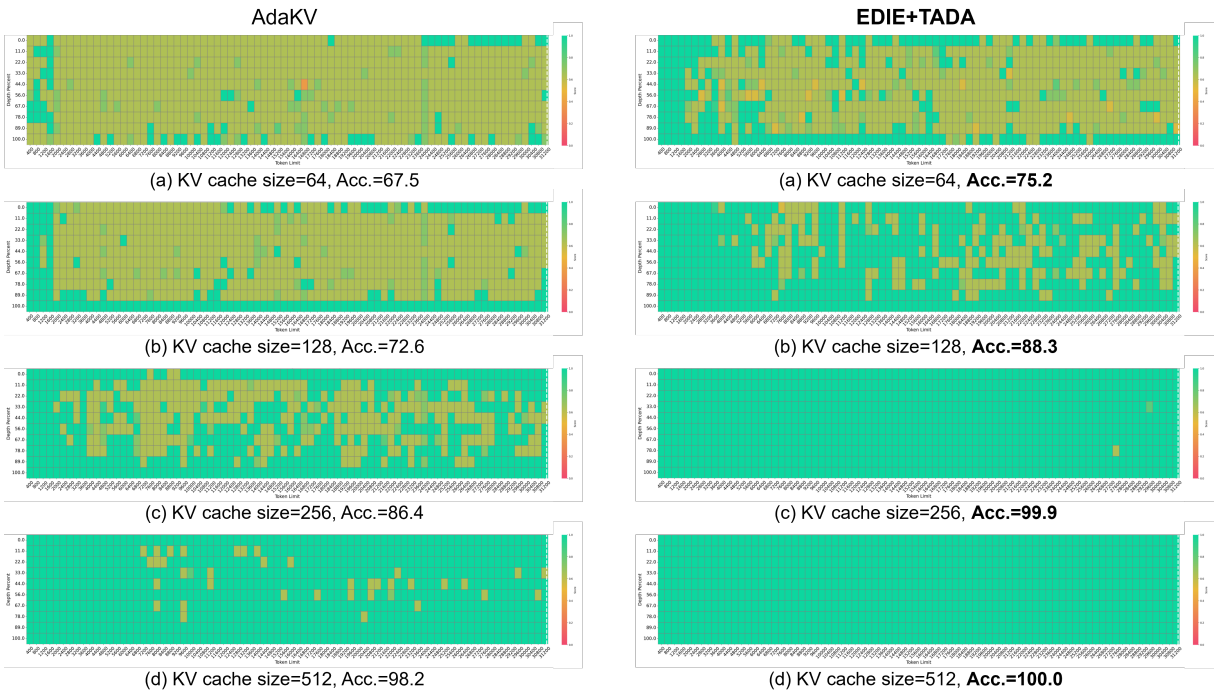


Figure 7: The evaluation results from Needle-In-A-Haystack across 64, 128, 256, and 512 cache sizes for LLaMA3.1-8B-Instruct in context size of 32k.

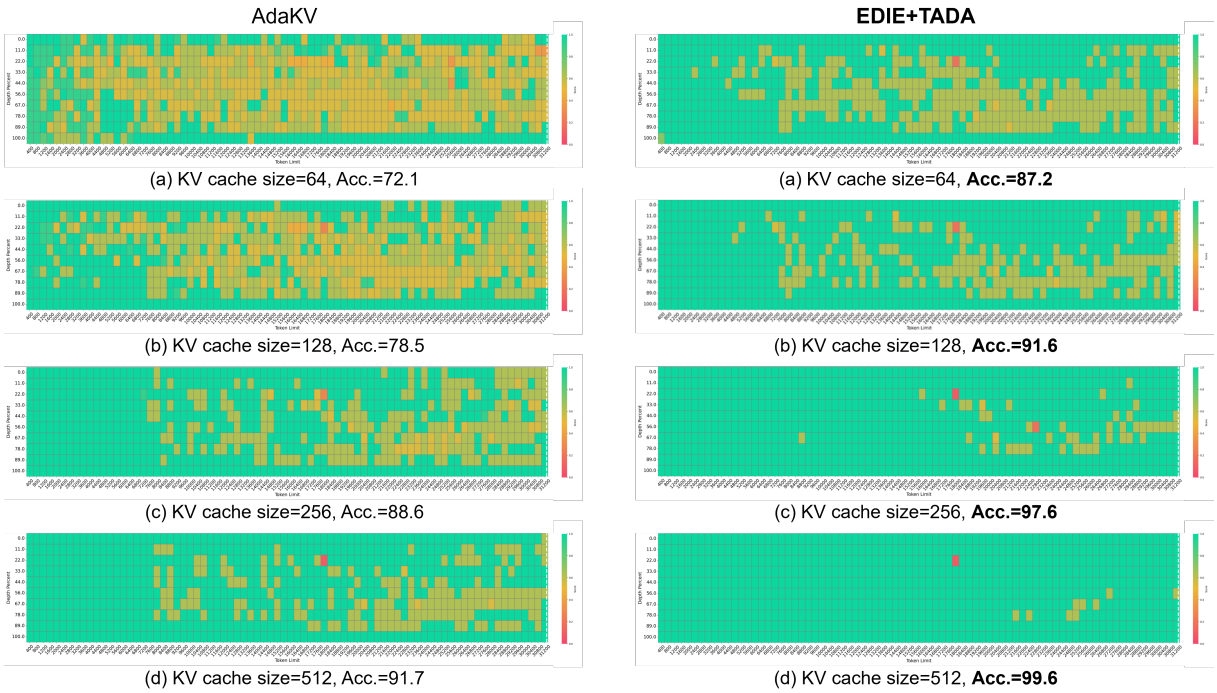


Figure 8: The evaluation results from Needle-In-A-Haystack across 64, 128, 256, and 512 cache sizes for Mistral-7B-Instruct-v0.2 in context size of 32k.