Efficient Cooperation-Aware Key and Value Management for LLM Inference

Qiheng Sun
The Hong Kong Polytechnic University
Zhejiang University
qiheng.sun@polyu.edu.hk

Hongwei Zhang
Zhejiang University
hongweizhang@zju.edu.cn

Junxu Liu
The Hong Kong Polytechnic University
junxu.liu@polyu.edu.hk

Haocheng Xia
University of Illinois Urbana-Champaign
hxia7@illinois.edu

Jinfei Liu*
Zhejiang University
jinfeiliu@zju.edu.cn

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

Haibo Hu
The Hong Kong Polytechnic University
haibo.hu@polyu.edu.hk

ABSTRACT

Key-value (KV) caching is a widely used technique for boosting performance in database and storage systems. It keeps frequently accessed data in fast storage to minimize redundant data fetching and improve throughput. This same idea has been adopted in Large Language Models (LLMs), where it avoids recomputing the key and value states of previous tokens in attention heads during autoregressive decoding, thereby greatly accelerating inference. However, the KV cache in LLMs faces a significant challenge due to the substantial memory required to store these KV pairs in the inference process. This issue arises because each attention head in the LLM stores its own KV cache for all context tokens, leading to the cache size that grows linearly with sequence length. This has spurred research into efficient management of the KV cache of LLMs.

One of the promising directions is KV cache budget allocation, with several approaches proposing head-level allocation as they recognize that different attention heads play distinct roles. However, these methods assess each head in isolation, overlooking their cooperative contributions within the model, which results in a deviation from their true impact. To address this limitation, we propose CoKV, a novel method that efficiently manages the KV cache in LLM inference by modeling the cooperation among attention heads as a cooperative game. By attributing the contribution of each head within the model in advance, CoKV can more effectively allocate the global KV cache budget in KV cache optimization techniques such as eviction and quantization. Extensive experiments demonstrate the effectiveness of CoKV on long-context benchmarks (e.g., LongBench, NIAH, and RULER) and mathematical reasoning benchmarks (e.g., GSM8K and MATH) across multiple model families, including Qwen, Llama, and Mistral.

PVLDB Reference Format:
Qiheng Sun, Hongwei Zhang, Junxu Liu, Haocheng Xia, Jinfei Liu, Kui Ren, and Haibo Hu. Efficient Cooperation-Aware Key and Value Management for LLM Inference. PVLDB, 19(9): 2154 - 2167, 2026.
doi:10.14778/3819518.3819541

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DIVER/CoKV.

1 INTRODUCTION

A key-value (KV) cache is a data storage structure that organizes intermediate computation results, enabling efficient retrieval and reducing redundant computations. In database systems, key-value cache techniques are essential for improving data access performance. For instance, query result caching employs KV structures to store SQL query results as pairs, eliminating redundant executions [41]. Row/page-level caching leverages KV mappings to reduce disk I/O in transactional systems [43]. Furthermore, distributed metadata caching utilizes KV stores to manage high-speed metadata access, such as Hitmap [44] and MetaHive [18]. Recently, as AI applications have grown to serve millions of users and are now widely deployed across various domains such as content generation [26], automated services [5], and decision support systems [17], the underlying infrastructure has become critically important. Caching design techniques from database systems have become valuable for optimizing AI infrastructure, particularly for managing key-value caches in large language model inference. However, the unique computational patterns and memory access characteristics of LLM attention mechanisms require tailored solutions rather than direct application of existing database techniques.

The text generation process in LLMs primarily relies on auto-regressive decoding, where each new token is predicted based on all preceding tokens. This process heavily depends on the self-attention mechanism, which computes a weighted sum of values (V) for each token, with weights derived from the compatibility between its query (Q) and the keys (K) of all tokens in the sequence. During auto-regressive decoding, a naive implementation requires recalculating the key and value states for all previous tokens for each new token generated, leading to significant redundant computation. To eliminate this overhead, the KV cache mechanism is employed, which stores computed KV states to avoid recomputation, thereby greatly accelerating the decoding process. However, the total KV cache size becomes excessively large when processing long sequences or a large number of inputs, as it is proportional to

doi:10.14778/3819518.3819541

the number of attention heads and sequence length. This inevitably strains GPU memory and substantially raises deployment costs for large-scale applications. For example, Qwen3-32B processing a single 120k token input would require up to 80GB of memory for the KV cache, which significantly surpasses the 64GB memory required for the model parameters.

To address this challenge, research efforts have advanced in several directions by leveraging the unique characteristics of the KV cache in LLM to improve its efficiency and reduce memory cost. Some studies have explored methods to rank the importance of tokens within a single attention head, retaining only the top-k most significant ones. For example, methods like H2O, StreamingLLM, and SnapKV prioritize tokens within attention head by summing attention weights, discarding middle segments, or pooling attention scores within local windows, respectively [28, 49, 57]. In parallel, several studies [21, 39] have also investigated strategies for optimizing KV quantization to reduce KV cache costs, focusing on employing reduced bit-width representations to store KV states. Recent research has shown that attention heads exhibit highly heterogeneous behaviors, with some attending narrowly to a few critical tokens and others distributing attention more broadly across the context. This behavioral diversity has led to approaches that dynamically allocate the KV cache budget across heads. Specifically, DuoAttention [48] uses a reinforcement learning-based algorithm with synthetic data to identify retrieval heads and then assign a larger KV budget to these heads. AdaKV [12] assigns less cache to heads with highly concentrated attention distributions, as they can be approximated with fewer tokens, and more cache to heads with broader distributions to prevent information loss. HeadKV [13] evaluates a retrieval-reasoning score for each head by measuring its tendency to assign higher attention weights to critical tokens, and then allocates cache proportionally to these scores.

Motivation. While prior work has made significant advancements in adaptive KV cache management, a key limitation remains unresolved: *existing methods lack a global view for measuring attention head importance.* Treating attention heads as independent units, these approaches fail to capture the reality that their true utility emerges from heads' interaction in LLM. This independent evaluation can lead to suboptimal global cache allocation, as it may undervalue heads that play a critical supporting role in a collaborative context. To address this, we propose **CoKV** (**Co**operation-based **K**ey-**V**alue Cache), a novel method that evaluates the contribution of all attention heads based on their cooperative utility within the model. Leveraging principles from game theory, CoKV dynamically allocates KV cache budgets according to this evaluation.

We are inspired by the Shapley value [38], a seminal concept in cooperative game theory that offers a mathematically rigorous framework for fair contribution allocation. The Shapley value of a player p_i measures the expected marginal contribution that p_i provides to a coalition of players. In this work, each attention head can be treated as a player, with its importance assessed via its Shapley value. The marginal contribution is defined as $\mathcal{U}(\mathcal{S} \cup \{p_i\}) - \mathcal{U}(\mathcal{S})$ where $\mathcal{S}$ is a coalition of players excluding i and $\mathcal{U}$ is the utility function. A simple intuition for computing the Shapley value of each head in LLMs is to define $\mathcal{U}$ as the model performance metric. The Shapley value can capture each head's expected contribution to model utility across all possible subsets of retained heads, which is a coalition-aware notion that aligns naturally with KV cache allocation, where a head's priority should reflect its collaborative gain to the global inference performance. This is a #P-hard [10] problem as there are an exponential number of coalitions and corresponding marginal contributions, thus requiring an enormous number of model inferences. Although many studies [23, 33] have explored approximating the Shapley value to reduce computational costs, evaluating the importance of heads in LLMs remains prohibitively expensive.

The computational bottleneck in calculating the Shapley value arises from the fact that each sample of the marginal contribution can only be applied to a single player. Fortunately, Shapley value can be expressed as the expectation of the weighted complementary contribution, defined as $\mathcal{U}(\mathcal{S}) - \mathcal{U}(\mathcal{N} \setminus \mathcal{S})$, where $\mathcal{N}$ represents the set of all players [53]. Complementary contribution has an advantage over the marginal contribution in that $\mathcal{U}(\mathcal{S}) - \mathcal{U}(\mathcal{N} \setminus \mathcal{S})$ can be used to update the Shapley values for all players in $\mathcal{S}$. By expressing the Shapley value in terms of complementary contributions, we can interpret it as an expectation over these contributions computed at different coalition sizes $|\mathcal{S}|$. However, in the LLM setting, the cost of computing the complementary contributions in all coalition sizes is still prohibitively high. We observe that the average complementary contribution of a single player at different coalition sizes exhibits a strong correlation in Section 6.4. This insight allows us to approximate attention head importance by computing complementary contributions at only a few selected coalition sizes, rather than evaluating all possible sizes (i.e., from 1 to $|\mathcal{N}|$). By focusing on a few representative coalition sizes, we can significantly reduce the computational cost of estimating the contributions of heads. Additionally, we provide a theoretical error bound of this approach and demonstrate its efficiency. Moreover, CoKV eliminates cold start costs through a "one-time computation, shared usage" paradigm, allowing model providers to precompute importance scores offline on generic validation sets and distribute them alongside model weights for immediate adoption without additional overhead.

CoKV is a simple yet effective method and can integrate well with other inference optimization techniques. We integrate CoKV with widely used methods, including FlashAttention [9] and group query attention (GQA) [1]. CoKV achieves state-of-the-art performance in LongBench [3] using Qwen3-32B [51], Llama-3-8B-Instruct [11] and Mistral-7B [24] models. For Qwen3-32B, CoKV achieves 98.83% of the performance of the full KV when retaining an average of 1024 KV pairs. Additionally, we evaluate all methods within the token range up to 61k in the Needle-in-a-Haystack test and the RULER dataset [22], which are widely recognized benchmarks for evaluating long-text processing capabilities of LLMs, where CoKV also demonstrated the best performance. Experiments on mathematical reasoning datasets also demonstrate that CoKV possesses strong cross-task capabilities. Beyond efficiency-oriented inference, CoKV also has the potential to support LLM safety analysis by quantifying how attention heads contribute to risks such as prompt leakage [29], model extraction [30], memorization [47], incomplete unlearning [50], and robustness degradation under token compression [56].

We briefly summarize our contribution as follows.

- We identify the limitations of existing methods that assess attention heads independently. For the first time, we leverage cooperative game theory to measure the collaborative utility of the heads for a more efficient KV cache allocation.
- We introduce an efficient algorithm that estimates head contributions by reformulating the Shapley value and sampling key coalition sizes, achieving a drastic complexity reduction with a proven error bound for feasible LLM evaluation.
- Extensive experiments on multiple models and benchmarks show that CoKV significantly outperforms baselines, demonstrating strong practicality and cross-task generalization.

2 RELATED WORKS

In modern data centers, key-value caches are indispensable for delivering low-latency data access [45]. This has spurred significant research into enhancing cache efficiency within databases, focusing on areas such as cache replacement policies [54] and resource allocation [2]. Concurrently, the growing complexity of AI models has driven advancements in specialized optimization systems [58], creating new applications for traditional data management techniques. Among these, the key-value cache has emerged as a particularly prominent and beneficial component in AI systems [14, 27]. We now review KV cache compression techniques for LLMs and cooperative game theory, which are the primary focus of this work.

KV Cache Compression. The memory overhead of storing key-value (KV) pairs for LLM has motivated extensive research on KV cache compression. StreamingLLM [49] preserves the initial and recent tokens, which empirically exhibit higher informativeness during generation. Similarly, Scissorhands [31] proposes the persistence of importance to identify and retain pivotal tokens. H2O [57] employs a heavy-hitter oracle to drop tokens with low attention scores. SnapKV [28] uses the attention scores of the recent tokens to retain critical tokens. CCA-LLM [7] groups input tokens, compressing each group into a core token, which is then combined with recent tokens for attention computation. DGA-LLM [55] aggregates less important tokens while preserving important tokens. While these methods reduce memory usage and accelerate inference, they implicitly assume uniform importance across attention heads, limiting their applicability. Recent works address head diversity through layer-wise and head-wise optimizations. PyramidKV [4] implements a hierarchical allocation strategy, assigning larger cache budgets to lower layers based on the observed attention patterns across layers. FastGen [15] is an adaptive KV cache compression method that reduces LLMs' memory usage by profiling attention modules and constructing caches adaptively. RazorAttention [42] and Duoattention [48] divide attention heads into retrieval heads(critical for long-context processing [46]) and non-retrieval heads, apply full KV cache to retrieval heads and compressed KV cache to non-retrieval heads. ArkVale [6] proposes a page-based KV cache manager that asynchronously copies filled pages into external memory (e.g., CPU memory) as a backup and supports the recall of important tokens that were previously evicted. AdaKV [12] dynamically adjusts cache budgets across heads based on their concentration degrees and HeadKV [13] calculates head importance scores to allocate cache budget before inference. In addition to KV cache eviction methods, KV cache quantization is also one of the mainstream approaches for KV cache compression [32, 52]. However, while eviction methods can be used to retain less than 1% of the cache, KV cache compression cannot be applied to such an extent because it must preserve at least 1 bit. Nevertheless, existing methods lack a global assessment of attention head importance and allocate cache budgets accordingly

Cooperative Game Theory. Cooperative game theory offers a principled framework for understanding how multiple players jointly contribute to system performance. Shapley value [38], a classic solution, allocates benefits based on marginal contributions, though exact calculation is costly. Extensive research has optimized this through approximation [16, 34] and complementary contributions [40, 53]. In the context of LLMs, recent works have applied game theory to analyze attention heads from different perspectives. For instance, Held and Yang [19] employ Shapley values to identify and prune interference-inducing heads for multilingual model adaptation, while Qu et al. [36] leverage Harsanyi dividends to quantify cooperative and competitive interactions within head groups for attention calibration. However, these methods are primarily designed for model adaptation and structural refinement rather than inference-time KV cache management. Specifically, they rely on standard marginal contributions that require evaluating each head independently, leading to computational complexity that scales linearly with the number of heads. This becomes prohibitively expensive for long-context inference, where head importance must be estimated efficiently across thousands of tokens and multiple decoding steps. Moreover, their objectives, which focus on structural pruning and interaction analysis, do not produce the continuous, per-head importance scores needed for fine-grained cache budget allocation. To address this, we propose the *Sliced Shapley value*, which samples only a subset of coalition sizes based complementary contribution. This approach not only accelerates computation to practical levels but also accurately reflects individual head contributions for fine-grained cache allocation.

3 PRELIMINARIES

In this section, we begin by formalizing the problem of key-value caching in multi-head attention (MHA). Then we review two strategies for KV cache compression. Finally, we present the Shapley value, a foundational concept from cooperative game theory.

3.1 Key-Value Caching in MHA

We consider a transformer-based LLM built from stacked layers that integrate multi-head attention mechanisms, and each layer consists of n heads $\mathcal{H} = \{h_1, \cdots, h_n\}$. Let $\mathbf{X} \in \mathbb{R}^{m \times d_{\text{model}}}$ be an input sequence consisting of m embedded tokens of d_{model} dimensions each. The attention output of head h_i is given by

$$\mathbf{O}_i = \mathbf{A}_i \mathbf{V}_i = \text{Softmax}(\mathbf{Q}_i \mathbf{K}_i^\top / \sqrt{d_h}) \mathbf{V}_i,$$

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q, \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K, \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V \in \mathbb{R}^{m \times d_h}$$

where $d_h = d_{\text{model}}/n$ is the dimensionality of each head. $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_h}$ represent the mapping matrices for the query, key, and value, respectively. Note that $\mathbf{A}_i$ is also known as the attention weights.

The objective of KV caching in MHA is to store all previously computed $\mathbf{K}_i$ and $\mathbf{V}_i$ pairs, thereby avoiding redundant recomputation during subsequent decoding stages. Upon the arrival of a new generated token $\mathbf{x} \in \mathbb{R}^{1\times d_{\text{model}}}$, one can first obtain its corresponding query, key, and value as follows:

$$\mathbf{q}_i = \mathbf{x}\mathbf{W}_i^Q, \mathbf{k}_i = \mathbf{x}\mathbf{W}_i^K, \mathbf{v}_i = \mathbf{x}\mathbf{W}_i^V,$$

then update $\mathbf{K}_i$ and $\mathbf{V}_i$ by appending the newly generated $\mathbf{k}_i$ and $\mathbf{v}_i$, respectively

$$\mathbf{K}_i = \text{Cat}[\mathbf{K}_i, \mathbf{k}_i], \quad \mathbf{V}_i = \text{Cat}[\mathbf{V}_i, \mathbf{v}_i]. \tag{1}$$

The attention output vector of head h_i for the new token is computed as

$$\mathbf{o}_i = \text{Softmax}(\mathbf{q}_i\mathbf{K}_i^\top/\sqrt{d_h})\mathbf{V}_i.$$

The output of MHA for the new token, $\mathbf{y} \in \mathbb{R}^{1\times d_{\text{model}}}$, is then obtained by concatenating the output vectors of all heads and applying a linear transformation:

$$\mathbf{o} = \text{Cat}[\mathbf{o}_1, \cdots, \mathbf{o}_n], \quad \mathbf{y} = \mathbf{o}\mathbf{W}^O,$$

where $\mathbf{W}^O \in \mathbb{R}^{d_{\text{model}}\times d_{\text{model}}}$ denotes the weight matrix.

3.2 KV Cache Compression

As the attention heads process an increasing number of tokens, the KV cache expands proportionally, leading to higher memory consumption and computational overhead. To address this while maintaining model performance, we introduce two compression strategies, namely eviction and quantization.

KV Cache Eviction. KV cache eviction methods aim to discard KV pairs whose contributions to the final outputs are marginal. For example, H2O [57] ranks tokens based on the sum of their attention weights across preceding tokens, retaining only those with the highest cumulative scores. Let $\hat{\mathbf{K}}_i \in \mathbb{R}^{s\times d_h}$ and $\hat{\mathbf{V}}_i \in \mathbb{R}^{s\times d_h}$ denote the compressed KV caches, where $s \ll m$ is the number of retained tokens. Then the attention output of h_i is given by $\hat{\mathbf{o}}_i = \text{Softmax}(\mathbf{q}_i\hat{\mathbf{K}}_i^\top/\sqrt{d_h})\hat{\mathbf{V}}_i$, and $\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i$ can be dynamically updated following Eq. (1).

Traditional approaches typically apply uniform cache budgets across all attention heads. However, recent research reveals that attention heads exhibit highly heterogeneous behaviors, with varying importance and functionality. This observation has motivated adaptive cache allocation strategies that dynamically assign different cache budgets to different heads based on their characteristics. Our work follows this direction by proposing a cooperative game-theoretic approach to evaluate the attention head importance and determine optimal cache allocation across heads.

KV Cache Quantization. The uniform quantization to the KV cache process is shown as follows. For each head h_i with bit-width b_i, and for each token position t, we compute a per-token dynamic range along the head dimension. Define the integer range as:

$$q_{\max}(b_i) = \max\left(1, 2^{b_i-1} - 1\right), \quad b_i > 1$$

and the scale as:

$$s_{i,t} = \frac{\max|\mathbf{K}_{i,t}|}{q_{\max}(b_i)}, \quad u_{i,t} = \frac{\max|\mathbf{V}_{i,t}|}{q_{\max}(b_i)}$$

Uniform quantization maps float element $\mathbf{K}_{i,t,j}(1 \le j \le d_h)$ to integers via:

$$\overline{\mathbf{K}}_{i,t,j} = \text{clip}\left(\text{round}\left(\frac{\mathbf{V}_{i,t,j}}{s_{i,t}}\right), -q_{\max}(b_i), q_{\max}(b_i)\right)$$

and for values:

$$\overline{\mathbf{V}}_{i,t,j} = \text{clip}\left(\text{round}\left(\frac{\mathbf{V}_{i,t,j}}{u_{i,t}}\right), -q_{\max}(b_i), q_{\max}(b_i)\right)$$

The dequantization process restores the values by:

$$\tilde{\mathbf{K}}_{i,t,j} = \overline{\mathbf{K}}_{i,t,j} \times s_{i,t}, \quad \tilde{\mathbf{V}}_{i,t,j} = \overline{\mathbf{V}}_{i,t,j} \times u_{i,t}$$

Similarly, the compressed KV caches $\tilde{\mathbf{K}}_i, \tilde{\mathbf{V}}_i$ can be updated according to Eq. (1), and the attention output of h_i is then computed as $\tilde{\mathbf{o}}_i = \text{Softmax}(\mathbf{q}_i\tilde{\mathbf{K}}_i^\top/\sqrt{d_h})\hat{\mathbf{V}}_i$. Since we only cache the quantized matrices $\overline{\mathbf{K}}_i$ and $\overline{\mathbf{V}}_i$, which use b_i bits for storing each element, it can significantly reduce memory usage during model inference. Our approach can extend this by allowing different attention heads to be assigned different bit-widths based on their importance, enabling more fine-grained memory optimization.

3.3 Shapley Value

Consider a set of players $\mathcal{N} = \{p_1, \ldots, p_n\}$. A *coalition* $\mathcal{S}$ is a subset of $\mathcal{N}$ that cooperates to complete a task. A utility function $\mathcal{U}(\mathcal{S})$ is the utility of coalition $\mathcal{S}$ for the task. The *marginal contribution* of player p_i with respect to a coalition $\mathcal{S}$ is $\mathcal{U}(\mathcal{S}\cup\{p_i\}) - \mathcal{U}(\mathcal{S})$, and the Shapley value of p_i measures its expected marginal contribution

$$\mathcal{SV}_i = \frac{1}{n}\sum_{\mathcal{S}\subseteq\mathcal{N}\setminus\{p_i\}} \frac{\mathcal{U}(\mathcal{S}\cup\{p_i\}) - \mathcal{U}(\mathcal{S})}{\binom{n-1}{|\mathcal{S}|}}. \tag{2}$$

It is evident that the exact computation of $\mathcal{SV}_i$ requires evaluating the utility of all possible coalitions, with player-specific marginal contributions that cannot be reused. As a result, the computational cost grows exponentially with the number of players. To overcome this challenge, Zhang et al. [53] proposed using the *complementary contribution* of a coalition $\mathcal{S}$, denoted as $\mathcal{U}(\mathcal{S}) - \mathcal{U}(\mathcal{N}\setminus\mathcal{S})$, and demonstrated that $\mathcal{SV}_i$ can be approximately given by

$$\mathcal{SV}_i = \frac{1}{n}\sum_{\mathcal{S}\subseteq\mathcal{N}\setminus\{p_i\}} \frac{\mathcal{U}(\mathcal{S}\cup\{p_i\}) - \mathcal{U}(\mathcal{N}\setminus(\mathcal{S}\cup\{p_i\}))}{\binom{n-1}{|\mathcal{S}|}}. \tag{3}$$

Note that the complementary contributions can be shared among all players in $\mathcal{S}$, reducing redundant calculations and enhancing the efficiency of Shapley value computation. In the context of KV caches in MHA, attention heads are treated as players for evaluating their importance to each specific task. $\mathcal{U}(\mathcal{S})$ is defined as the model accuracy when the attention heads in $\mathcal{N}\setminus\mathcal{S}$ are masked. We retain only the KV pairs within the local window for masked heads.

4 IMPORTANCE-AWARE KV CACHE COMPRESSION VIA *SLICED SHAPLEY VALUE*

Effectively assessing the importance of attention heads is crucial for effective KV cache compression. However, existing methods treat heads independently, ignoring their collaborative nature. While the Shapley value offers a principled solution, its exact computation is intractable for LLMs due to the exponential number of head coalitions. To address these challenges, we propose a two-step method for efficient KV caching in MHA. We first precompute the

importance scores for each attention head based on the Shapley value. These scores are subsequently utilized for KV cache compression during inference for model application. The overview of our approach is illustrated in Figure 1.

4.1 Head Importance Evaluation

Given a set of heads $\mathcal{H} = \{h_1, \ldots, h_n\}$ and $i, j \in [1, n]$. We first introduce two key concepts that underpin our method.

- j-**coalition**: a subset of $\mathcal{H}$ containing exactly j heads;
- (i, j)-**coalition**: a j-coalition that includes a specific head h_i.

Let $\mathfrak{S}_{i,j} \triangleq \{\mathcal{S} \cup \{h_i\} | \mathcal{S} \subseteq \mathcal{H} \setminus \{h_i\}, |\mathcal{S}| = j - 1\}$ denote a set of (i, j)-coalitions. Following Eq. (3), the Shapley value associated with head h_i corresponds to its expected complementary contribution over all possible (i, j)-coalitions, that is,

$$\mathcal{SV}_i = \frac{1}{n} \sum_{j=1}^{n} \mathcal{SV}_{i,j},$$

$$\text{where } \mathcal{SV}_{i,j} \triangleq \sum_{\mathcal{S} \in \mathfrak{S}_{i,j}} \frac{\mathcal{U}(\mathcal{S}) - \mathcal{U}(\mathcal{H} \setminus \mathcal{S})}{\binom{n-1}{j-1}}, \tag{4}$$

While leveraging complementary contributions can enhance the efficiency of Shapley value approximation, the computation of $\mathcal{SV}_{i,j}$ for every coalition size $j \in [1, n]$ remains prohibitively expensive when n is large.

Our approach is grounded in the key insight that the distribution of $\mathcal{SV}_{i,j}$ across all attention heads remains statistically consistent as the coalition size j varies, as shown in Section 6.4. This observation suggests that the importance of heads can be effectively captured without exhaustively enumerating all coalition sizes, but by considering only a selected subset. We refer to this subset as a *slice* of the coalition size space, denoted by $\mathcal{L} \subseteq [n]$, and define the *sliced Shapley value* ($\mathcal{SSV}$) as follows.

Definition 1. **(Sliced Shapley Value)**
The *sliced Shapley value* of head h_i with respect to $\mathcal{L}$ is defined as the expected complementary contribution over j-coalitions for $j \in \mathcal{L}$:

$$\mathcal{SSV}_i^{\mathcal{L}} = \frac{1}{|\mathcal{L}|} \sum_{j=1}^{n} \mathcal{SV}_{i,j} \cdot \mathbb{I}_j^{|\mathcal{L}|}, \tag{5}$$

where $\mathbb{I}_j^{\mathcal{L}}$ is an indicator function that returns 1 if $j \in \mathcal{L}$ and 0 otherwise.

The intuition of why such stability arises across different coalition sizes is that complementary contribution evaluates binary budget assignments. Varying coalition size j primarily induces a global utility baseline shift, where preserving more heads retains more information, without altering the relative importance of individual heads. Since head-specific sensitivities (e.g., reliance on long-range vs. local context) are intrinsic and independent of j, changing coalition size shifts overall contribution levels but preserves importance rankings. This ensures SSV approximations remain reliable across different choices of j. It can be proven that the error between $\mathcal{SSV}_i^{\mathcal{L}}$ and $\mathcal{SV}_i$ for any attention head h_i is bounded by $O\left(\sqrt{1/|\mathcal{L}|}\right)$, as shown in Theorem 1 below. The detailed proof is provided in Section 5.1.

Algorithm 1: Head Importance Evaluation in LLMs.

input : The set of heads $\mathcal{H} = \{h_1, \ldots, h_n\}$; the set of selected coalition sizes $\mathcal{L}$, number of samples M
output: The approximate *sliced Shapley value* $\overline{\mathcal{SSV}_i^{\mathcal{L}}}$ for each head h_i.

```
for i = 1 to n do
    SSV_i^L ← 0;
    for j = 1 to n do
        SV_{i,j} ← 0; m_{i,j} ← 0;
for k = 1 to M do
    π^k ← a random permutation of {1, ..., n};
    j ← a coalition size randomly selected from L;
    S ← {π^k[1], ..., π^k[j]};
    H \ S ← {π^k[j+1], ..., π^k[n]};
    // U(S) is the model performance when heads in H \ S are masked
       and vice versa for U(H \ S).
    u ← U(S) − U(H \ S);
    foreach i ∈ S do
        SV_{i,j} += u;
        m_{i,j} += 1;
for i = 1 to n do
    SSV_i^L = (1/|L|) Σ_{j=1}^{n} SV_{i,j} / m_{i,j};
return SSV_1^L, ..., SSV_n^L.
```

Theorem 1. *Assume $\mathcal{SV}_{i,j} \in [a, b]$ for all $i, j \in [n]$. For any $\delta \in (0, 1)$, we have*

$$|\mathcal{SV}_i - \mathcal{SSV}_i^{\mathcal{L}}| \leq (b - a) \sqrt{\frac{(n - |\mathcal{L}| + 1) \ln(2/\delta)}{2|\mathcal{L}|n}},$$

with probability at least $1 - \delta$.

Although $\mathcal{SSV}_i^{\mathcal{L}}$ greatly reduces the number of coalitions to be considered, the precise calculation of $\mathcal{SV}_{i,j}$ for each $j \in \mathcal{L}$ remains intractable, as the cardinality of $\mathfrak{S}_{i,j}$ expands exponentially with the total number of heads. To alleviate this, we adopt a Monte Carlo approach that iteratively estimates the expected complementary contribution of each (i, j)-coalition, denoted as $\overline{\mathcal{SV}_{i,j}}$. This enables us to further obtain an approximate sliced Shapley value $\overline{\mathcal{SSV}_i^{\mathcal{L}}}$ for each attention head h_i, as outlined in Algorithm 1. We recommend selecting coalition sizes via a simple interpolation strategy under a fixed compute budget, e.g., $\mathcal{L} = \{n/2\}$, $\mathcal{L} = \{n/2, n/4\}$, or $\mathcal{L} = \{n/2, n/3, n/6\}$, where n is the total number of (KV) heads. If more precomputation is available, one can further enlarge L by adding more coalition sizes. We use interpolation-style selection because adjacent coalition sizes are highly redundant in practice that the estimated head-importance distribution changes smoothly with coalition size, and nearby j values tend to produce very similar rankings. Therefore, evaluating many closely spaced coalition sizes usually provides limited additional information relative to its computation cost, while a few well-spaced sizes can already capture the main trend effectively.

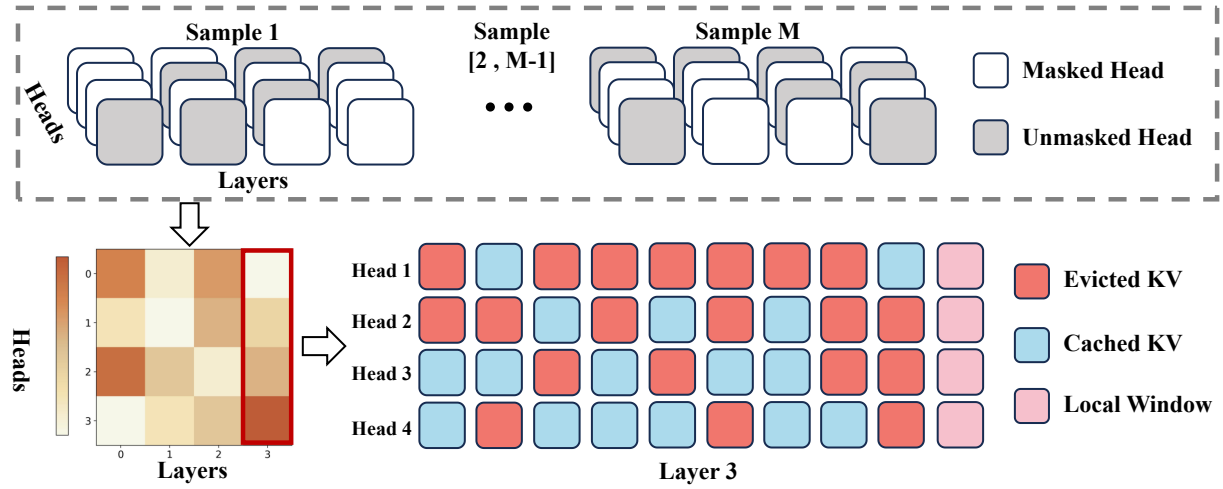

Figure 1: Overview of our proposed method: (1) **Head Importance Evaluation (Upper Part):** For a 4-layer × 4-head model, we measure head importance using the *sliced Shapley value* ($\mathcal{SSV}$). To approximate $\mathcal{SSV}$, we sample M different sets of masked heads and compute their complementary contributions. The average complementary contribution of each head is its estimated $\mathcal{SSV}$. (2) **KV Cache Compression (Lower Part):** Using the 4 heads in Layer 3 and the KV cache eviction method as an example, each head stores KV pairs for a small local window of recent tokens. Heads with higher $\mathcal{SSV}$ (represented by darker areas in the heatmap) are allocated more cache size to retain KV pairs prior to the local window. For adaptive KV cache quantization, we can assign heads with higher $\mathcal{SSV}$ more bits, while heads with lower $\mathcal{SSV}$ receive fewer bits.

Description of Algorithm 1. At k-th iteration, we sample a random permutation π^k of $\{1, \ldots, n\}$, which specifies a random ordering of the heads. We then randomly choose a coalition size $j \in \mathcal{L}$ and construct the set of selected head indices $\mathcal{S}$, which includes the first j elements in π^k, i.e., $\mathcal{S} = \{\pi^k[1], \ldots, \pi^k[j]\}$. Next, we mask all heads with indices in $\mathcal{H} \setminus \mathcal{S}$ and evaluate the resulting model accuracy, denoted as $\mathcal{U}(\mathcal{S})$. Similarly, we obtain $\mathcal{U}(\mathcal{H} \setminus \mathcal{S})$ by masking all heads with indices in $\mathcal{S}$ (Lines 3-6). Subsequently, we update the Shapley value estimate $\mathcal{SV}_{i,j}$ and the count value $m_{i,j}$ for each $i \in \mathcal{S}$ (Lines 7-10). After M iterations are completed, we calculate the approximated *sliced Shapley value* $\overline{\mathcal{SSV}_i^{\mathcal{L}}}$ for all heads by averaging the complementary contributions.

As demonstrated in Theorem 2, our algorithm achieves an (ϵ, δ)-approximation with time complexity $O\left(\frac{T|\mathcal{L}|\ln(2|\mathcal{L}|/\delta)}{\epsilon^2}\right)$, significantly improving over the $O\left(\frac{Tn\ln(2n/\delta)}{\epsilon^2}\right)$ complexity required for the Shapley value approximation. The detailed proof can be found in Appendix C.

Theorem 2. *Algorithm 1 returns an (ϵ, δ)-approximation of sliced Shapley value with time complexity $O(\frac{T|\mathcal{L}|\ln\frac{2|\mathcal{L}|}{\delta}}{\epsilon^2})$ where T is the time cost of evaluating a complementary contribution which is the time to inference on the validation dataset of each task in our setting. In contrast, Shapley value requires the time complexity of $O(\frac{Tn\ln\frac{2n}{\delta}}{\epsilon^2})$ to achieve an (ϵ, δ)-approximation.*

4.2 KV Cache Compression

In this section, we present how our proposed head importance evaluation method can be effectively applied to KV cache compression, with two primary techniques, including KV cache eviction and KV cache quantization. Our contributions focus on the eviction-based approach. It preserves strong model performance with an extremely small cache, such as one with an average of only 128 KV pairs. In contrast a 2-bit quantization method still requires 12.5% of the KV cache and delivers much worse performance. Nevertheless, we demonstrate that the same importance scores can be seamlessly integrated into quantization frameworks and that this application shows the generality of our method.

KV Cache Eviction Budget Allocation. An intuitive approach suggests that the least important heads, which contribute minimally or even negatively to the model performance, may not require cache allocation. Let α represent the number of such heads, which serves as the sole hyperparameter in our allocation scheme. For the remaining $n - \alpha$ heads, we employ a normalization method to normalize their importance scores and allocate the cache size proportionally based on their normalized scores.

Specifically, we normalize their contributions using min-max normalization for the $n - \alpha$ heads:

$$\mathcal{NSV}_i^{\mathcal{L}} = \frac{\mathcal{SSV}_i^{\mathcal{L}} - \min^{\alpha}(\mathcal{SSV}^{\mathcal{L}})}{\max(\mathcal{SSV}^{\mathcal{L}}) - \min^{\alpha}(\mathcal{SSV}^{\mathcal{L}})},$$

where $\min^{\alpha}(\cdot)$ and $\max(\cdot)$ extract the α-th smallest and maximum value, respectively. For the α heads with the smallest *sliced Shapley values*, we set the normalized score as 0. This ensures that all normalized scores lie in the range $[0, 1]$.

Next, the cache size c_i allocated to head h_i is determined by the local window size s and linearly distributing the remaining shared cache size B based on the normalized scores:

$$c_i = B \cdot \frac{\mathcal{NSV}_i^{\mathcal{L}}}{\sum_{j=1}^{n} \mathcal{NSV}_j^{\mathcal{L}}} + s. \tag{6}$$

Algorithm 2: Token Eviction Using CoKV.

input : Shared budget size B, local window size s, tokens in local window $\mathbf{X}^{win} \in \mathbb{R}^{s\times d}$, KV in local window $\{\mathbf{K}_i^{win}, \mathbf{V}_i^{win}\}$, KV outside local window $\{\mathbf{K}_i^{out}, \mathbf{V}_i^{out}\}$

output : Retained KV Cache $\{\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i\}$

$\mathbf{Q}_i^{win} = \mathbf{X}^{win}\mathbf{W}_i^{Q}$;

// Compute attention weights of queries in local window and prefix Keys.

$\overline{\mathbf{A}}_i = \text{Softmax}(\mathbf{Q}_i^{win}\mathbf{K}_i^{\top})$;

$\overline{\mathbf{A}}_i = \overline{\mathbf{A}}_i.maxpooling(dim = 1).mean(dim = 0)$;

// Calculate token scores outside the local window.

Get the cache size c_i using Algorithm 1 and Equation 6;

$indices = \overline{\mathbf{A}}_i.topk(c_i).indices$;

Select $\{\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i\}$ from $\{\mathbf{K}_i^{out}, \mathbf{V}_i^{out}\}$ according *indices*;

$\{\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i\} = \text{Cat}(\{\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i\}, \{\mathbf{K}_i^{win}, \mathbf{V}_i^{win}\})$;

// Keep top c_i KV pairs in the cache.

return Retained KV Cache $\{\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i\}$.

Our method uses Sliced Shapley values as a proxy to guide allocation. This approach works well in practice for two reasons: (i) By definition, a head with low Shapley value indicates that compressing or removing the head's KV cache has minimal impact on model utility. (ii) Over-allocating budget to important heads incurs limited inefficiency compared to the performance degradation caused by under-allocating them. As long as critical heads are protected (i.e., ensured to receive sufficient KV budget), overall model performance is largely preserved.

Algorithm Description. The detailed KV cache eviction steps for a single head are outlined in Algorithm 2. First, we allocate the KV cache size for each head based on their normalized *sliced Shapley values*. Next, we rank the importance of KV pairs within each head following SnapKV. Specifically, the most recent tokens within local windows guide the KV cache selection. Attention scores from these local windows to the remaining tokens are aggregated via pooling, with higher-scoring tokens retained in the cache for each head.

KV Cache Quantization. The head importance scores derived by our method are not limited to eviction and can be directly applied to guide non-uniform quantization strategies. Specifically, our scores enable an adaptive bit allocation scheme where more important heads are assigned higher precision. This principle can complement advanced quantization techniques like Kvquant [21], allowing for a head-aware quantization policy that operates on top of their sophisticated per-channel methods. In our experiments, we demonstrate that a simple integration, which allocates bits proportionally to our importance scores, consistently outperforms baselines that use alternative head importance metrics under the same average bit-width. This validates the general utility of our cooperation-based evaluation framework across different compression paradigms.

Compatibility with Paged Attention.

CoKV operates at the logical level of KV cache management (determining per-head retention budgets), which is orthogonal to the physical page-based memory layout in systems like vLLM. CoKV's budget decisions translate directly to page allocation counts, and its eviction policy functions independently of memory contiguity, enabling seamless integration with paged attention without modifying core paging mechanisms.

5 PROOFS

This section provides the proof of Theorem 1. The proof of Theorem 2 can be found in Appendix C due to the limited space.

5.1 Proof of Theorem 1

Without loss of generality, we assume $\mathcal{SV}_{i,j} \in [a, b]$ for $j \in [n]$. The sum of $\mathcal{SV}_{i,j}$ for all $j \in [n]$ is

$$U = \sum_{j=1}^{n} \mathcal{SV}_{i,j} = n \cdot \mathcal{SV}_i.$$

For a randomly sampled coalition size subset $\mathcal{L} \subseteq [n]$ of size $|\mathcal{L}|$ drawn without replacement, the sum of $\mathcal{SV}_{i,j}$ for all $j \in \mathcal{L}$ is

$$Z = \sum_{j\in\mathcal{L}} \mathcal{SV}_{i,j} = |\mathcal{L}| \cdot \mathcal{SSV}_i^{\mathcal{L}}.$$

Let $R = b - a$. By applying Serfling's inequality for sampling without replacement (Corollary 1.1 in Serfling [37]), we have that for any $\epsilon \geq 0$,

$$P(Z - \frac{|\mathcal{L}|}{n} \cdot U \geq |\mathcal{L}|\epsilon) \leq \exp\left[-\frac{2|\mathcal{L}|\epsilon^2 n}{(n - |\mathcal{L}| + 1)R^2}\right].$$

Dividing both sides of the probability event by $|\mathcal{L}|$, we obtain

$$P(\mathcal{SSV}_i^{\mathcal{L}} - \mathcal{SV}_i \geq \epsilon) \leq \exp\left[-\frac{2|\mathcal{L}|\epsilon^2 n}{(n - |\mathcal{L}| + 1)R^2}\right].$$

By symmetry, we have

$$P(|\mathcal{SSV}_i^{\mathcal{L}} - \mathcal{SV}_i| \geq \epsilon) \leq 2\exp\left[-\frac{2|\mathcal{L}|\epsilon^2 n}{(n - |\mathcal{L}| + 1)R^2}\right]$$

Setting the right-hand side to δ and solving for ϵ:

$$2\exp\left[-\frac{2|\mathcal{L}|\epsilon^2 n}{(n - |\mathcal{L}| + 1)R^2}\right] = \delta,$$

$$\ln(2) - \frac{2|\mathcal{L}|\epsilon^2 n}{(n - |\mathcal{L}| + 1)R^2} = \ln(\delta)$$

$$\epsilon^2 = \frac{R^2}{2|\mathcal{L}|n}(n - |\mathcal{L}| + 1)\ln\left(\frac{2}{\delta}\right).$$

Thus, with probability at least $1 - \delta$:

$$|\mathcal{SV}_i - \mathcal{SSV}_i^{\mathcal{L}}| \leq R\sqrt{\frac{(n - |\mathcal{L}| + 1)\ln(2/\delta)}{2|\mathcal{L}|n}}.$$

It is clear that $\frac{(n-\mathcal{L}+1)}{n} \leq 1$ as $\mathcal{L} \geq 1$. Therefore, we have

$$|\mathcal{SV}_i - \mathcal{SSV}_i^{\mathcal{L}}| \leq R\sqrt{\frac{\ln(2/\delta)}{2|\mathcal{L}|}}.$$

By omitting the constants, we obtain the asymptotic error bound $O(\sqrt{(}1/|\mathcal{L}|)$. This completes the proof.

6 EXPERIMENTS

This section presents a comprehensive evaluation of CoKV, demonstrating its advantages over baselines in KV cache eviction across multiple datasets. We further evaluate decoding latency and memory usage in Section 6.7; additional KV quantization and precomputation results are provided in Appendix B, which is included in the full version of our paper available on GitHub.

6.1 Experiment Settings

Datasets and Models. We evaluate our methods on five open benchmark datasets: LongBench [3], Needle In A Haystack [25], RULER [22], GSM8K [8], and MATH [20] using `Qwen3-32B` [51], Llama-3-8B-Instruct [11] (hereafter `Llama3-8B`) and Mistral-7B-Instruct-v0.2 [24] (hereafter `Mistral-7B`) models. Due to the page limit, details of all experimental datasets are provided in the appendix.

Baselines and Settings. We compare our CoKV against four KV cache compression methods. For fair comparison, all methods are implemented using the same underlying components, i.e., the grouped-query attention (GQA) [1] architecture and the FlashAttention [9] kernel, with an identical total cache size. Notably, within the GQA design, a group of heads shares a common KV cache.

- **SnapKV** [28] uses the last several tokens as local windows. Attention scores from these windows to the remaining tokens are pooled to guide the KV selection in each head.
- **PyramidKV** [4] allocates more KV caches to lower layers to retain key information while reducing the budget for higher layers where information is already aggregated.
- **Ada-KV** [12] dynamically allocates budgets to heads within each layer based on their concentration degrees, and can be combined with SnapKV or PyramidKV. Ada-SnapKV is used as the baseline due to its superior performance over Ada-PyramidKV.
- **HeadKV-R2** [13] allocates budgets to heads based on their retrieval-reasoning score, and uses SnapKV to rank the importance of KV pairs within each head. For compatibility with GQA, we instead compute the importance score of each group by averaging the retrieval-reasoning scores of its constituent heads.

Following standard practices in Ada-KV and HeadKV-R2, we perform cache eviction after the prefilling phase of each layer for consistent comparison. When evaluating HeadKV-R2, we compute head scores for the `Qwen3-32B` model using the maximum context length supported by an H100 96G GPU server. For `Llama3-8B` and `Mistral-7B`, we use the head scores reported in the original paper [13]. Furthermore, in all `Llama3-8B` experiments, only the first and last 4k tokens are used if the test data exceeds the maximum input length of the model.

CoKV Settings. Given the adoption of the GQA architecture, we treat each group of heads as a player in the cooperative game to evaluate their *sliced Shapley value*. We set $\mathcal{L} = \{32, 64, 96, 128\}$ for both `Llama3-8B` and `Mistral-7B` which have 256 groups (32 layers, 8 groups in each layer), and set $\mathcal{L} = \{256\}$ for `Qwen3-32B` which has 512 groups (64 layers, 8 groups in each layer). Our ablation experiments of $\mathcal{L}$ in Section 6.3 show that CoKV with only one coalition size works well, but more slices will enhance CoKV. Following HeadKV-R2, we set the local window size $s = 8$. We randomly constructed a validation set by selecting 20 data instances from each task of LongBench to compute the *sliced Shapley value* of each task, with the remaining data serving as the test set. We use the average *sliced Shapley value* of all tasks on Longbench as a general head importance score to assess model performance on the LongBench test set and four other datasets of long context retrieval and math reasoning, which confirms the strong generalizability of our approach. The optimal hyperparameter α for `Qwen3-32B` is determined through a grid search over the candidate values $\{5, 10, 15, 20, 30, 40, 50\}$ on the validation dataset, while the candidate values for `Llama3-8B` are $\{2, 4, 6, 8, 10, 12, 14\}$. The optimal hyperparameter α for `Llama3-8B` and `Qwen3-32B`, determined through grid search, is 8 and 40, respectively. Detailed results are provided in Section B.1 in the appendix. We also test a task-specific variant, termed CoKV-Task, which uses the *sliced Shapley value* computed on each task's own validation data. This approach allows CoKV to be adapted to domain-specific scenarios with less reliance on validation data.

6.2 KV Cache Eviction Results

LongBench Results. Longbench is a widely adopted benchmark for evaluating model performance. It comprises 16 tasks from diverse domains, with context lengths ranging from 1k to 18k tokens. In our study, we compared various KV cache eviction methods under the same constrained KV cache budget. A well-designed eviction method is capable of retaining most of the model's performance even when storing only a very small portion of KV pairs. For instance, experimental results with CoKV on the `Llama3-8B` model demonstrate that by keeping 128 KV pairs per cache, which represents only an average 1.6% of the full cache size, the model still achieves 96.13% of the performance attained with the full KV cache. The complete benchmark results are presented in Tables 16 and 17 in the appendix. We include a simplified table (Table 1), showing the performance of `Llama3-8B` and `Qwen3-32B` when keeping 64 KV pairs on average for `Llama3-8B` and 128 KV pairs on average for `Qwen3-32B`. The results demonstrate that CoKV consistently outperforms all baseline methods. The superior performance of CoKV arises from its ability to effectively evaluate the importance of each cache within a group while considering the cooperation among all groups. CoKV-Task outperforms CoKV on the `Llama3-8B` model. We attribute this superiority to its task-specific hyperparameter optimization and the model's smaller size, which simplifies head importance assessment. However, for the larger and more complex `Qwen3-32B` model, CoKV's multi-task evaluation demonstrates superior performance by offering a more robust measure of overall head importance.

Since both HeadKV-R2 and CoKV provide importance scores for each group, we conduct an experiment to compare their effectiveness without introducing any hyperparameters for fair evaluation. In this experiment, we mask groups based on their importance scores assigned by each algorithm. Specifically, we mask the highest-ranked (**top**) and lowest-ranked groups (**low**) for both methods. Table 2 presents the simplified results of masking groups of `Qwen3-32B` and `Llama3-8B`, and the complete results are shown in Tables 18, 19 and 20 in the appendix. The results show that

Table 1: Comprehensive KV Cache Eviction Results on LongBench

Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Llama3-8B model with KV size=64																	
Full Cache	23.37	30.82	40.25	42.9	33.03	22.04	28.46	23.07	26.8	75.0	90.41	41.94	3.11	71.67	56.82	52.68	41.39
DuoAttention(12.5%)	19.27	11.55	28.59	36.69	28.22	17.43	18.01	21.55	16.40	48.99	89.14	36.25	3.07	72.75	54.24	50.77	35.01
SnapKV	18.81	12.28	29.72	37.32	28.58	17.93	17.45	21.86	17.07	49.44	88.77	35.9	3.0	71.67	54.12	50.92	34.67
Pyramid	19.64	14.06	33.0	38.6	26.16	18.52	18.01	21.88	18.57	62.78	87.46	36.76	3.31	71.39	53.06	49.4	35.78
Ada-SnapKV	18.7	14.03	32.96	39.51	29.58	18.14	18.15	21.53	18.7	60.56	89.07	35.74	3.61	70.93	53.25	51.79	36.01
HeadKV-R2	19.46	16.14	36.67	34.83	25.32	19.09	18.70	21.80	20.18	64.93	88.15	37.76	2.94	71.67	57.08	55.90	36.92
CoKV-Task	20.02	19.40	35.71	42.76	33.02	20.59	17.92	22.31	18.49	70.0	90.16	37.96	3.98	71.67	55.92	56.28	38.51
CoKV	19.92	23.67	32.14	42.33	33.8	17.27	16.26	22.67	16.16	68.89	89.34	37.19	3.89	71.94	51.81	49.68	37.30
Qwen3-32B model with KV size=128																	
Full Cache	37.14	45.51	49.17	58.2	54.74	38.36	32.9	23.72	25.08	72.78	72.57	37.95	17.78	100.0	62.72	70.08	49.92
SnapKV	30.18	32.54	41.96	55.2	47.04	34.31	22.74	21.19	19.0	47.22	67.89	36.77	17.22	98.89	58.52	50.48	42.65
Pyramid	27.17	32.11	40.93	30.13	37.2	34.43	22.01	20.95	18.71	38.1	68.43	35.55	9.52	100.0	56.48	51.29	40.19
Ada-SnapKV	22.74	32.11	40.02	32.42	40.44	27.97	23.43	21.67	18.98	46.67	68.43	38.93	9.52	100.0	57.9	49.71	39.43
HeadKV-R2	29.78	33.39	41.51	51.85	52.06	36.17	24.17	21.04	19.03	48.89	67.98	36.44	18.02	99.44	56.88	52.47	43.07
CoKV-Task	28.73	37.56	44.80	53.48	53.07	35.17	23.89	21.34	19.28	54.44	69.35	38.23	19.22	100.0	57.13	56.87	44.53
CoKV	31.19	37.84	46.1	57.13	49.04	37.68	23.77	22.04	20.98	54.44	77.6	34.9	13.89	100.0	64.7	56.4	45.48

Table 2: Comprehensive Masking Top Important Heads Results on LongBench

Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Masking Top 16 heads with the Llama3-8B model																	
Full Cache	23.37	30.82	40.25	42.9	33.03	22.04	28.46	23.07	26.8	75.0	90.41	41.94	3.11	71.67	56.82	52.68	41.39
Random	20.27	28.69	32.38	42.36	20.02	21.46	28.18	23.38	26.19	75.0	90.45	40.46	3.52	71.3	48.71	41.11	38.34
HeadKV-R2(low)	20.99	14.11	32.45	28.81	27.5	13.19	27.58	23.42	26.86	75.0	90.38	41.86	3.98	70.69	36.39	38.61	35.73
HeadKV-R2(top)	18.62	11.64	27.87	31.41	17.77	15.17	19.86	22.58	22.04	68.89	87.19	35.8	3.94	68.31	27.83	28.1	31.68
CoKV(low)-Task	22.72	33.84	41.49	46.15	39.5	20.7	29.13	22.98	27.49	75.0	90.81	42.52	3.89	70.57	63.17	61.96	43.24
CoKV(top)-Task	6.17	9.54	9.3	9.84	12.55	5.39	5.69	17.04	4.43	42.78	69.21	22.75	3.58	34.24	11.84	18.34	17.66
CoKV(low)	22.54	34.66	41.88	45.62	39.59	22.33	28.2	22.98	26.86	75.0	90.14	41.39	3.89	62.41	48.62	48.8	40.93
CoKV(top)	15.94	6.16	14.4	10.62	11.39	4.56	8.13	21.4	15.14	72.78	77.07	34.06	3.49	43.44	20.89	22.01	23.84
Masking Top 64 heads with the Qwen3-32B model																	
Full Cache	37.14	45.51	49.17	58.2	54.74	38.36	32.9	23.72	25.08	72.78	72.57	37.95	17.78	100	62.72	70.08	49.92
Random	30.13	44.81	50.07	56.51	54.06	39.24	27.33	23.32	24.82	72.78	71.12	37.72	12.94	100.0	22.71	23.79	43.21
HeadKV-R2(top)	28.38	34.55	32.15	47.25	45.26	25.33	20.14	22.17	13.98	55.56	56.69	21.18	8.69	98.33	14.44	18.47	33.91
CoKV(low)-Task	22.72	33.84	41.49	46.15	39.5	20.7	29.13	22.98	27.49	75.0	90.81	42.52	3.89	70.57	63.17	61.96	43.24
CoKV(top)-Task	28.85	28.55	18.91	25.1	19.78	12.69	13.48	22.97	23.8	35.56	26.43	10.41	8.04	17.22	5.2	5.05	18.88
CoKV(low)	34.64	46.54	54.31	56.15	55.69	42.36	22.11	24.67	24.98	69.7	74.74	38.28	15.15	100.0	21.95	26.42	44.23
CoKV(top)	12.07	16.96	7.72	19.81	16.13	4.72	14.47	12.14	4.72	34.34	22.01	8.58	14.14	1.01	15.76	14.37	13.68

when masking the top-ranked groups identified by each method, the performance of CoKV degrades more significantly than that of HeadKV-R2. This suggests that CoKV is more effective at evaluating group importance, as it better distinguishes between critical and non-critical caches. Conversely, the results in the full tables show that when masking the unimportant groups (**low**), the performance of CoKV declines more gradually than HeadKV-R2. When masking the 64 most important heads of Qwen3-32B, CoKV achieves an average accuracy of only 13.68%, while HeadKV-R2 maintains 33.91%. Surprisingly, the results of masking 16 most unimportant groups of Llama3-8B in Table 2 outperformed the FullKV approach. This further demonstrates that CoKV can identify attention head groups that have a negative impact on the model. By removing the KV pairs from these groups, the model inference not only optimizes

Table 3: Qwen3-32B with KV size = 128 on GSM8K and MATH

Dataset	Full Cache	SnapKV	Pyramid	AdaKV	HeadKV-R2	CoKV
GSM8K	93.47	70.71	69.56	68.69	79.44	82.10
MATH	52.53	39.10	37.58	41.02	43.65	46.17

Table 4: Mask Top Important Heads on GSM8K and MATH

Dataset	Method	16 heads	32 heads	64 heads	96 heads
GSM8K	Random	92.24	91.73	89.57	88.15
	HeadKV-R2(top)	91.24	90.38	72.15	36.85
	CoKV(top)	86.57	19.40	3.77	1.89
MATH	Random	52.15	51.06	46.97	42.39
	HeadKV-R2(top)	51.41	36.54	32.83	10.61
	CoKV(top)	45.45	21.37	2.92	1.70

storage and decoding speed but also enhances overall performance. Results for Mistral-7B are available in the appendix.

Mathematical Reasoning Evaluation. To assess the mathematical reasoning capability of CoKV, we evaluate it on the widely used GSM8K and MATH datasets using Qwen3-32B. The used head importance scores are still the average scores across all tasks in LongBench. We perform experiments involving both KV cache eviction and head masking, using 5 shots in all the experiments. As shown in Table 3, CoKV outperforms all other compression baselines, achieving the highest accuracy on both GSM8K and MATH. Furthermore, Table 4 demonstrates that masking the top heads identified by CoKV results in a more severe performance drop, further validating its precise assessment of head importance. Most importantly, the fact that the head importance scores computed from the LongBench validation set effectively transferred to mathematical reasoning tasks strongly demonstrates CoKV's capability to identify universally important heads across different domains. This validates the practical utility of CoKV.

Needle-in-a-Haystack and RULER Results. A critical concern regarding KV cache eviction methods is their stability in super-long-context retrieval tasks. To evaluate the stability of our proposed CoKV method under such conditions, we conducted tests using the Needle-in-a-Haystack benchmark and RULER, employing both the Mistral-7B and Qwen3-32B models. Notably, the Llama3-8B instruct model is omitted due to its limited 8k context length. We used the head scores computed on LongBench, which demonstrates that CoKV provides a general ability to measure head importance across different tasks. With the average KV cache size 128, we systematically insert target texts (needles) at ten equidistant positions (11%, 22%, . . . , 100%) across varying context lengths ranging from 1,000 to 31,000 tokens (in 1,000-token increments) for Mistral-7B, as its context length is 32k. As shown in Table 5, CoKV outperforms other baseline methods, achieving an average score of 95.89% - the closest performance to the FullKV baseline. We further extend our evaluation to the Qwen3-32B model under the same KV cache budget (average size = 128). Needles are inserted at the same ten relative positions across context lengths ranging from 1,000 to 61,000 tokens (in 4,000-token increments). CoKV again demonstrates superior performance, attaining an average accuracy of 73.86%—the highest among all compressed methods and the closest to the FullKV benchmark. The detailed evaluation results for both models are provided in Figure 15 in the appendix.

Table 5: KV Compression Methods with KV size = 128 on NIAH for Different Models

Model	Full Cache	SnapKV	Pyramid	AdaKV	HeadKV-R2	CoKV
Mistral-7B	99.65	92.38	93.14	93.20	95.78	95.89
Qwen3-32B	100.0	55.11	52.33	61.69	73.18	73.86

RULER generates examples to evaluate long-context language models with configurable sequence length and task complexity. RULER includes four task categories, and we select the representative task from each category for our assessment. We set the masking group size to 64 and test performance across various context lengths (8k, 16k, and 31k). We compare CoKV with HeadKV-R2 because HeadKV-R2 is not only the strongest baseline method but also provides per-head importance scores. We use Mistral-7B and Qwen3-32B in this experiment, which supports a 32k-token context window. As shown in Table 6, our method shows significant advantages over baseline approaches: masking critical groups (top) leads to substantially larger drops in performance compared with other methods. We present detailed results in Table 14 including masking less important groups (low) in the appendix. These results indicate that CoKV reliably differentiates between heads that are essential for long-context comprehension and those that are irrelevant.

Table 6: Evaluation results of masking 64 groups on the RULER benchmark

Method	Retrieval	Tracing	Aggregation	QA	
	niah	vt	fwe	qa1	qa2
Mistral-7B 31k Context Length					
Full Cache	100.0	86.08	89.2	71.4	53.4
Random	0.8	59.6	54.4	53.6	19.4
Headkv-R2(top)	0.0	0.48	0.0	28.6	13.8
CoKV(top)	0.0	0.12	0.0	12.8	7.0
Mistral-7B 16k Context Length					
Full Cache	100.0	90.44	94.73	76.8	54.6
Random	0.2	6.92	71.67	63.8	52.4
Headkv-R2(top)	0.0	0.56	0.0	29.2	17.0
CoKV(top)	0.0	0.76	0.27	15.4	7.2
Qwen3-32B 31k Context Length					
Full Cache	98.0	99.0	90.0	83.0	67.0
Random	73.68	10.0	44.67	72.0	58.0
Headkv-R2(top)	0.0	7.4	1.33	11.0	14.0
CoKV(top)	0.0	0.0	0.0	2.0	4.0
Qwen3-32B 16k Context Length					
Full Cache	93.0	100.0	94.0	86.0	65.0
Random	88.0	88.4	29.33	80.0	59.0
Headkv-R2(top)	0.0	3.6	0.33	19.0	18.0
CoKV(top)	0.0	0.0	0.0	5.0	9.0

6.3 Ablation Study

We analyze the number of coalition sizes in $\mathcal{L}$ using the `Llama3-8B` model. We conducted experiments by masking the 16 most important attention heads (top) and the 16 least important heads (low) based on their significance scores. This approach eliminates the need for hyperparameter tuning, ensuring a fair comparison between CoKV and HeadKV by removing potential fluctuations caused by parameter selection. We compare the results for $\mathcal{L} = \{32, 64, 96, 128\}$ with $\mathcal{L} = \{128\}$. The results show that using the expected complementary contribution of a single coalition size can still outperform the baselines, and CoKV is more effective with a larger number of coalition sizes. The results are shown in Table 7. In particular, the single-slice setting already identifies high-impact heads whose removal causes a clear performance drop, indicating that a representative coalition size carries a strong importance signal. Using multiple slices averages this signal across budget regimes, improving robustness to coalition-size selection with only modest additional precomputation cost.

6.4 Coalition Distribution of *SSV*

In Appendix Figures 19 and 20, we present the distributions of the expected complementary contributions of heads in `Llama3-8B` model on the *hotpotqa* dataset (multi-document question answering) and the *lcc* dataset (code generation), with coalition sizes of {32, 64, 96, 128, 160, 192, 224}. We observe strong correlations in the distributions across all coalition sizes. Additionally, the distributions of the expected complementary contributions for coalition sizes S and $n - |S|$ are nearly identical. These observations provide a justification for our approach of computing complementary contributions using only a small subset of coalition sizes, as it effectively captures the contributions of the heads.

As showing all the distribution of all datasets costs too much pages, we conduct additional experiment analysis comparing the averaged complementary contributions of small coalitions (j=32,64,96) and large coalitions (j=160,192,224) by measuring the overlap rate of top-contributing heads between them in 16 datasets in LongBench. Specifically, we computed the percentage of shared heads in their respective top-k lists (k=32,64,128). The results show consistently high overlap rates (averaging over 85%) across all 16 datasets and two different LLMs, confirming that the distributional similarity is not model- or dataset-specific. This pattern suggests an underlying structural property of attention heads in transformer-based LLMs, where the complementary contribution of heads remains stable across different coalition scales. The results are shown in Tables 8.

Furthermore, we perform pairwise comparisons between the average importance vectors computed at different coalition sizes $j \in \{32, 64, 96, 128\}$, using (i) Spearman rank correlation and (ii) Top-25 head overlap. We observe consistently positive and moderate-to-high Spearman correlations across coalition sizes. For LLaMA-8B, Spearman ρ ranges from 0.48 to 0.92 across all coalition-size pairs (with the highest correlation between $j = 96$ and $j = 128$, $\rho = 0.92$). For Mistral-7B, Spearman ρ ranges from 0.52 to 0.78. Moreover, Top-25 overlaps are substantial: 0.48–0.76 for LLaMA-8B and 0.48–0.80 for Mistral-7B. These results provide direct empirical evidence that head-importance *rankings and top-head sets* are relatively stable with respect to coalition-size selection, supporting the robustness of our SSV-based approximation and reducing sensitivity to the choice of L.

6.5 *SSV* Stability Analysis

We evaluate whether precomputed head importance remains stable during decoding and across tasks using `Qwen3-32B`. Using reference-continuation teacher forcing, we compute step-wise SSV scores at steps {1, 5, 10, 20, 40}. As shown in Fig. 9 in the appendix, adjacent-step Spearman correlations range from 0.67–0.90, and Top-100 overlap exceeds 55% as shown in Fig. 14, indicating that the relative ordering of important heads is largely preserved. We further compare head-importance vectors across tasks. Instead of relying only on rank-level agreement, we validate cross-task stability through downstream transfer. Although task-specific deviations exist, averaging SSV scores preserves reusable head-importance signals and yields robust performance on out-of-domain GSM8K, Math, NIAH, and RULER datasets. This suggests that CoKV captures a shared structural prior over head utility, enabling practical task-agnostic deployment without per-request re-estimation. Due to limited space, we put the detailed stability analysis including setting and results in the appendix section B.2.

6.6 Head- and Layer-wise Budget Allocation

we analyze how CoKV allocates KV-cache budgets across heads and layers using Qwen3-32B and the precomputed LongBench head-importance scores. We consider a head is non-trivial if its allocated cache budget exceeds the uniform allocation. Under average = 1024 KV, the number of head-groups (out of $64 \times 8 = 512$) with budget > 1024 is 243, i.e., about 47.4% of heads receive above-uniform budgets. Although many heads retain non-zero budgets, the allocation is clearly concentrated on high-importance heads. On average, the top-100 head-groups (about 19.5% of 512) account for 37.4% of the total allocated budget. Similarly, top-20 account for 10.1% and top-60 account for 24.9%. This provides an interpretable view of how CoKV progressively prioritizes a subset of important heads as the budget tightens. We further summarize layer-wise allocation by aggregating token budgets within each layer and counting how many head-groups per layer exceed the uniform baseline. We observe that CoKV tends to allocate relatively more budget to deeper layers (e.g., layers around 50–60 show higher average allocated tokens and higher above-uniform counts), indicating a non-uniform depth structure that differs from layer-only heuristics. These statistics explicitly characterize how CoKV behaves as a structured cache allocator that it maintains broad coverage while concentrating additional capacity on a smaller set of important heads and deeper layers.

6.7 Decoding Latency and Memory Usage

We conduct experiments using the `Qwen3-32B` model, which supports a maximum context window of 128k tokens with YaRN [35], with FlashAttention enabled as the default setting. All compressed methods are executed on a single H100 96GB GPU, whereas FullKV requires two H100 96GB GPUs for the longest context setting. We set the average KV cache size to 128 tokens for all KV cache eviction methods.

Table 7: Ablation Study on LongBench with `Llama3-8B`

Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NtrQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Full Cache	23.37	30.82	40.25	42.9	33.03	22.04	28.46	23.07	26.8	75.0	90.41	41.94	3.11	71.67	56.82	52.68	41.39
Random	20.27	28.69	32.38	42.36	20.02	21.46	28.18	23.38	26.19	75.0	90.45	40.46	3.52	71.3	48.71	41.11	38.34
HeadKV-R2(low)	20.99	14.11	32.45	28.81	27.5	13.19	27.58	23.42	26.86	75.0	90.38	41.86	3.98	70.69	36.39	38.61	35.73
HeadKV-R2(top)	18.62	11.64	27.87	31.41	17.77	15.17	19.86	22.58	22.04	68.89	87.19	35.8	3.94	68.31	27.83	28.1	31.68
CoKV(coalition 128)(low)	21.95	33.58	39.53	44.25	37.45	20.66	25.92	23.4	27.01	75.0	90.04	40.14	3.95	69.78	31.15	34.26	38.62
CoKV(coalition 128)(top)	19.91	9.36	32.55	19.44	18.53	10.31	16.73	22.23	20.62	74.44	85.89	36.0	3.89	66.37	36.08	28.9	31.32
CoKV(low)	22.54	34.66	41.88	45.62	39.59	22.33	28.2	22.98	26.86	75.0	90.14	41.39	3.89	62.41	48.62	48.8	40.93
CoKV(top)	15.94	6.16	14.4	10.62	11.39	4.56	8.13	21.4	15.14	72.78	77.07	34.06	3.49	43.44	20.89	22.01	23.84

Table 8: Overlap Results of `Llama3-8B`

Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NtrQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Top 32 heads	0.81	0.66	0.81	0.91	0.84	0.81	0.62	0.66	0.50	0.84	0.84	0.59	0.97	0.97	0.81	0.75	0.77
Top 64 heads	0.77	0.81	0.88	0.88	0.91	0.91	0.67	0.72	0.66	0.88	0.77	0.80	0.98	0.98	0.83	0.86	0.83
Top 128 heads	0.82	0.88	0.80	0.93	0.90	0.95	0.83	0.83	0.80	0.95	0.84	0.80	0.99	0.99	0.88	0.85	0.88

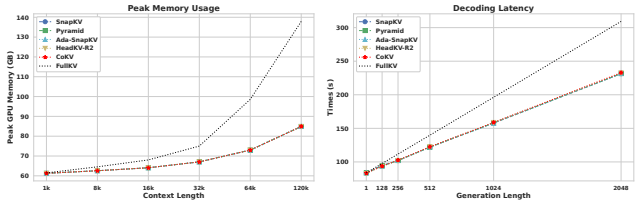

Figure 2: Peak GPU memory usage and decoding latency on `Qwen3-32B`.

Peak Memory Usage. Under a fixed generation length of 1 token, we measure peak GPU memory usage, including model parameters and runtime states, across input contexts from 1k to 120k tokens. As shown in Figure 2, CoKV reduces memory usage by 38.4% compared with FullKV at 120k input length. Notably, CoKV can accommodate 120k-token inputs on a single H100 GPU, whereas FullKV supports fewer than 60k tokens in this setting. We use 1-token generation because KV cache eviction occurs after prefill, meaning the peak memory usage of CoKV is reached before the first token is generated.

Decoding Latency. With a fixed input context length of 120k tokens, we measure decoding latency, including both prefilling time and decoding time, across generation lengths from 1 to 2048 tokens. While CoKV maintains a time-to-first-token comparable to baselines that do not evict KV cache after prefill, it significantly reduces per-token decoding latency and thus overall generation time. As shown in Figure 2, CoKV uses less than 25.14% of the total latency of FullKV, with negligible differences among compressed methods under different KV budgets.

7 CONCLUSION

Large language models (LLMs) face significant challenges in inference cost due to the excessive memory and latency overhead associated with the growing size of the KV cache. Existing adaptive KV cache methods lack a global perspective on attention head importance, overlooking how their utility arises from interactions within the LLM. To this end, we introduce CoKV, a novel method designed to evaluate the collaborative importance of attention heads and dynamically allocate cache sizes based on *sliced Shapley value*. Our experimental results demonstrate that CoKV achieves state-of-the-art performance across 16 LongBench datasets, as well as on the NIAH, RULER, and math reasoning benchmarks. CoKV provides a scalable and practical solution for enhancing the efficiency of LLMs in real-world applications. In future work, we plan to extend CoKV to jointly optimize KV cache budget allocation and model safety by integrating safety-aware importance metrics into the sliced Shapley value framework. This extension will explicitly account for critical safety considerations, including robustness against adversarial prompts, and effective mitigation of hallucinations.

ACKNOWLEDGMENTS

This work was supported in part by the National Key RD Program of China (2022YFB3103401), NSFC (62472378, U23A20306, U25A20430, 62502416), the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars (LR25F020001), the Zhejiang Province Pioneer Plan (2024C01074, 2025C01084), the Research Grants Council of Hong Kong SAR, China (Grant No. 15207725), and the Innovation and Technology Fund of Hong Kong SAR, China (Grant No. ITS-140-23FP).

REFERENCES

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*. https://openreview.net/forum?id=hmOwOZWzYE

[2] Aaron Archer, Kevin Aydin, MohammadHossein Bateni, Vahab S. Mirrokni, Aaron Schild, Ray Yang, and Richard Zhuang. 2019. Cache-aware load balancing of data center applications. *Proc. VLDB Endow.* 12, 6 (2019), 709–723. https://doi.org/10.14778/3311880.3311887

[3] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 3119–3137. https://doi.org/10.18653/v1/2024.acl-long.172

[4] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069 [cs.CL] https://arxiv.org/abs/2406.02069

[5] Jin Chen, Zheng Liu, Xu Huang, Chenwang Wu, Qi Liu, Gangwei Jiang, Yuanhao Pu, Yuxuan Lei, Xiaolong Chen, Xingmei Wang, Kai Zheng, Defu Lian, and Enhong Chen. 2024. When large language models meet personalization: perspectives of challenges and opportunities. *World Wide Web (WWW)* 27, 4 (2024), 42. https://doi.org/10.1007/S11280-024-01276-1

[6] Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024. ArkVale: Efficient Generative LLM Inference with Recallable Key-Value Eviction. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/cd4b49379efac6e84186a3ffce108c37-Abstract-Conference.html

[7] Yaofo Chen, Zeng You, Shuhai Zhang, Haokun Li, Yirui Li, Yaowei Wang, and Mingkui Tan. 2024. Core context aware transformers for long context language modeling. *arXiv preprint arXiv:2412.12465* (2024).

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

[9] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=H4DqfPSibmx

[10] Xiaotie Deng and Christos H. Papadimitriou. 1994. On the Complexity of Cooperative Solution Concepts. *Math. Oper. Res.* 19, 2 (1994), 257–266. https://doi.org/10.1287/MOOR.19.2.257

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). https://doi.org/10.48550/ARXIV.2407.21783 arXiv:2407.21783

[12] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. arXiv:2407.11550 [cs.CL] https://arxiv.org/abs/2407.11550

[13] Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. 2025. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=FJFVmeXusW

[14] Shihong Gao, Xin Zhang, Yanyan Shen, and Lei Chen. 2025. Apt-Serve: Adaptive Request Scheduling on Hybrid Cache for Scalable LLM Inference Serving. *Proc. ACM Manag. Data* 3, 3 (2025), 130:1–130:28. https://doi.org/10.1145/3725394

[15] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=uNrFpDPMyo

[16] Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*. PMLR, 2242–2251.

[17] Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, et al. 2024. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature medicine* 30, 9 (2024), 2613–2622.

[18] Alireza Heidari, Amirhossein Ahmadi, Zefeng Zhi, and Wei Zhang. 2024. MetaHive: A Cache-Optimized Metadata Management for Heterogeneous Key-Value Stores. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. https://vldb.org/workshops/2024/proceedings/CloudDB/clouddb-3.pdf

[19] William Held and Diyi Yang. 2023. Shapley head pruning: Identifying and removing interference in multilingual transformers. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. 2416–2427.

[20] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874* (2021).

[21] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems* 37 (2024), 1270–1303.

[22] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654* (2024).

[23] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J. Spanos. 2019. Towards Efficient Data Valuation Based on the Shapley Value. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Masashi Sugiyama (Eds.), Vol. 89. PMLR, 1167–1176. https://proceedings.mlr.press/v89/jia19a.html

[24] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[25] Greg Kamradt. 2023. Needle In A Haystack - Pressure Testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

[26] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024. Pre-Trained Language Models for Text Generation: A Survey. *ACM Comput. Surv.* 56, 9 (2024), 230:1–230:39. https://doi.org/10.1145/3649449

[27] Yuhang Li, Rong Gu, Chengying Huan, Zhibin Wang, Renjie Yao, Chen Tian, and Guihai Chen. 2025. HotPrefix: Hotness-Aware KV Cache Scheduling for Efficient Prefix Sharing in LLM Inference Systems. *Proc. ACM Manag. Data* 3, 4 (2025), 250:1–250:27. https://doi.org/10.1145/3749168

[28] 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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=poE54GOq2l

[29] Zi Liang, Haibo Hu, Qingqing Ye, Yaxin Xiao, and Haoyang Li. 2024. Why Are My Prompts Leaked? Unraveling Prompt Extraction Threats in Customized Large Language Models. *CoRR* abs/2408.02416 (2024). https://doi.org/10.48550/ARXIV.2408.02416 arXiv:2408.02416

[30] Zi Liang, Qingqing Ye, Yanyun Wang, Sen Zhang, Yaxin Xiao, Ronghua Li, Jianliang Xu, and Haibo Hu. 2025. "Yes, My LoRD." Guiding Language Model Extraction with Locality Reinforced Distillation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1441–1465.

[31] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=JZfg6wGi6g

[32] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=L057s2Rq8O

[33] Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. 2022. Sampling Permutations for Shapley Value Estimation. *J. Mach. Learn. Res.* 23 (2022), 43:1–43:46. https://jmlr.org/papers/v23/21-0439.html

[34] Christopher Musco and R Teal Witter. 2024. Provably accurate shapley value estimation via leverage score sampling. *arXiv preprint arXiv:2410.01917* (2024).

[35] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. YaRN: Efficient Context Window Extension of Large Language Models. arXiv:2309.00071 [cs.CL] https://arxiv.org/abs/2309.00071

[36] Xiaoye Qu, Zengqi Yu, Dongrui Liu, Wei Wei, Daizong Liu, Jianfeng Dong, and Yu Cheng. 2025. Cooperative or Competitive? Understanding the Interaction between Attention Heads From A Game Theory Perspective. In *Proceedings of*

the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 14079–14099.

[37] Robert J Serfling. 1974. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics* (1974), 39–48.

[38] Lloyd S Shapley. 1953. A value for n-person games. *Contribution to the Theory of Games* 2 (1953).

[39] Yi Su, Yuechi Zhou, Quantong Qiu, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Accurate KV Cache Quantization with Outlier Tokens Tracing. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, 12895–12915. https://aclanthology.org/2025.acl-long.631/

[40] Qiheng Sun, Jiayao Zhang, Jinfei Liu, Li Xiong, Jian Pei, and Kui Ren. 2024. Shapley Value Approximation Based on Complementary Contribution. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 9263–9281. https://doi.org/10.1109/TKDE.2024.3438213

[41] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, et al. 2020. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 1493–1509.

[42] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Danning Ke, Yiwu Yao, and Gongyi Wang. 2025. RazorAttention: Efficient KV Cache Compression Through Retrieval Heads. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=tkiZQlL04w

[43] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 18–32.

[44] Kefei Wang and Feng Chen. 2023. Catalyst: Optimizing Cache Management for Large In-memory Key-value Systems. *Proc. VLDB Endow.* 16, 13 (2023), 4339–4352. https://doi.org/10.14778/3625054.3625068

[45] Kefei Wang, Jian Liu, and Feng Chen. 2020. Put an Elephant into a Fridge: Optimizing Cache Efficiency for In-memory Key-value Stores. *Proc. VLDB Endow.* 13, 9 (2020), 1540–1554. https://doi.org/10.14778/3397230.3397247

[46] Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. 2025. Retrieval Head Mechanistically Explains Long-Context Factuality. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=EytBpUGB1Z

[47] Zhiyao Wu, Zi Liang, and Haibo Hu. 2026. Decoding Web Memorization: A Semantic Membership Inference Attack on LLMs. In *Proceedings of the ACM Web Conference 2026*. 3624–3632.

[48] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=cFu7ze7xUm

[49] 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*. https://openreview.net/forum?id=NG7sS51zVF

[50] Xiaoyu Xu, Xiang Yue, Yang Liu, Qingqing Ye, Huadi Zheng, Peizhao Hu, Minxin Du, and Haibo Hu. 2025. Unlearning Isn't Deletion: Investigating Reversibility of Machine Unlearning in LLMs. *arXiv preprint arXiv:2505.16831* (2025).

[51] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[52] June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024. No Token Left Behind: Reliable KV Cache Compression via Importance-Aware Mixed Precision Quantization. *CoRR* abs/2402.18096 (2024). https://doi.org/10.48550/ARXIV.2402.18096 arXiv:2402.18096

[53] Jiayao Zhang, Qiheng Sun, Jinfei Liu, Li Xiong, Jian Pei, and Kui Ren. 2023. Efficient Sampling Approaches to Shapley Value Approximation. *Proc. ACM Manag. Data* 1, 1, Article 48 (May 2023), 24 pages. https://doi.org/10.1145/3588728

[54] Qizhen Zhang, Philip A. Bernstein, Daniel S. Berger, and Badrish Chandramouli. 2021. Redy: Remote Dynamic Memory Cache. *Proc. VLDB Endow.* 15, 4 (2021), 766–779. https://doi.org/10.14778/3503585.3503587

[55] Shuhai Zhang, Zeng You, Yaofo Chen, Zhiquan Wen, Qianyue Wang, Zhijie Qiu, Yuanqing Li, and Mingkui Tan. 2025. Curse of High Dimensionality Issue in Transformer for Long Context Modeling. In *Forty-second International Conference on Machine Learning*.

[56] Xinwei Zhang, Hangcheng Liu, Li Bai, Hao Wang, Qingqing Ye, Tianwei Zhang, and Haibo Hu. 2026. On the Adversarial Robustness of Large Vision-Language Models under Visual Token Compression. *arXiv preprint arXiv:2601.21531* (2026).

[57] 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*. https://openreview.net/forum?id=RkRrPp7GKO

[58] Xiangyu Zhi, Xiao Yan, Bo Tang, Ziyao Yin, Yanchao Zhu, and Minqi Zhou. 2023. CoroGraph: Bridging Cache Efficiency and Work Efficiency for Graph Algorithm Execution. *Proc. VLDB Endow.* 17, 4 (2023), 891–903. https://doi.org/10.14778/3636218.3636240