FairKV: Balancing Per-Head KV Cache for Fast Multi-GPU Inference

Anonymous ACL submission

Abstract

KV cache techniques in Transformer models aim to reduce redundant computations at the expense of substantially increased memory usage, making KV cache compression an important and popular research topic. Recently, state-of-the-art KV cache compression methods implement imbalanced, per-head allocation algorithms that dynamically adjust the KV cache budget for each attention head, achieving excellent performance in single-GPU scenarios. However, we observe that such imbalanced compression leads to significant load imbalance when deploying multi-GPU inference, as some GPUs become overburdened while others remain underutilized. In this paper, we propose FairKV, a method designed to ensure fair memory usage among attention heads in systems employing imbalanced KV cache compression. The core technique of FairKV is *Fair-Copying*, which replicates a small subset of memory-intensive attention heads across GPUs using data parallelism to mitigate load imbalance. Theoretical analysis provides insights, while experiments on popular models, including LLaMA 70b and Mistral 24b model, demonstrate that FairKV increases throughput by 1.66x compared to standard tensor parallelism inference. Our code will be released as open source upon acceptance.

1 Introduction

1.1 Background and Motivation

Large-scale Transformer-based models are at the core of modern artificial intelligence. To support fast inference, these models rely on a key-value (KV) cache(Vaswani et al., 2017; Dai et al., 2019; Rae et al., 2019) that stores key and value embeddings from previously generated tokens, trading memory usage for reduced redundant computation. This cache prevents redundant computations and is crucial for efficient sequence generation. However, the huge memory usage of the KV cache becomes a bottleneck for inference systems, and a great number works have investigated KV cache compression(Ge et al., 2024; Liu et al., 2023; Xiao et al., 2024; Li et al., 2024; Cai et al., 2024; Feng et al., 2024; Fu et al., 2024; Adnan et al., 2024).

Traditional KV cache compression methods (StreamingLLM(Xiao et al., 2024), SnapKV(Li et al., 2024) and PyramidKV(Cai et al., 2024)) naturally allocate an equal (fair) KV cache budget to all attention heads. This fair allocation simplifies the design and ensures uniform memory usage across heads. Recently, SOTA works of KV cache compression methods (AdaSnapKV(Feng et al., 2024) and HeadKV(Fu et al., 2024)) introduce *imbalanced per-head KV cache compression algorithms* that dynamically adjust the KV cache budget for each attention head. These methods compress the KV cache more effectively and achieve SOTA performance in single-GPU scenarios.

Unfair head load Problem: Our investigation reveals that although imbalanced per-head KV cache compression techniques reduce overall memory usage, they result in an *uneven per-head KV cache load distribution* during multi-GPU inference, which we refer to as the **unfair head load** problem. In systems using tensor parallelism, one of the most common parallel strategies, this non-uniform KV cache usage forces some GPUs to handle a disproportionate share of memory-intensive attention heads. This imbalance increases GPU idle time, elevates inference latency, and ultimately degrades system throughput. Despite extensive research on load balancing in parallel inference systems, no prior work has identified or addressed this specific imbalance caused by imbalanced KV cache compression. Addressing this imbalance is crucial for achieving efficient, low-latency inference in large-scale Transformer models, particularly in multi-GPU settings.

1.2 Our solution

To address the unfair head load problem, we propose FairKV. Our approach targets the fair memory usage among attention heads in imbalanced KV cache compression systems. FairKV mainly employs two techniques to address this issue: best-effort assignment and fair-copying. *Best-effort Assignment* is responsible for distributing attention heads across GPUs to achieve a relatively balanced workload. *Fair-Copying* involves replicating certain attention heads to participate in the assignment, thereby reducing the workload of the replicated heads.

Technique I: Best-effort Assignment. Best-effort Assignment works as follows. We first analyze the KV cache consumption of each attention head under imbalanced compression. Based on this analysis, our allocation algorithm assigns attention heads to GPUs such that the aggregate memory and computational load is balanced across GPUs in a best-effort manner. This method requires no additional overhead and can be integrated into existing multi-GPU inference systems easily, offering a practical improvement in load balancing without modifying the underlying KV cache compression scheme.

Technique II: Fair-Copying. While best-effort assignment offers a straightforward solution, some attention heads remain significantly more memory-intensive, leading to persistent bottlenecks. To further improve load balance, we propose another technique called *Fair-Copying*, which utilizes Data Parallel techniques to enhance the effectiveness of the assignment. In this technique, we set a replication budget that allows the algorithm to attempt replicating attention heads. By utilizing data parallelism to reduce the load on the replicated heads, these replicated heads participate in the assignment alongside the original ones, thereby expanding the search space and enabling finer-grained partitioning. This replication method minimizes additional overhead while substantially reducing GPU idle time and inference latency.

Key Contributions. This paper makes the following key contributions:

1) To the best of our knowledge, we are the first to reveal the unfair head load problem: imbalanced per-head KV cache compression causes significant load imbalance in multi-GPU inference environments.
2) To address the unfair head load problem, we propose FairKV, which includes two mechanisms—best-effort assignment and fair-copying.
3) Theoretical analysis provides insights, while experiments on popular models, including LLaMA 70b and Mistral 24b model, demonstrate that FairKV increases throughput by 1.66x compared to standard tensor parallelism inference. Our code will be released as open source upon acceptance.

2 Related Work

Inference efficiency for large language models is critically limited by both memory bandwidth and computational power. KV cache compression (Ge et al., 2024; Zhang et al., 2024; Yang et al., 2024), a widely adopted optimization technique, reduces memory for storing previous key-value states in attention layers. It can generally be classified into two categories: *Balanced (Fair) Per-Head Compression* and *Imbalanced (Unfair) Per-Head Compression*.

Balanced (Fair) Per-Head Compression methods applies the same strategy to all attention heads. For instance, StreamingLLM (Xiao et al., 2024) retains the initial k sink tokens along with the recent window. H2O (Zhang et al., 2024) further prioritizes important cache entries based on accumulated attention scores, while SnapKV (Li et al., 2024) selects entries using attention scores from the observation window. Recently, Pyramid (Yang et al., 2024; Cai et al., 2024) recognize distinct attention distribution patterns across different layers. However, these methods disregard the varying importance of different heads in actual computations.

In contrast, *Imbalanced (Unfair) Per-Head Compression* algorithms, like Ada-SnapKV (Feng et al., 2024) and HeadKV(Fu et al., 2024), dynamically adjust the KV cache budget per attention head based on the current layer's computational and memory requirements. Ada-SnapKV determines the budget for each head during inference, with a fully dynamic allocation. HeadKV, however, pre-allocates a fixed base budget for each head according to its importance and then adds a dynamic budget. This tailored cache allocation offers more flexibility and optimization potential. Table 3 in the appendix show that Ada-SnapKV outperforms other methods on multiple tasks in the LongBench v1 (Bai et al., 2024), demonstrating the effectiveness of *Imbalanced (Unfair) Per-Head Compression* approach.

Table 1: Cosine Similarity of Retained KV Cache in Per-Head KV Compression across LongBench Sub-datasets

KV Budget	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Coding		Avg	Max	Min	Std
	NtrQA	Qasper	MF-en	HpQA	2WMQA	Musiq	GovRp	QMSum	MNews	TREC	TriQA	SAMSum	LCC	RB-P				
Llama-3.3-70B-Instruct																		
128	0.974	0.977	0.977	0.979	0.980	0.979	0.971	0.979	0.974	0.974	0.978	0.969	0.975	0.977	0.976	0.980	0.969	0.003
256	0.961	0.965	0.967	0.970	0.971	0.969	0.958	0.966	0.962	0.962	0.968	0.949	0.964	0.968	0.964	0.971	0.949	0.006
512	0.955	0.962	0.962	0.965	0.968	0.964	0.954	0.960	0.955	0.957	0.964	0.944	0.959	0.965	0.959	0.968	0.944	0.006
1024	0.950	0.961	0.959	0.961	0.966	0.959	0.953	0.956	0.943	0.956	0.962	0.947	0.949	0.962	0.956	0.966	0.943	0.006
Meta-Llama-3-8B																		
128	0.904	0.935	0.932	0.919	0.933	0.914	0.929	0.919	0.938	0.919	0.928	0.934	0.933	0.932	0.926	0.938	0.904	0.009
256	0.873	0.912	0.913	0.897	0.917	0.887	0.912	0.891	0.923	0.896	0.909	0.911	0.915	0.909	0.905	0.923	0.873	0.013
512	0.886	0.916	0.917	0.909	0.924	0.902	0.917	0.904	0.933	0.909	0.917	0.908	0.925	0.915	0.913	0.933	0.886	0.011
1024	0.912	0.932	0.934	0.930	0.937	0.930	0.931	0.927	0.949	0.929	0.935	0.922	0.944	0.932	0.932	0.949	0.912	0.009
Mistral-Small-24B-Instruct-2501																		
128	0.970	0.973	0.974	0.974	0.975	0.973	0.971	0.975	0.970	0.959	0.975	0.971	0.971	0.973	0.972	0.975	0.959	0.004
256	0.960	0.963	0.965	0.965	0.967	0.965	0.959	0.965	0.960	0.944	0.967	0.951	0.960	0.965	0.961	0.967	0.944	0.006
512	0.954	0.958	0.960	0.958	0.964	0.958	0.951	0.959	0.959	0.945	0.963	0.936	0.955	0.962	0.956	0.964	0.936	0.007
1024	0.954	0.961	0.961	0.959	0.965	0.957	0.951	0.959	0.963	0.954	0.962	0.937	0.959	0.962	0.957	0.965	0.937	0.007

However, a significant challenge arises when applying *Imbalanced (Unfair) Per-Head Compression* in tensor parallelism, which has become the preferred method for inference in large language models (Lu et al., 2017; Shazeer et al., 2018; Shoeybi et al., 2020; Rajbhandari et al., 2020). The non-uniform distribution of KV cache across heads causes computational load imbalance, degrading overall inference efficiency. This highlights the urgent need for balancing per-head KV cache.

Table 2: GPU Utilization of Different Models. We applied Ada-SnapKV to three models, setting the KV cache budget to 128, 256, 512, and 1024, and measured its GPU utilization under tensor parallel sizes of 2, 4, and 8.

Model	KV Cache Budget	TP = 2	TP = 4	TP = 8
LLaMA-3.3-70B-Instruct	128	92.5	81.6	64.7
	256	87.5	74.1	57.2
	512	86.4	70.5	55.7
	1024	87.2	71.2	58.2
Meta-LLaMA-3-8B	128	92.1	84.4	70.8
	256	91.8	82.0	68.5
	512	90.6	81.6	68.8
	1024	90.9	82.1	69.8
Mistral-Small-24B-Instruct	128	93.2	86.3	75.2
	256	91.7	82.9	71.9
	512	91.2	82.0	70.8
	1024	91.2	82.1	70.8

3 Preliminary

3.1 Observation of KV Cache Selection

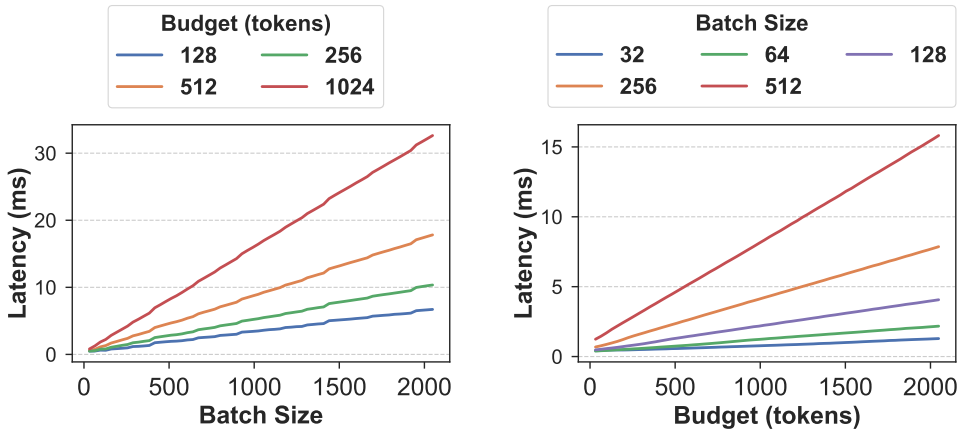
We examined per-head KV cache eviction patterns across multiple datasets using different models (LLaMA-3.3-70B-Instruct, Meta-LLaMA-3-8B, and Mistral-Small-24B-Instruct) and several subsets from LongBench v1 for our experiments and set the KV budget for the attention heads to 128, 256, 512, and 1024 as our basic experimental setup.

The per-head KV cache compression algorithm results in varying KV cache budgets across different attention heads, leading to imbalanced workloads when combined with tensor parallelism. The results, summarized in Table 2, show a clear trend:

Decreasing GPU Utilization with Larger TP Sizes: For instance, in LLaMA-3.3-70B-Instruct with KV cache budget 128, GPU utilization drops from 92.5% (TP=2) to 81.6% (TP=4) and further to 64.7% (TP=8). A similar trend is observed in Meta-LLaMA-3-8B, where GPU utilization declines from 92.1% (TP=2) to 84.4% (TP=4) and 70.8% (TP=8) under the same KV cache budget.

Impact of KV cache Budget: Lower KV cache budgets generally yield better GPU utilization. For example, in Mistral-Small-24B-Instruct with TP=4, GPU utilization is 86.3% at KV=128 but decreases to 82.9% at KV=256 and further to 82.1% at KV=1024. Similarly, in LLaMA-3.3-70B-Instruct with TP=4, GPU utilization drops from 81.6% (KV=128) to 71.2% (KV=1024). This suggests that larger KV cache budgets lead to greater workload imbalance among attention heads, which negatively impacts GPU efficiency.

Additionally, as shown in Table 1, we use cosine similarity to quantify the difference in KV cache allocation between a subset and the remaining datasets. Finally, we averaged the metrics across all subsets to obtain an overall indicator. Our analysis indicates that eviction patterns remain largely consistent across datasets, confirming that the statistics for retained KV cache can guide optimization. Furthermore, the allocation pattern of the KV cache budget is influenced by the specific model used. Given this dataset-invariant nature, optimization strategies can be designed based on these profiles without requiring per-dataset recali-



(a) Latency vs. Batch Size under Different Budgets (b) Latency vs. Budget under Different Batch Sizes

Figure 1: Impact of Batchsize and KV Cache Budget on Inference Latency

bration.

3.2 Empirical Performance Analysis

To optimize the distribution of GPU workload, we built an empirical model mapping batch size, retained KV cache count, and inference latency. Measurements were taken across various configurations on a multi-GPU setup.

We conducted experiments on the LongBench v1 dataset by systematically varying both batch size and KV cache budget parameters. Specifically, we controlled these variables independently to evaluate their individual and combined effects. We simulated the inference scenario of LLaMA 70b on a single layer and obtained the latency during decoding one token. The experimental results, as illustrated in Figure 1, demonstrate the performance characteristics under different configurations.

For the batch size latency model (Figure a), we observe that latency (L) increases approximately linearly with batch size (B) across different budget configurations. The relationship can be expressed as $L \approx \alpha B + \beta$, where α represents the slope and β the initial offset. The graph shows four distinct budget levels (128, 256, 512, and 1024), with higher budget values corresponding to steeper slopes in the linear relationship. Similarly, the KV cache latency model (Figure b) demonstrates a comparable linear pattern, where latency (L) increases proportionally with the KV cache budget (C). This can be represented as $L \approx \gamma C + \delta$, where γ and δ are the slope and offset parameters respectively. The data presents five different batch sizes (32, 64, 128, 256, and 512), with larger batch sizes showing more pronounced slopes in their linear relationships. In both cases, the approximately linear nature of these relationships is particularly noteworthy, as it suggests predictable scaling behavior in the system's performance characteristics.

4 Design of FairKV

4.1 Overview

During inference with tensor parallelism, per-head KV cache compression algorithms can lead to imbalanced GPU workloads, which in turn reduces inference efficiency.To address this issue, we designed FairKV, a load-aware static approach that uses a search algorithm to reassemble and rearrange attention heads across layers, thereby balancing the load among GPUs. Moreover, FairKV leverages fair-copying mechanism to expand the search space via replication of attention heads and DataParallel techniques, enabling a more fine-grained balancing of GPU loads and enhancing GPU utilization during inference. Figure 2 visually illustrates the core concept of FairKV.

The FairKV algorithm requires predefining the model and the KV cache budget to be used. It then samples a dataset to analyze the proportion of KV cache budget allocated to attention heads across different layers for that model, summarizing the findings into a statistical profile. Based on this data, we first replicate attention heads to expand the search space. Next, we perform a constrained search to determine the optimal attention head arrangement. Finally, the model weights are loaded according to this arrangement for inference.

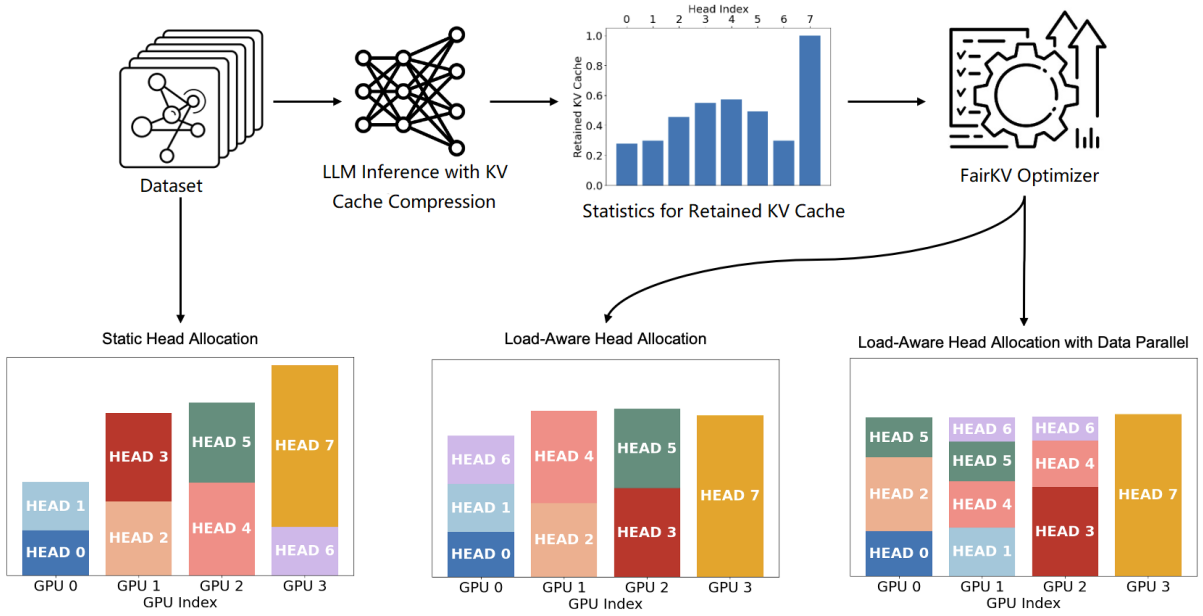


Figure 2: Illustration of different head allocation strategies for multi-GPU inference in large-scale transformer models. The figure shows the following strategies: (1) Static Head Allocation (SHA), where attention heads are evenly distributed across GPUs without considering computational load; (2) Load-Aware Head Allocation (FairKV-NoDP), where attention heads are allocated based on their computational load, ensuring a balanced GPU utilization; (3) Load-Aware Head Allocation with DataParallel (FairKV-DP), where heads are replicated across GPUs for improved load distribution and efficiency;

4.2 Search Space

Fair-copying leverages Data Parallel techniques to expand the search space by replicating some redundant heads, with the goal of achieving more effective search outcomes. Experiments were conducted with selective head replication, and the results demonstrated improved latency distribution across GPUs. Allowing for limited redundancy provides an effective way to enhance parallel efficiency without excessive computational overhead.

4.3 Mathematical Model for Hybrid Parallelism in FairKV

This section presents a comprehensive mathematical formulation of the FairKV system for optimizing multi-GPU inference in large language models with KV cache compression.

4.3.1 System Parameters

The system parameters are defined to model the components involved in the FairKV strategy, for each transformer layer $l \in L$:

- $H_l = \{h_1, ..., h_n\}$: attention heads
- $G = \{g_1, ..., g_m\}$: available GPUs
- w_i: workload for head h_i
- r_{ij}: replication factor of head h_i on GPU g_j

4.3.2 Decision Variables

The decision variables are crucial for determining the optimal assignment of attention heads to GPUs:

$$x_{ij} = \begin{cases} r_{ij} & \text{if head } h_i \text{ is assigned to GPU } g_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

4.3.3 Parallelism Constraints

The hybrid parallelism approach in FairKV combines Tensor Parallelism and Data Parallelism to address the challenges posed by head-wise KV Cache compression. Tensor Parallelism ensures that the computational workload is distributed across multiple GPUs by partitioning the attention heads, while Data Parallelism introduces redundancy by allowing selective replication of attention heads across GPUs. This hybrid approach aims to balance the computational load, ensuring that no single GPU becomes a bottleneck due to uneven KV Cache compression rates.

Head Distribution Each head must have at least one GPU assignment to ensure that all computa-

tions are accounted for.

$$\sum_{j \in G} \mathbb{1}(x_{ij} > 0) \geq 1 \quad \forall i \in H_l, \forall l \in L \tag{2}$$

Data Parallelism Total replication factor for each head:

$$\sum_{j \in G} r_{ij} \leq R_{max} \quad \forall i \in H_l, \forall l \in L \tag{3}$$

The total replication factor for each head across all GPUs must not exceed a predefined maximum, R_{max}. This prevents over-replication, which could lead to increased communication overhead and reduced efficiency.

4.3.4 Optimization Objective

The primary goal of the FairKV strategy is to minimize the maximum processing time across all GPUs. This is formulated as:

$$\min \max_{j \in G} \sum_{l \in L} \sum_{i \in H_l} \frac{x_{ij} w_i}{r_{ij}} \tag{4}$$

4.3.5 System Efficiency

System efficiency is calculated to evaluate how well the GPUs are utilized. The formula for efficiency is:

$$E = \frac{1}{|G|} \sum_{j \in G} \frac{\sum_{l \in L} \sum_{i \in H_l} \frac{x_{ij} w_i}{r_{ij}}}{\max_{k \in G} \sum_{l \in L} \sum_{i \in H_l} \frac{x_{ik} w_i}{r_{ik}}} \tag{5}$$

4.4 Optimizer Design

We introduce a recursive backtracking algorithm that systematically explores possible head distributions while ensuring workload balance.

This approach systematically searches for the best head distribution by iterating over all valid partitions. By ensuring balanced allocation, the algorithm minimizes inference latency across GPUs, leading to improved system efficiency.

4.5 Key Innovations

Workload-aware Redistribution A static yet optimized head allocation strategy balances GPU workloads efficiently.

Hybrid Parallelism Selective head replication combines tensor and data parallelism.

Latency-driven Optimization Backtracking minimizes inference latency, improving GPU utilization.

Algorithm 1 Backtracking-Based Partitioning Algorithm

Input: List of heads L, max GPU load m
Output: Optimal partitioning of L
Initialize result set R
procedure BACKTRACK($index, current$)
  if $index == |L|$ and $|current| \leq m$ **then**
    Add $current$ to R **return**
  end if
  Add $L[index]$ to $current$ and recurse BACKTRACK($index + 1, current$)
  Remove last element from $current$
  for $n = 2$ to $m - |current| + 1$ **do**
    Split $L[index]$ into n parts and add to $current$ BACKTRACK($index + 1, current$)
    Remove last n elements from $current$
  end for
end procedure

5 Evaluation

In this section, we primarily conduct a detailed evaluation of the FairKV method proposed in the previous sections, assessing whether FairKV can effectively improve inference efficiency in hybrid parallelism approaches during Per-Head KV cache Compression.

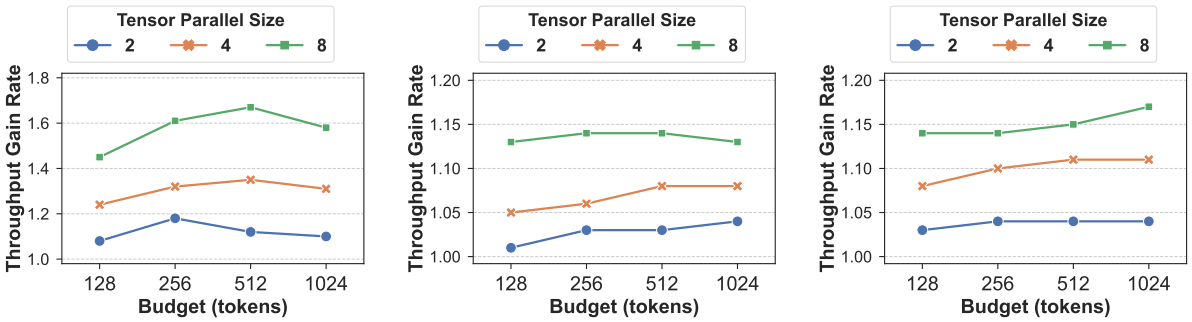
5.1 Experimental Setup

We used Python 3.10.16 as the programming language and implemented FairKV on PyTorch. We leveraged AdaKV from KVPress as the per-head KV cache compression algorithm and tested our approach using Llama models (Touvron et al., 2023) of different sizes as well as the Mistral model (Jiang et al., 2023).

Model and Hardware Configurations We used the LLaMA-3.3-70B-Instruct model, the Meta-LLaMA-3-8B model, and the Mistral-Small-24B-Instruct-2501 model as our experimental models. Our hardware environment consists of four Nvidia A100-80G GPUs and 960GB of CPU memory.

Dataset Configurations We used LongBench v1 (Bai et al., 2024) as the evaluation dataset. LongBench v1 is a bilingual, multi-task benchmark for long-text understanding, enabling a more rigorous evaluation of long-text comprehension.

Baseline To the best of our knowledge, this study is the first to highlight the issue of GPU workload imbalance caused by per-head KV cache compression algorithms in tensor parallelism. Therefore,



(a) Rate on LLaMA-3.3-70B-Instruct (b) Rate on Meta-LLaMA-3-8B (c) Rate on Mistral-Small-24B-Instruct

Figure 3: Throughput Gain Rates of FairKV on different models, where the throughput of the baseline model is regarded as 1.0.

we chose the situation of conducting tensor parallel inference with Per-Head KV Compression but without FairKV as the baseline. By comparing the cases with and without FairKV, we can determine whether FairKV can improve GPU utilization and reduce inference latency.

Evaluation Metrics The purpose of the evaluation experiments is to verify the following two questions: First, whether FairKV can effectively reduce inference latency, and second, whether FairKV can effectively increase GPU utilization. Therefore, our evaluation metrics are the following two aspects: inference time and GPU utilization.

5.2 Performance Evaluation

Performance evaluation is a critical component in assessing the effectiveness of the FairKV method in improving multi-GPU inference efficiency for large language models. This subsection outlines the key metrics and methodologies used to evaluate the performance of FairKV, ensuring a comprehensive understanding of its impact on system efficiency.

5.2.1 Throughput Improvement

We evaluated the performance of FairKV across different models. Specifically, we selected a diverse set of models, including LLaMA-3.3-70B-Instruct, Meta-LLaMA-3-8B, and Mistral-Small-24B-Instruct, and set the tensor parallel size to either 4 or 8 with RC fixed at 4. Throughput gains relative to SHA were measured under KV cache budgets of 128, 256, 512, and 1024. As shown in Figure 3, FairKV accelerates various models, and due to the inherent characteristics of each model, its throughput under SHA conditions differs. As a result, the acceleration effect of FairKV may vary. Among them, FairKV achieves the highest benefit of up to 1.66 on the Llama-3.3-70B-Instruct model.

We also evaluated the performance of FairKV under different tensor parallel sizes. Firstly, we observed that FairKV provides acceleration benefits under all tensor parallel size conditions. By comparing subfigures a, b, and c, we can see that as the tensor parallel size increases, the acceleration effect of FairKV significantly improves under the same model and KV cache budget conditions. This suggests that FairKV is more likely to achieve better acceleration performance when the tensor parallel size is larger.

As the KV cache budget increases, the acceleration effect of FairKV shows a slight improvement for Meta-LLaMA-3-8B and Mistral-Small-24B-Instruct, while for LLaMA-3.3-70B-Instruct, the acceleration effect initially improves significantly and then slightly decreases. Overall, across these three models, the acceleration effect of FairKV tends to improve as the KV cache budget increases.

In general, FairKV demonstrates acceleration benefits across different models, tensor parallel sizes, and KV cache budgets. At the same time, the effectiveness of FairKV is influenced by these factors, and the results may vary accordingly.

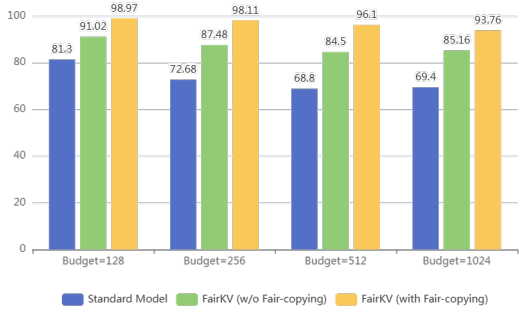


Figure 4: Ablation Test Among Standard Model, FairKV w/o Fair-copying and FairKV with Fair-copying

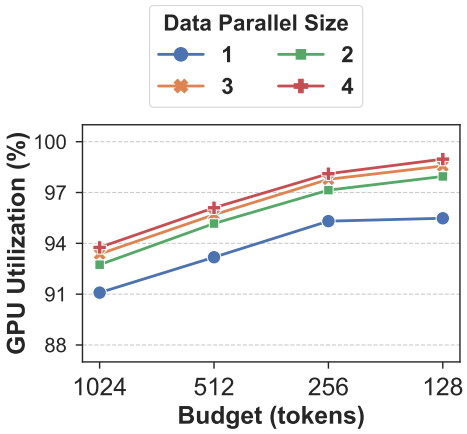


Figure 5: GPU Utilization with different Data Parallel Size on LLaMA-3.3-70B.

5.2.2 GPU Utilization Improvement

We used LLaMA-3.3-70B-Instruct to evaluate the impact of FairKV on GPU utilization. In Figure 4, we conducted ablation tests among standard model, FairKV without Fair-copying and FairKV with Fair-copying. The result indicate that both FairKV with or without Fair-copying significantly improves GPU utilization compared to standard model, demonstrating that the FairKV method can effectively balance GPU loads. Moreover, FairKV with Fair-copying shows further improvement over FairKV without Fair-copying. Then, to measure the impact of the parameter on the FairKV with Fair-copying group, we set the size of parallel size, which is also the count of copied heads (CH), to 1, 2, 3, and 4. We measured the GPU utilization for these groups under KV cache budgets of 128, 256, 512, and 1024, with the results shown in Figure 5. The results suggesting that incorporating only a small number of copied heads via Data Parallel can enhance the performance of the FairKV strategy greatly. We also observed a positive correlation between the performance of FairKV and CH; as CH increases, the GPU utilization curve becomes less steep, indicating that while increases in CH yield significant benefits when CH is small, the incremental gains diminish when CH is larger.

6 Conclusion

In this paper, we propose *FairKV*, a novel optimization technique designed to improve inference efficiency by dynamically adjusting the allocation of attention heads in Tensor Parallelism. FairKV leverages the statistics of the retained Key-Value (KV) cache to partition attention heads based on their computational load, ensuring a balanced workload distribution across GPUs.

Our method has been extensively evaluated across various configurations, demonstrating consistent performance improvements. Specifically, FairKV significantly reduces inference latency and improves GPU utilization, regardless of the number of GPUs used, the model size (including different versions of LLaMa and Mistral), or the KV cache budget. This versatility highlights the robustness of our approach, making it effective across different settings and resource constraints. Experimental results show that FairKV achieves up to a 66% increase in inference throughput, with minimal impact on model accuracy. These findings confirm that FairKV is a promising solution for optimizing large-scale model inference in real-world, real-time applications.

Future work will explore the adaptability of FairKV to dynamic KV cache budgets and investigate its potential for scaling to models such as Mixture of Experts (MoE), where the computational load can vary significantly across different tasks. These enhancements aim to further extend the applicability of FairKV to a broader range of models and dynamic environments.

7 Limitations and Future works

Although *FairKV* provides significant improvements in inference efficiency, there are a few limitations that need to be addressed. First, FairKV is designed for a single machine with multiple GPUs and does not account for scenarios involving distributed systems across multiple machines. Second, the current method primarily focuses on the parallelization of the inference process without considering the separation of the prefill and decode stages, which is critical in certain real-time applications where the inference process is more complex and involves sequential decoding. Additionally, FairKV relies on the accuracy of the KV cache statistics for optimal head allocation. This means that the effectiveness of KV cache compression depends on the statistical properties of the cache, and any deviations from these properties could reduce the method's performance. We will address these limitations in future work.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *ArXiv*, abs/2403.09054.

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. *Preprint*, arXiv:2308.14508.

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. *Preprint*, arXiv:2406.02069.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics*.

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. *Preprint*, arXiv:2407.11550.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *Preprint*, arXiv:2410.19258.

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. *Preprint*, arXiv:2310.01801.

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. *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. *Preprint*, arXiv:2404.14469.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2023. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*.

Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 553–564. IEEE.

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

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-lm: Training multi-billion parameter language models using model parallelism. *Preprint*, arXiv:1909.08053.

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 M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *Preprint*, arXiv:2406.02069.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Preprint*, arXiv:2402.06262.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Coding		Ave. Score
	NtrQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	LCC	RB-P	
KV Budget = 128																	
StreamingLLM	13.57	11.48	24.47	**34.25**	**25.35**	**11.31**	**20.06**	**17.39**	17.80	**30.50**	50.14	35.33	**3.00**	5.50	15.50	60.25	23.49
Pyramid	**13.91**	**13.05**	**18.71**	27.43	14.36	6.35	17.93	17.25	19.52	21.00	89.09	**38.03**	1.11	5.00	61.93	57.00	26.35
SnapKV	12.23	11.88	17.93	25.32	12.41	8.20	17.88	16.84	19.44	22.00	90.97	37.83	2.50	**5.50**	61.62	**57.54**	26.26
Ada-SnapKV	13.52	12.79	18.30	28.94	14.13	9.35	19.04	17.26	**19.94**	25.00	**91.44**	37.96	1.54	4.00	**63.34**	56.39	**27.06**
KV Budget = 256																	
StreamingLLM	14.25	12.92	**28.75**	31.80	**19.19**	**11.39**	**22.86**	17.26	22.73	**44.00**	52.80	38.77	4.00	9.00	16.11	59.79	25.35
Pyramid	13.57	**17.36**	19.29	30.95	18.10	8.21	20.30	17.65	20.90	26.00	90.94	40.02	4.73	4.50	64.71	55.06	28.27
SnapKV	14.82	16.32	18.44	29.53	14.25	9.26	20.32	17.56	21.70	28.50	91.49	40.15	4.12	5.00	64.60	**54.78**	28.18
Ada-SnapKV	**15.19**	15.97	20.05	**35.08**	16.31	8.32	21.31	**17.99**	**22.11**	33.00	**91.68**	**41.28**	**5.93**	**5.50**	**66.14**	54.76	**29.41**
KV Budget = 512																	
StreamingLLM	14.47	16.86	**34.18**	31.88	18.76	**11.80**	**26.01**	18.20	**25.10**	**53.00**	60.04	41.23	4.00	9.50	19.74	**57.87**	27.66
Pyramid	15.67	21.59	23.56	35.63	**24.36**	9.77	22.18	19.05	23.25	34.00	91.11	42.44	4.61	**17.50**	65.87	54.41	31.56
SnapKV	15.22	22.90	23.41	33.00	22.54	8.91	22.63	18.31	23.50	35.00	91.39	41.85	4.50	15.50	**66.79**	53.75	31.20
Ada-SnapKV	**17.72**	**24.22**	25.38	**38.00**	23.47	9.25	23.55	**18.91**	23.82	43.00	**92.14**	**42.81**	**6.34**	15.00	66.12	55.32	**32.82**
KV Budget = 1024																	
StreamingLLM	13.36	21.55	**41.70**	32.80	22.63	12.88	28.40	19.36	**25.93**	**62.00**	75.54	41.76	2.00	11.00	22.91	**57.05**	30.68
Pyramid	16.67	29.11	28.74	40.13	27.89	13.43	24.63	20.36	25.01	43.00	91.86	43.48	**6.78**	52.50	65.34	54.87	36.49
SnapKV	19.57	31.67	31.52	**42.04**	27.89	12.94	25.31	19.66	25.12	48.50	91.37	42.83	5.99	**56.00**	**66.56**	53.97	37.56
Ada-SnapKV	**19.74**	**30.47**	31.42	40.82	**28.98**	**16.07**	**25.35**	**20.57**	25.57	53.50	**91.76**	**43.81**	4.63	53.50	65.75	55.90	**37.99**
KV Budget = 2048																	
StreamingLLM	16.86	31.83	**48.22**	33.59	29.36	15.74	**30.11**	20.89	26.68	**65.50**	**92.49**	44.12	5.25	21.00	39.75	**56.66**	36.13
Pyramid	21.16	36.66	37.56	43.28	36.25	**19.72**	27.90	21.73	26.35	53.00	92.36	**44.61**	6.94	**89.50**	64.23	53.11	42.15
SnapKV	21.24	39.86	38.21	45.96	35.36	21.20	28.68	21.49	26.50	58.00	92.36	44.06	6.31	88.50	**64.26**	53.73	42.86
Ada-SnapKV	**22.42**	**40.26**	40.13	**49.60**	**38.23**	19.70	28.49	**21.76**	**26.77**	61.50	92.35	44.04	**8.27**	85.00	64.11	54.09	**43.55**

Table 3: Comparison of Common KV Cache Compression Methods.

A Value of Per-Head KV Cache Compression Methods

Table 3 presents the performance of common KV cache compression methods on the LongBench v1 (Bai et al., 2024) dataset. We tested different KV cache compression methods on Llama-3.1-8B-Instruct with KV Cache budgets set to 128, 256, 512, 1024, and 2048. As described in Table 3, across various KV cache budget settings, Ada-SnapKV (the optimized version of SnapKV based on Ada-KV) achieved higher average scores than other methods on multiple tasks in the Longbench v1 dataset.

B Common KV Cache Compression Methods with Tensor Parallelism

Figure 6 shows the combination of common KV cache compression methods with tensor parallelism. As can be seen from the figure, the traditional Balanced (Fair) Per-Head Compression methods result in a balanced computational load after the partitioning in tensor parallelism. In these methods, each part of the parallel computation bears a relatively equal amount of work. On the contrary, the Imbalanced (Unfair) Per-Head Compression methods lead to an unbalanced computational load.

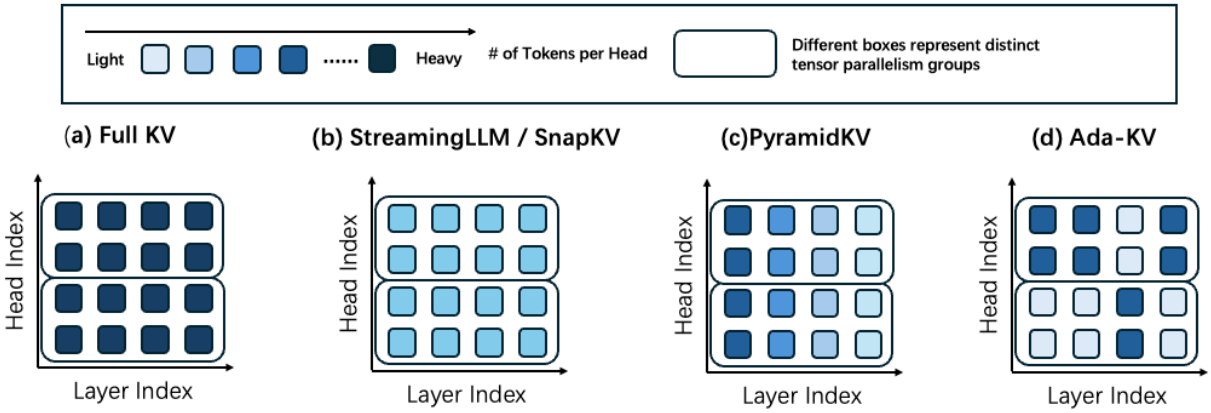


Figure 6: Common KV Cache Compression Methods with Tensor Parallelism