Rethinking KV Cache Pruning with Channel Interdependence for Efficient Long-Context Inference

Anonymous ACL submission

Abstract

Managing the extensive Key-Value (KV) cache is critical for efficient long-context processing in Large Language Models (LLMs). Conventional channel pruning techniques for KV cache typically assess each channel in isolation, neglecting the interdependencies among channels. Accordingly, we introduce an **I**nterdependence-**A**ware KV Cache **P**runing (IAP) method, moving beyond the conventional paradigm of isolated channel scoring. Specifically, we first analyze the existence of inter-channel interactions, then reformulate channel selection objective with the channel interdependence component, and propose a graph-based algorithm to identify channels for pruning. Furthermore, IAP mitigates the challenge of query distribution shifts during decoding by strategically retaining high-magnitude key channels. Extensive experiments on LongBench with LLaMA and Mistral models demonstrate that IAP marked improvements in preserving model performance post-pruning compared to established baselines, offering a more robust approach to KV cache compression.

1 Introduction

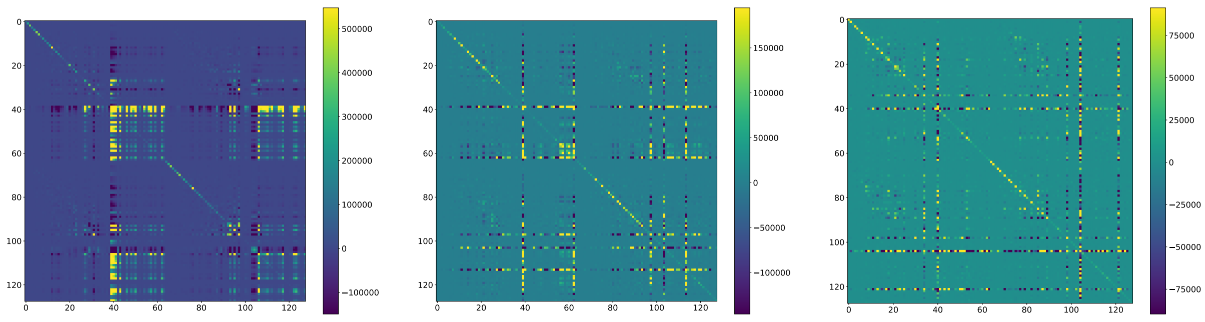
Large Language Models (Achiam et al., 2023; Touvron et al., 2023; Jiang et al., 2023; DeepSeek-AI, 2025) have revolutionized natural language processing by achieving unprecedented performance in tasks such as text generation, reasoning, and contextual understanding. Central to their success is the scaling law principle, which posits that increasing model size and training data leads to emergent capabilities (Kaplan et al., 2020). Alongside the scaling of data and model parameters, recent years have also seen significant research efforts to enhance large language models by expanding their context window size (Team et al., 2024).

While the scaling law has driven significant improvements in model capabilities, it also substantially escalates the computational and GPU memory overhead during inference, particularly in scenarios requiring long-context interactions. Although Key-Value (KV) cache techniques mitigate computational redundancy by avoiding repeated calculations, the linearly growing KV-cache size with sequence length , attention heads and channel dimensions, imposes prohibitive memory pressure. This is particularly problematic in long-context tasks, where the KV cache can become a bottleneck, limiting the model's ability to process extended sequences efficiently.

To this end, researchers have conducted in-depth explorations to reduce the KV cache and have achieved significant progress from the following three perspectives: (1) Model-level: Reducing the number of attention heads in the model (Shazeer, 2019; Ainslie et al., 2023; Brandon et al., 2024). (2) Token-level: Employing KV cache eviction methods (Li et al., 2024; Zhang et al., 2023; Xiao et al., 2024b). (3) Quantization: Applying different quantization strategies to the key and value caches respectively (Liu et al., 2024; Hooper et al., 2024).

In recent years, a channel-level pruning method (Zhang et al., 2024a; Xu et al., 2024) has emerged, which is orthogonal and compatible with the aforementioned techniques, offering substantial promise by significantly reducing the size of the KV cache while preserving model performance. However, existing approaches evaluate channel importance in isolation, neglecting the impact of inter-channel interactions on model effectiveness. Moreover, they overlook the distributional shift between the queries within the observation window and the overall query distribution, which may further degrade performance.

To address these issues, we propose the Interdependence-Aware KV Cache Pruning (IAP) method. During the pruning process, IAP reformulates the optimization objective as a graph-theoretic problem, explicitly modeling and leveraging inter-channel dependencies to minimize performance



(a) Layer 1, Head 0 (b) Layer 15, Head 0 (c) Layer 31, Head 0

Figure 1: This figure illustrates the inter-channel interaction terms., where heatmap$_{i,j} = \mathbf{k}_i^T\mathbf{k}_j \times \mathbf{q}_i^T\mathbf{q}_j$. We can observe that a large number of maximum and minimum values are scattered on the off-diagonal, and as depth increases, these values gradually cluster on the diagonal and in specific rows and columns. Input are randomly selected samples from the Qasper dataset.

loss. In addition, we design a strategy to directly identify and preserve important channels, effectively mitigating the negative impact of query distribution shift. Extensive experiments on LLaMA-3-8B (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023) demonstrate consistent improvements across a range of evaluation metrics.

2 Related Work

While numerous studies have explored methods for reducing the KV cache, this subsection briefly reviews token-level and channel-level pruning techniques, as well as KV cache quantization.

Token-level: These methods retain the KV cache for more important tokens while discarding those of less important ones. Some of these methods operate only during the model's prefilling stage, while others are applied during both prefilling and decoding stages. StreamingLLM (Xiao et al., 2024b) retains initial tokens as attention sinks and uses a sliding window for recent tokens. H2O (Zhang et al., 2023) preserves tokens that have historically accumulated high attention scores, as well as recent tokens. SnapKV (Li et al., 2024) applies a one-dimensional convolution to token attention scores, considering the importance of both the token itself and its neighbors. PyramidKV (Cai et al., 2024) dynamically adjusts the KV cache size for different layers, allocating more cache to lower layers and less to higher layers. CAKE (Qin et al., 2025) assesses layer-specific preferences by considering attention dynamics in both spatial and temporal dimensions, allocates rational cache size for layers accordingly, and manages memory constraints in a cascading manner. SCOPE (Wu et al., 2024) observes that different tasks require varying degrees of compression during the prefilling and decoding stages. It dynamically allocates different KV cache sizes for the two stages and separately perform KV cache optimization during the prefill and decoding phases.

Channel-level: LoRC (Zhang et al., 2024a) identifies the low-rank characteristics of KV cache matrices. They propose a low-rank approximation of KV weight matrices, allowing for plug-in integration with existing transformer-based LLMs without model retraining. THINK (Xu et al., 2024) discovered a highly imbalanced distribution across the channels of the key cache. Thus, they designed an algorithm that depends on query values to prune less important channels.

KV Cache Quantization: To alleviate the memory bottleneck during inference, several studies have explored quantizing the key and value caches into low-bit representations. SmoothQuant (Xiao et al., 2023) demonstrates that the KV cache can be quantized to 8-bit precision with negligible performance loss, offering a practical trade-off between efficiency and accuracy. Building on this, Q-Hitter (Zhang et al., 2024b) proposes a selection mechanism leveraging accumulated attention importance and token-level quantization sensitivity to identify critical tokens that require higher precision to preserve model generalization. KIVI (Liu et al., 2024) highlights the asymmetric roles of key and value caches in attention computation, applying per-channel quantization to key representations to maintain attention selectivity, and per-token quantization to value representations to preserve content fidelity.

The Interdependence-Aware KV Cache Pruning approach introduced in our paper works alongside

Method	**Single-Document QA**			**Multi-Document QA**			**Summarization**			**Few-shot Learning**			**Synthetic**		**Code**		**Avg**
	NrtvQA	***Qasper***	***MF-en***	***HotpotQA***	***2WikiMQA***	***Musique***	***GovReport***	***QMSum***	***MultiNews***	***TREC***	***TriviaQA***	***SAMSum***	***PRe***	***PCount***	***Lcc***	***RB-P***	
LLaMA-3-8B-Instruct, KV-size 256																	
SnapKV																	
+Think (0.5)	23.03	18.80	34.35	**40.89**	30.30	20.61	**20.10**	21.92	**22.10**	56.00	89.57	39.31	**69.50**	**5.93**	59.87	**58.38**	38.17
+IAP (0.5)	**24.20**	**19.15**	**34.99**	40.63	**30.82**	**20.75**	20.02	**22.07**	21.94	**56.50**	**89.71**	**39.41**	68.75	5.62	**59.97**	57.86	**38.27**
+THINK (0.6)	**24.03**	**17.20**	36.31	40.06	**30.90**	19.37	**19.55**	21.75	**21.39**	**49.50**	**89.19**	37.72	**69.50**	**5.89**	57.31	**57.95**	**37.35**
+IAP (0.6)	23.30	17.09	**36.64**	**40.31**	30.61	**20.20**	19.50	**21.91**	20.82	46.50	89.05	**38.34**	68.67	**5.89**	**58.33**	56.86	37.13
LLaMA-3-8B-Instruct, KV-size 512																	
SnapKV																	
+Think (0.5)	24.61	25.54	36.97	41.51	**32.39**	**20.83**	21.37	**22.70**	**23.79**	69.00	90.31	39.85	69.50	**5.84**	61.37	59.22	40.30
+IAP (0.5)	**25.06**	**25.84**	**37.82**	**42.19**	32.25	20.77	**21.48**	22.50	23.69	**69.50**	**90.39**	**40.52**	**69.83**	5.79	**61.84**	**58.52**	**40.50**
+THINK (0.6)	24.84	**23.49**	37.48	**40.42**	**33.15**	19.43	20.66	**22.10**	**22.73**	**59.00**	**90.37**	37.31	**69.50**	**6.39**	**59.62**	58.72	39.07
+IAP (0.6)	**25.12**	22.12	**39.73**	40.28	32.09	**20.00**	**21.07**	21.81	22.63	**59.00**	90.12	**38.40**	**69.50**	6.04	59.37	**58.83**	**39.13**
LLaMA-3-8B-Instruct, KV-size 1024																	
SnapKV																	
+Think (0.5)	25.26	27.06	**39.58**	42.74	32.39	20.1	**23.41**	**22.75**	25.00	**71.50**	**90.43**	**40.74**	**69.25**	5.38	**62.38**	**59.84**	**41.11**
+IAP (0.5)	**25.36**	**28.59**	38.56	**42.95**	**32.40**	**20.31**	23.27	22.73	**25.24**	**71.50**	**90.43**	40.51	69.00	**5.49**	61.83	59.62	**41.11**
+THINK (0.6)	24.40	**27.47**	**38.03**	**42.10**	31.50	20.50	21.72	22.52	23.71	69.50	**90.12**	38.57	**69.50**	**6.26**	58.92	**59.49**	40.27
+IAP (0.6)	**25.13**	26.36	37.78	42.05	**31.84**	**20.91**	**22.19**	**22.67**	**23.91**	**70.00**	89.87	**39.18**	68.92	5.81	**60.01**	59.17	**40.36**
H2O																	
+THINK (0.5)	25.27	20.57	**37.47**	40.91	**31.27**	18.79	**22.43**	22.38	24.65	**46.50**	**90.39**	40.59	69.25	5.09	**61.56**	**58.42**	38.47
+IAP (0.5)	**25.48**	**23.15**	36.85	**41.42**	31.11	**18.98**	22.06	**22.56**	**24.84**	45.50	90.28	**40.93**	**69.33**	**5.45**	61.39	58.25	**38.60**
+THINK (0.6)	24.26	17.37	**37.03**	38.51	**29.93**	**19.99**	21.23	**22.22**	**23.43**	**44.5**	**90.16**	**39.29**	**69.50**	**5.84**	58.23	**58.84**	**37.52**
+IAP (0.6)	**24.30**	**19.14**	36.79	**40.18**	29.71	19.02	**21.24**	22.13	23.01	41.50	89.89	38.57	69.25	5.46	**58.59**	57.68	37.28
LLaMA-3-8B-Instruct, KV-size 2048																	
SnapKV																	
+Think (0.5)	24.84	30.26	39.21	43.22	**33.29**	21.04	24.93	**23.09**	26.04	73.00	**90.37**	**41.21**	**69.25**	5.49	**62.13**	59.72	41.69
+IAP (0.5)	**24.87**	**30.27**	**39.47**	**43.59**	33.22	**21.51**	**25.29**	22.92	**26.15**	**73.50**	**90.37**	40.59	69.08	**5.54**	61.93	**59.89**	**41.76**
+THINK (0.6)	**24.64**	28.57	**40.54**	41.17	31.07	**21.51**	23.32	23.04	**24.94**	72.00	**90.36**	38.67	**69.50**	**6.07**	59.32	59.28	40.87
+IAP (0.6)	25.31	**28.63**	38.80	**42.13**	**31.40**	21.21	**23.69**	**23.24**	24.75	**72.50**	89.86	**38.69**	69.25	5.50	**59.97**	**59.50**	**40.90**
H2O																	
+THINK (0.5)	25.01	25.59	38.79	42.27	**31.26**	20.46	23.71	**23.34**	25.64	**53.00**	**90.37**	**41.29**	**69.50**	5.20	**61.70**	**59.16**	39.77
+IAP (0.5)	**25.77**	**25.73**	**39.74**	**43.41**	30.44	**20.62**	**23.84**	23.24	**25.88**	**53.00**	**90.37**	41.19	**69.50**	**5.32**	61.52	59.02	**39.91**
+THINK (0.6)	**24.43**	22.09	38.73	40.53	29.64	**20.71**	**22.20**	**22.64**	**24.56**	**49.50**	**90.41**	**39.77**	69.20	5.76	59.24	**59.23**	38.66
+IAP (0.6)	24.42	**22.99**	**39.55**	**41.23**	**30.37**	20.65	22.17	22.55	24.47	49.00	89.61	39.12	**69.50**	**5.83**	**60.06**	58.25	**38.73**

Table 1: Comparison of our method against THINK on the LongBench dataset using LLaMA-3-8B as the base model. Experiments were conducted with SnapKV and H2O as distinct preceding token-level methods, for KV cache sizes of 256, 512, 1024, and 2048, and pruning ratios λ of 0.5 and 0.6. The best results are highlighted in bold.

existing token-level pruning and quantization methods, enabling their combination to further reduce memory requirements while maintaining model performance. Unlike previous channel-level pruning techniques, our work emphasizes the critical nature of inter-channel interactions in KV cache pruning and develops a graph-based algorithm to strategically identify channels for pruning.

3 Method

In this section, we provide a detailed description of our proposed method. We begin by establishing the notation that will be used throughout.

Let $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ denote the weight matrices for query, key, and value projections in the attention module. Let $\mathbf{X}_p \in \mathbb{R}^{L_p\times d}$ represent the token matrix for input prompts, where L_p is the prompt length. The key cache and query matrix for the entire prompt sequence are $\mathbf{K}_p = \mathbf{X}_p\mathbf{W}_k, \mathbf{Q}_p = \mathbf{X}_p\mathbf{W}_q \in \mathbb{R}^{L_p\times d}$. Adding the superscript obs indicates the last L_{obs} rows of these matrices, which is referred to as the observation window, such as $\mathbf{X}_p^{obs}, \mathbf{K}_p^{obs}, \mathbf{Q}_p^{obs} \in \mathbb{R}^{Lobs\times d}$.

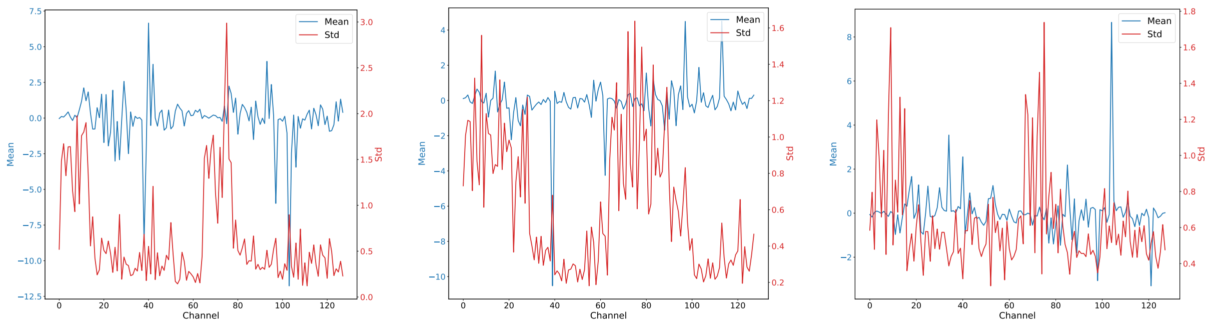
3.1 Preliminary

Previous research (Xu et al., 2024; Liu et al., 2024; Xiao et al., 2024a) has found that in the Key cache, only a subset of channels have large values, while others have smaller values. This implies that these significant channels have a more substantial impact compared to others, suggesting that channel pruning of the key cache can save memory space. At the -th generation step, we obtain token $\mathbf{x}^t$. The query and key vectors for this token are then computed as $\mathbf{q}^t = \mathbf{W}_q\mathbf{x}^t$ and $\mathbf{k}^t = \mathbf{W}_k x^t$, respectively. Subsequently, the key cache $\mathbf{K}^t$ is formed by concatenating $\mathbf{k}^t$ with the previous cache $\mathbf{K}^{t-1}$. The channels to be pruned can then be found by optimizing the following equation:

$$\begin{aligned}
\min_{\mathbf{S}} & \left\| \mathbf{q}^t \mathbf{K}^{\mathbf{t}^T} - \mathbf{q}^t(\mathbf{K}^t\mathbf{S})^T \right\|_F \\
\text{s.t.} & \ \textbf{Trace}(\mathbf{S}) = \lfloor (1-\lambda)d \rfloor \\
& \ \mathbf{S} = \text{diag}(s_1, \ldots, s_D), s_j \in \{0,1\}
\end{aligned} \quad (1)$$

where λ is the channel pruning ratio, and $\mathbf{S}$ is a column selection matrix where diagonal elements of 1 correspond to the channels to be retained.

To actually reduce memory utilization, the corresponding $\mathbf{S}$ in each generation step should be the



(a) Layer 0, Head 0 (b) Layer 15, Head 0 (c) Layer 31, Head 0

Figure 2: This figure displays the mean (blue) and standard deviation (red) for each channel of the query vectors within the observation window across different layers. It can be seen that many channels have a mean close to 0, but the standard deviation is large, indicating that the variation in that channel is unstable. This makes it difficult to substitute for the overall query values by only using the query vectors within the observation window.

same; otherwise, we would still need to store the complete Key cache for subsequent steps. Additionally, this structured pruning can achieve computational acceleration. This means that $\mathbf{S}$ should be computed at an earlier stage and this fixed $\mathbf{S}$ should be used in subsequent processes.

SnapKV confirmed that during the LLM decoding process, the attention pattern of newly generated tokens on the prompts is very similar to the observation window $\mathbf{Q}_p^{obs}$. Therefore, THINK reformulated the optimization problem as:

$$\begin{aligned} \min_{\mathbf{S}} & \left\| \mathbf{Q}_p^{obs}\mathbf{K}_p^T - \mathbf{Q}_p^{obs}\mathbf{S}\mathbf{K}_p^T \right\|_F \\ \text{s.t } & \mathbf{Trace}(\mathbf{S}) = \lfloor (1-\lambda)d \rfloor \\ & \mathbf{S} = \mathrm{diag}(s_1, \ldots, s_D), s_j \in \{0,1\} \end{aligned} \tag{2}$$

Thus, THINK obtains $\mathbf{S}$ during the prefilling stage. The smaller the observation window, the more computational savings can be achieved. To solve this optimization problem, THINK scores each channel. The score for channel j is $\text{score}_j = \|\mathbf{Q}_p^{obs}[:,j]\mathbf{K}_p^{obs}[:,j]^T\|_F$, and the channels with the highest scores are retained.

3.2 Methodology

The optimization equation Eq.(2) can actually be a special case of CR decomposition (Drineas et al., 2006): assume $\mathbf{W}$ can be decomposed into the form $\mathbf{W_1W_2}$, and $\mathbf{S}_k$ is a column selection matrix that selects k columns, approximating the original equation as $\mathbf{W} \approx \mathbf{W_1}\mathbf{S}_k\mathbf{S}_k^T\mathbf{W_2}$. Eq.(2) signifies performing a CR decomposition on $\mathbf{W} = \mathbf{Q}_p^{obs}\mathbf{K}_p^T$. This is achieved by selecting k columns of $\mathbf{Q}_p^{obs}$ and k rows of $\mathbf{K}_p^T$ using $\mathbf{S}_k$, where $k = \lfloor (1-\lambda d) \rfloor$. Furthermore, $\mathbf{S}_k$ is a square matrix, and consequently, $\mathbf{S}_k\mathbf{S}_k^T = \mathbf{S}_k$.

Let $A = \{i|s_i = 1\}, B = \{i|s_i = 0\}$, with $A \cap B = \varnothing$, represent the index sets of retained and discarded channels, respectively. Let $\mathbf{q}_i, \mathbf{k}_i$ denote the i-th columns of $\mathbf{Q}_p^{obs}$ and $\mathbf{K}_p$, respectively. Then we can obtain(See Appendix A):

$$\begin{aligned} & \left\| \mathbf{Q}_p^{obs}\mathbf{K}_p^T - \mathbf{Q}_p^{obs}\mathbf{S}\mathbf{K}_p^T \right\|_F^2 \\ & = \sum_{i \in B}\sum_{j \in B} \mathbf{k}_i^T\mathbf{k}_j \times \mathbf{q}_i^T\mathbf{q}_j \end{aligned} \tag{3}$$

Considering a single i in set B, the error it introduces is:

$$\begin{aligned} \Delta_i &= \sum_{j \in B} \mathbf{k}_i^T\mathbf{k}_j \times \mathbf{q}_i^T\mathbf{q}_j \\ &= \sum_{j \in B-i} \mathbf{k}_i^T\mathbf{k}_j \times \mathbf{q}_i^T\mathbf{q}_j + \|\mathbf{q}_i\mathbf{k}_i^T\|_F^2 \end{aligned} \tag{4}$$

Here, the first term describes the impact of interactions between channels and the second term describes the impact of the individual channel itself on the error (which is the scoring method considered in THINK). If $\mathbf{k}_i$ and $\mathbf{q}_i$ were random vectors, the dot product of these random vectors in high-dimensional space tends towards 0, indicating almost no interaction between channels (Van Handel, 2014). However, $\mathbf{k}_i$ and $\mathbf{k}_i$ are generated by large models, so the channels cannot be completely random. Secondly, since the $\mathbf{k}_i$ vector only intercepts tokens in the observation window, its length is not large and does not fall into the high-dimensional category.

As shown in Fig. 1, there are still many regions with extremely large or small values, indicating that inter-channel interactions are non-negligible and play a significant role in performance degradation.

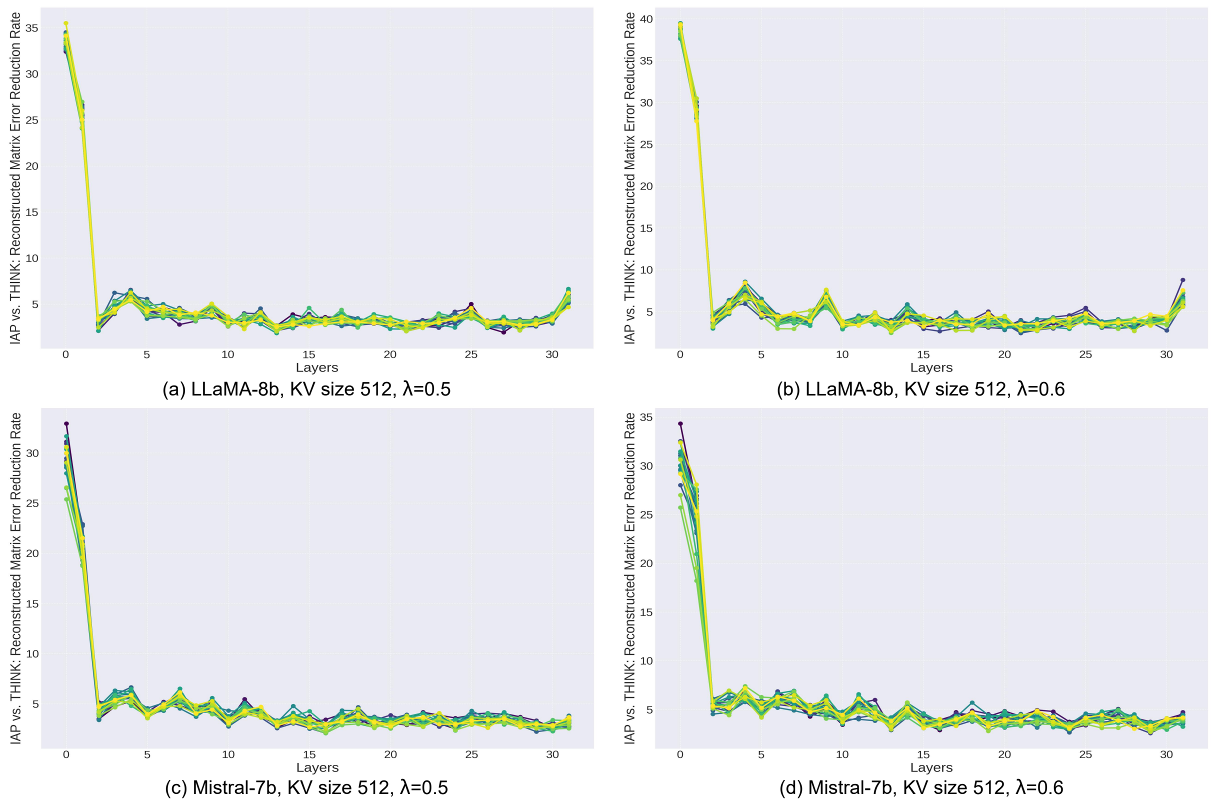


Figure 3: This figure shows the performance of our method relative to THINK, in optimizing equation Eq.(2). Different colored curves represent different input samples(20 samples in total). The x-axis represents the depth of the decode layer, and the y-axis represents how much smaller our method's reconstructed matrix error compared to THINK. LLaMA's test samples were randomly selected from the NrtvQA dataset, while Mistral's test samples were from the Qasper dataset.

Furthermore, we observe that as the model depth increases, these extreme values tend to concentrate along the diagonal or in specific rows and columns, suggesting that channel correlations diminish progressively in deeper layers.

This optimization problem can be equivalently transformed into a graph problem: a complete undirected graph $\mathcal{G} = (V, E)$ with d vertices, $|V| = d$. Each node has its own weight $w_v^i = \|\mathbf{q}_i \mathbf{k}_i^T\|_F^2$, $\forall i \in V$, and each edge also has a weight $w_{(i,j)}^e = 2 \times \mathbf{k}_i^T \mathbf{k}_j \times \mathbf{q}_i^T \mathbf{q}_j$, $\forall (i,j) \in E$. The problem is how to select a subgraph $\mathcal{G}'$ with λd nodes from $\mathcal{G}$ such that the sum of weights of $\mathcal{G}'$ is minimized. Ultimately, the nodes in $\mathcal{G}'$ correspond to the channels that need to be pruned.

We designed a greedy algorithm (Algorithm 1) for this problem that considers edge weights. This algorithm initializes the scores of all channels (**Score**) using the inherent weights of the nodes. In each selection process, the node j with the minimum score is selected and added to the discard set **Drop**, and **Score** is updated using the edge weights between this node and the remaining nodes.

Algorithm 1 Search Algorithm Considering Inter-Channel Interactions

Require: Inputs $\mathbf{Q}_p, \mathbf{K}_p \in \mathbb{R}^{L_p \times d}$, Prun ratio λ, Observation size L_{obs}, Recent size L_r.
Ensure: **Drop**: Set of pruned channel indices.

```
 1: Initiate Score ← ∅, Drop ← ∅, drop_num = $\lfloor \lambda d \rfloor$
 2: for $i$ from 0 to $d-1$ do
 3:     $\mathbf{q}_i = \mathbf{Q}_p[-L_{obs}:, i]$, $\mathbf{k}_i = \mathbf{K}_p[:L_p - L_r, i]$
 4:     Score = Score $\cup \{(i, \|\mathbf{q}_i \mathbf{k}_i^T\|_F^2)\}$
 5: end for
 6: for $i$ from 1 to drop_num do
 7:     $j, s_j \leftarrow$ Get_Min_Score(Score).
 8:     Score ← Score $\setminus \{(j, s_j)\}$.
 9:     Drop ← Drop $\cup \{j\}$
10:     for every $(k, s_k)$ in Score do
11:         $s_k \leftarrow s_k + 2 \times \mathbf{k}_k^T \mathbf{k}_j \times \mathbf{q}_k^T \mathbf{q}_j$.
12:     end for
13: end for
14: return Drop
```

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NrtvQA	*Qasper*	*MF-en*	*HotpotQA*	*2WikiMQA*	*Musique*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PRe*	*PCount*	*Lcc*	*RB-P*	
Mistral-7B-Instruct-v0.2, KV-size 256																	
SnapKV																	
+Think (0.5)	21.73	**23.90**	**47.66**	37.66	**22.15**	15.58	**21.77**	**23.20**	22.62	**59.00**	**85.67**	40.94	**83.95**	2.43	**54.63**	50.77	38.35
+IAP (0.5)	**22.84**	23.50	47.23	**37.79**	21.96	**16.58**	21.60	22.70	**22.96**	**59.00**	85.60	**41.02**	83.28	**3.06**	54.17	**51.23**	**38.41**
H2O																	
+THINK (0.5)	**22.39**	**23.38**	42.04	29.44	21.17	12.94	22.08	**23.07**	22.77	40.00	83.86	40.81	**83.40**	**3.22**	**52.44**	49.87	35.81
+IAP (0.5)	21.63	22.92	**43.04**	**29.82**	**22.23**	**13.47**	**22.42**	23.02	**22.93**	**40.50**	**83.88**	**41.11**	81.28	3.09	52.05	**49.91**	**35.83**
Mistral-7B-Instruct-v0.2, KV-size 512																	
SnapKV																	
+Think (0.5)	**24.51**	**28.54**	48.76	38.30	**24.30**	17.10	23.31	**24.07**	**24.42**	66.00	85.85	**41.94**	86.07	**3.02**	**56.20**	53.24	40.35
+IAP (0.5)	24.41	27.81	**49.93**	**38.57**	23.71	**17.29**	**23.58**	23.56	24.37	**66.50**	**85.90**	41.91	**87.36**	3.00	56.19	**53.37**	**40.47**
H2O																	
+THINK (0.5)	**22.99**	**25.73**	**44.43**	**31.85**	23.35	13.90	**23.52**	22.77	24.09	**41.50**	**85.55**	41.42	83.45	2.84	**54.65**	**50.90**	**37.06**
+IAP (0.5)	22.81	24.85	43.97	30.99	**23.37**	**14.58**	23.36	**22.97**	**24.13**	41.00	85.38	**41.82**	**84.32**	**3.06**	54.55	50.66	36.99
Mistral-7B-Instruct-v0.2, KV-size 1024																	
SnapKV																	
+Think (0.5)	25.68	29.97	49.36	40.64	**24.98**	**19.49**	**25.62**	23.92	**26.12**	**69.5**	**86.67**	**42.26**	85.39	**2.89**	57.26	53.58	**41.46**
+IAP (0.5)	**25.77**	**30.16**	**49.71**	**40.80**	24.92	19.40	25.19	**23.93**	25.83	**69.50**	86.40	42.15	**85.68**	2.81	**57.35**	**53.79**	**41.46**
H2O																	
+THINK (0.5)	**24.15**	**28.25**	**46.66**	35.52	**24.10**	14.63	**24.88**	23.26	**25.72**	**45.00**	86.16	**43.09**	83.7	**3.41**	55.90	**52.74**	**38.57**
+IAP (0.5)	23.88	27.95	46.10	**35.60**	**24.10**	**15.18**	24.76	**23.31**	25.65	44.50	**86.64**	42.91	**84.20**	3.07	**56.21**	52.20	**38.52**

Table 2: Comparison of our method against THINK on the LongBench dataset using Mistral-7B-Instruct as the base model. Experiments were conducted with SnapKV and H2O as distinct preceding token-level methods, for KV cache sizes of 256, 512 and 1024, and pruning ratios λ of 0.5. The best results are highlighted in bold.

3.3 Protecting Salient Channels

The method proposed in the preceding sections is entirely based on Eq.(2), which utilizes $\mathbf{Q}_p^{obs}$ for optimization. However, the issue is that during the decoding stage, the distribution of new $\mathbf{q}^t$ may differ significantly from the $\mathbf{q}$ in $\mathbf{Q}_p^{obs}$. As shown in Fig. 2, in many channels of the query vector, the standard deviation is very large, even much larger than the mean. This implies that optimizing Equation 2 does not necessarily yield a good solution for Eq.(1).

Inspired by studies such as LLM.int8() (Dettmers et al., 2022) and AWQ (Lin et al., 2024), we directly retain channels in $\mathbf{K}_p$ that have a large norm, excluding them from the pruning process. If these large-norm channels are prematurely eliminated due to small interaction terms, it could lead to a significant reconstruction loss for future, newly appearing $\mathbf{q}^t$. Therefore, it is necessary to preserve them.

We mark channels that exceed the mean plus one standard deviation as important channels. The proportion of these channels, $p \in (0, 1)$, can be very large or very small. Thus, it should be constrained by two thresholds $0 < a < b < 1$, and we take $p_{remain} = \min(\max(p, a), b)$ as the final limit. Under the premise of having the same observation window, the larger the kv size, the smaller the proportion $\mathbf{Q}_p^{obs}$ occupies in the total. A smaller sample size implies greater bias, so both thresholds a and b should be increased with long observation window size.

4 Experiments

4.1 Settings

Benchmark Datasets. We evaluate our method on the LongBench (Bai et al., 2024) dataset and compare it with state-of-the-art KV cache compression techniques. The LongBench dataset is widely used for testing KV cache compression techniques and is designed to assess a model's comprehension capabilities in long-context scenarios. It comprises 17 sub-datasets across 6 different tasks, covering single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. These datasets all feature very long prompts, posing challenges for large models to condense and extract key information.

Baseline. We use LLaMA-3-8B-Instruct and Mistral-7B-Instruct-v0.2 as our base test models, and select THINK as the baseline approach for experiments. THINK proposed an effective method for channel-wise KV cache reduction. Both THINK and IAP require the use of token-level compression methods to reduce prompt tokens beforehand. Here, we choose the token-level compression methods H2O and SnapKV: the former is designed to reduce memory usage by dynamically balancing recent tokens with Heavy Hitter (H2) tokens, while the latter introduces an automated compression mechanism that selects clustered, important KV positions for each attention head, optimizing the KV cache without sacrificing performance. We conduct experiments using NVIDIA RTX 3090.

Implementation Details. LLaMA-3-8B-Instruct

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NrtvQA	*Qasper*	*MF-en*	*HotpotQA*	*2WikiMQA*	*Musique*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PRe*	*PCount*	*Lcc*	*RB-P*	
KIVI(4/4)	19.52	19.45	32.96	29.96	26.86	10.10	24.10	20.73	25.63	63.00	84.13	41.34	9.00	4.50	58.89	53.21	32.72
+IAP(0.4)	19.33	18.33	31.79	29.46	27.12	9.60	23.86	20.63	26.19	63.00	83.51	41.31	7.00	4.00	58.34	51.76	32.20
+IAP(0.5)	19.29	18.30	30.65	28.56	24.02	9.08	23.95	20.50	25.64	63.00	83.12	41.83	5.50	3.50	57.48	48.53	31.43

Table 3: Performance evaluation of combining IAP with KIVI. 4/4 indicates that both the key and value are quantized using 4 bits. IAP(0.4) indicates using the IAP method with a pruning rate of 0.4.

Recent Size	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg
	NrtvQA	*Qasper*	*MF-en*	*HotpotQA*	*2WikiMQA*	*Musique*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PRe*	*PCount*	*Lcc*	*RB-P*	
								KV-size 1024, $\lambda = 0.5$									
32	25.64	27.71	40.05	42.63	32.91	20.27	23.42	22.45	25.50	71.50	90.43	40.21	69.50	5.18	62.27	59.56	**41.20**
64	26.00	27.00	39.03	42.87	31.79	19.58	23.63	22.38	25.39	71.50	90.43	40.23	68.92	5.83	61.90	60.36	41.05
								KV-size 2048, $\lambda = 0.5$									
32	25.05	29.83	38.92	43.56	33.01	21.1	24.87	23.31	26.60	73.50	90.37	40.79	69.08	5.75	61.47	58.65	41.61
64	24.66	30.44	39.53	43.41	32.74	21.30	25.04	22.89	26.32	72.50	90.37	40.84	69.50	6.01	62.01	58.89	**41.65**

Table 4: Performance comparison of key cache pruning with varying recent-sizes(without protecting salient weights). We evaluated the performance for various Recent Sizes when the KV cache size was set to 1024 and 2048.

and Mistral-7B-Instruct-v0.2 are directly obtained from HuggingFace (Wolf et al., 2020). For SnapKV, we set the kernel size to 7 and the window size to 32, and used max pooling as the pooling method. Additionally, we chose not to prune the most recent tokens and newly generated keys, as these tokens have a significant impact on performance. Given their small number, they do not affect the compression rate. Due to the presence of these unpruned tokens, we performed an alignment operation on Eq.(2), adjusting $\mathbf{K}^p$ to $\mathbf{K}^p[: -\text{recent\_size}]$ to ignore these tokens, thereby excluding them from compression. This adjustment ensures that the performance-critical tokens are preserved, maintaining the integrity of the model's output while still achieving a satisfactory compression ratio.

4.2 Results

Reconstruction Error Comparison: In Fig. 3, we compare the solution quality of our proposed method IAP and THINK on Eq. (2) using reconstruction error as the evaluation metric. We randomly sampled several examples from the Qasper dataset for this comparison. It can be observed that in the shallow layers of large language models, our method significantly outperforms THINK in terms of reconstruction error, highlighting the critical importance of inter-channel interactions in KV cache compression. In deeper layers, our method still achieves lower reconstruction errors than THINK, though the improvements are less pronounced. This trend is consistent across different model choices and various settings of λ.

These observations align with the results shown in Fig. 1, where we hypothesize that channel interdependence is stronger in shallower layers, leading to greater gains when taken into account. In contrast, while interdependence still exists in deeper layers, it is relatively weaker, hence the reduced benefit. Overall, Fig. 1 confirms the presence of inter-channel dependencies, validating the motivation behind this work, and Fig. 3 demonstrates that incorporating inter-channel interactions can effectively reduce reconstruction error, thereby verifying the effectiveness of our proposed method.

Results on LongBench: Tab. 1 and Tab. 2 present comparative experimental results for IAP and THINK on the LongBench dataset, using the LLaMA-3-8B and Mistral-7B models, respectively. SnapKV and H2O were employed as distinct preceding token-level pruning methods.

When LLaMA-3-8B was the base model, IAP demonstrated superior performance over THINK, irrespective of the preceding token-level pruning method applied. IAP achieved higher average scores in most experimental settings and also scored higher on a majority of sub-tasks. For instance, with a KV cache size of 512 and λ=0.5, IAP outperformed THINK on 11 out of 16 sub-tasks, and its average score of 40.50 surpassed THINK's 40.30.

With Mistral-7B as the base model, IAP consistently achieved higher average scores than THINK across various experimental setups when SnapKV was used as the preceding pruning method. For example, with a KV cache size of 256 and λ=0.5, IAP's average score of 38.41 exceeded THINK's 38.35. However, when H2O was the preceding pruning method, the performance difference between IAP and THINK became marginal. For in-

[a,b]	**Single-Document QA**			**Multi-Document QA**			**Summarization**			**Few-shot Learning**			**Synthetic**		**Code**		**Avg**
	NrtvQA	***Qasper***	***MF-en***	***HotpotQA***	***2WikiMQA***	***Musique***	***GovReport***	***QMSum***	***MultiNews***	***TREC***	***TriviaQA***	***SAMSum***	***PRe***	***PCount***	***Lcc***	***RB-P***	
								KV-size 512, $\lambda = 0.6$									
$[0, 0]$	25.12	22.12	39.73	40.28	32.09	20.0	21.07	21.81	22.63	59.0	90.12	38.4	69.5	6.04	59.37	58.83	**39.13**
$[3, 7]$	25.18	23.28	38.12	40.14	31.71	20.33	20.71	22.28	22.67	59.0	90.37	38.6	69.5	5.92	58.99	58.4	39.08
								KV-size 1024, $\lambda = 0.6$									
$[0, 0]$	25.15	25.32	39.08	41.48	31.19	20.33	22.24	22.89	23.87	69.0	89.91	38.11	69.17	6.03	59.82	59.09	40.17
$[3, 7]$	25.13	26.36	37.78	42.05	31.84	20.91	22.19	22.67	23.91	70.0	89.87	39.18	68.92	5.81	60.01	59.17	**40.36**
								KV-size 2048, $\lambda = 0.6$									
$[0, 0]$	24.51	27.75	39.97	42.25	32.8	20.55	23.21	22.99	24.73	72.5	90.36	38.04	68.54	5.75	58.84	58.08	40.68
$[3, 7]$	25.31	28.63	38.8	42.13	31.4	21.21	23.69	23.24	24.75	72.5	89.86	38.69	69.25	5.5	59.97	59.5	**40.90**

Table 5: Performance comparison of key cache pruning with varying [a,b]. We tested the results for KV cache sizes of 512, 1024, and 2048, where the hyperparameter pair $[a, b]$ was set to $[0, 0]$ and $[3, 7]$.

stance, at a KV cache size of 256 and λ=0.5, IAP's average score (35.83) was only 0.02 points higher than THINK's (35.81). Conversely, with a KV cache size of 1024, IAP's average score was 0.05 points lower than THINK's. A possible reason for this discrepancy is that SnapKV selects tokens using 1D convolution and pooling, while H2O considers each token individually. Consequently, the tokens selected by H2O exhibit greater diversity, potentially leading to a larger deviation between the keys in the observation window and the overall key distribution.

4.3 Compatibility with Quantization Methods

Quantizing the KV cache into lower-precision formats is also a crucial approach to reducing memory overhead. In this subsection, we explore whether the proposed Interdependence-Aware KV Cache Pruning (IAP) can be effectively integrated with existing quantization techniques such as KIVI, achieving significant memory savings while maintaining model accuracy. Such compatibility would further extend the practical applicability of our method.

As shown in Tab. 3, we integrate IAP with KIVI and report the performance. Specifically, we take KIVI with 4-bit quantization for both keys and values as the baseline, and combine it with IAP using pruning ratios of 0.4 and 0.5. Under these settings, we achieve a 20% to 25% reduction in KV cache size, with only slight degradation in the average performance.

Interestingly, on the MultiNews Summarization task, IAP not only reduces memory usage but also brings a slight improvement in performance: IAP(0.4) achieves a score of 26.19, outperforming the KIVI baseline score of 25.63.

4.4 Ablation Study

Impact of Varying Windows Sizes: Tab. 4 illustrates the model's performance across a range of window sizes. It is recognized that increasing the window size can produce an effect comparable to that of explicitly protecting important channels. Consequently, to isolate the distinct impact of window size modifications, this analysis was conducted without the concurrent application of important channel protection strategies. The results presented indicate that for a KV cache size of 1024, simply expanding the window size does not yield a discernible improvement in performance. In contrast, when the KV cache size is increased to 2048, a slight performance enhancement is observed with larger window sizes.

Influence of [a, b] Parameter Configuration on Performance. Tab. 5 details the impact of different [a,b] parameter pair configurations on model performance across varying KV cache sizes. The results clearly demonstrate a dependency between the optimal [a,b] settings and the available KV cache size. Specifically, for larger KV cache configurations, such as those with sizes of 1024 and 2048, a distinct trend emerges: employing larger values within the [a,b] pair yields noticeable performance enhancements. Conversely, this trend inverts when operating with smaller KV cache sizes.

5 Conclusion

This paper proposes a channel-level pruning method for the KV cache, viewing the channel selection task as a CR decomposition problem. We further analyze how inter-channel interactions contribute to performance degradation and reformulate the optimization as a graph-based theory problem that explicitly models these dependencies. Moreover, we observe instability in the query matrix within the observation window and mitigate this issue by retaining key channels deemed important. Extensive experiments on the LongBench dataset show that our method achieves substantial performance improvements over existing approaches.

6 Limitations

Choice of Hyperparameters: For different KV-sizes, our method requires selecting specific values for $[a, b]$ to enhance performance. In this paper, we empirically set different values of $[a, b]$ for each KV size, which lacks flexibility. However, designing an adaptive approach to automatically determine the optimal $[a, b]$ pair for varying KV sizes remains an open and worthwhile direction for future research.
FlashAttention Support: In our method, the key cache consists of two parts with different numbers of key-value channels: a pruned part and an unpruned part, with the former being predominant. However, FlashAttention does not support processing such key matrices. During computation, we still need to convert the key cache back to its original shape before calling FlashAttention. Although this remains a common limitation of current KV cache pruning methods, developing pruning strategies that integrate more effectively with acceleration techniques such as FlashAttention represents a promising avenue for future research.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. 2024. Reducing transformer key-value cache size with cross-layer attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

DeepSeek-AI. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Preprint*, arXiv:2208.07339.

Petros Drineas, Ravi Kannan, and Michael W Mahoney. 2006. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Sophia 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*, 37:1270–1303.

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.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for llm compression and acceleration. *Preprint*, arXiv:2306.00978.

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. *arXiv preprint arXiv:2402.02750*.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. Cake: Cascading and adaptive kv cache eviction with layer preferences. *arXiv preprint arXiv:2503.12491*.

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

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

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

Ramon Van Handel. 2014. Probability in high dimension. *Lecture Notes (Princeton University)*, 2(3):2–3.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. 2024. Scope: Optimizing key-value cache compression in long-context generation. *arXiv preprint arXiv:2412.13649*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2024a. Smoothquant: Accurate and efficient post-training quantization for large language models. *Preprint*, arXiv:2211.10438.

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

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2024. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. 2024a. Lorc: Low-rank compression for llms kv cache with a progressive compression strategy. *arXiv preprint arXiv:2410.03111*.

Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. 2024b. Q-hitter: A better token oracle for efficient llm inference via sparse-quantized kv cache. *Proceedings of Machine Learning and Systems*, 6:381–394.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Proof of Equation 3

First, we present a lemma: the Frobenius norm of a matrix $A \in \mathbb{R}^{m\times n}$ is equal to the square root of the trace of its Gram matrix:

$$\|A\|_F \triangleq \left(\sum_{j=1}^{n}\sum_{i=1}^{m}|a_{ij}|^2\right)^{\frac{1}{2}} = \left[\textbf{Trace}\left(A^TA\right)\right]^{\frac{1}{2}}$$

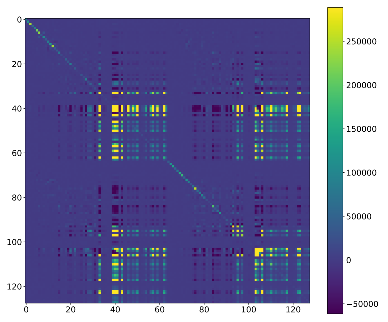
For Eq.(2), let $A = \{i \mid s_i = 1\}$ and $B = \{i \mid s_i = 0\}$, where $A \cap B = \varnothing$, representing the index sets of the retained and discarded channels, respectively. Let $\mathbf{q}_i$ and $\mathbf{k}_i$ denote the i-th columns of $\mathbf{Q}_p^{obs}$ and $\mathbf{K}_p$. Then, we have:

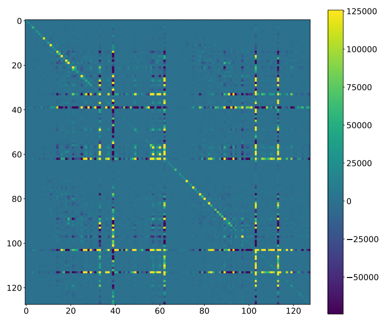
$$\begin{aligned}
\|\mathbf{Q}_p^{obs}\mathbf{K}_p^T - \mathbf{Q}_p^{obs}\mathbf{S}\mathbf{K}_p^T\|_F^2 &= \left\|\sum_{i\in B}\mathbf{q}_i\mathbf{k}_i^T\right\|^2 \\
&= \textbf{Trace}\left(\left(\sum_{i\in B}\mathbf{q}_i\mathbf{k}_i^T\right)\left(\sum_{i\in B}\mathbf{q}_i\mathbf{k}_i^T\right)^T\right) \\
&= \textbf{Trace}\left(\sum_{i\in B}\sum_{j\in B}\mathbf{q}_i\mathbf{k}_i^T\mathbf{k}_j\mathbf{q}_j^T\right) \\
&= \sum_{i\in B}\sum_{j\in B}\textbf{Trace}(\mathbf{q}_i\mathbf{k}_i^T\mathbf{k}_j\mathbf{q}_j^T) \\
&= \sum_{i\in B}\sum_{j\in B}\mathbf{k}_i^T\mathbf{k}_j\textbf{Trace}(\mathbf{q}_i\mathbf{q}_j^T) \\
&= \sum_{i\in B}\sum_{j\in B}\mathbf{k}_i^T\mathbf{k}_j \times \mathbf{q}_i^T\mathbf{q}_j
\end{aligned}$$

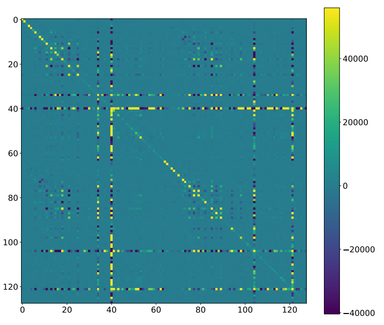
Therefore, the reconstruction error can be expressed as the sum of the dot products of the key vectors and query vectors corresponding to any pair $\{i, j\}$ in the set B.

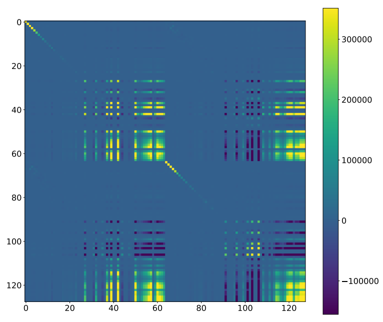
B Comparisons of Generation Speed

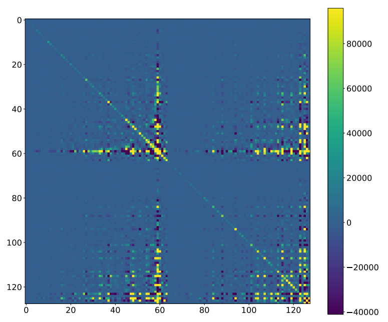
We evaluated three key metrics: Time To First Token (TTFT), Time Per Output Token (TPOT), and memory usage, with detailed results presented in Tab. 6. These experiments were conducted on an NVIDIA L20 GPU using the NrtvQA dataset and a batch size of 15. On average, each batch


(a) LLaMA, Layer 1, Head 0


(b) LLaMA, Layer 15, Head 0


(c) LLaMA, Layer 31, Head 0


(d) Mistral, Layer 1, Head 0


(e) Mistral, Layer 15, Head 0

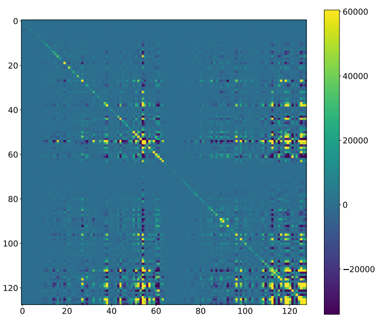

(f) Mistral, Layer 31, Head 0

Figure 4: These figures illustrate inter-channel interaction terms, where heatmap$_{i,j}$ $= \mathbf{k}_i^T \mathbf{k}_j \times \mathbf{q}_i^T \mathbf{q}_j$. For the LLaMA model, the NrtvQA dataset was employed, and for the Mistral model, the Qasper dataset was employed.

Method	KIVI(4/4)	+IAP(0.5)	+IAP(0.6)
Memory (GB)	44.9	38.9	38.2
TTFT (s)	10.4	12.2	12.4
TPOT (ms/token)	10.3	10.9	11.0

Table 6: Comparisons of TTFT (Time To First Token), TPOT (Time Per Output Token) and memory usage on LLaMA-2-7B.

KV size \ Method	SnapKV	H2O
256	[0,0]	–
512	[0,0]	–
1024	[3,7]	[3,7]
2048	[3,7]	[3,7]

Table 7: Selection of [a, b] pairs for LLaMA-3-8B

involved 61,455 input tokens and 1,920 output tokens. Our method, IAP, achieved significant GPU memory reductions: 15.53% at a 0.5 pruning ratio and 17.28% at a 0.6 pruning ratio. While IAP yielded these memory savings, TTFT experienced a slight increase, an effect attributed to the computational overhead of managing pruned channels during the prefilling phase. Conversely, TPOT remained largely unaffected after applying IAP. This stability is primarily due to FlashAttention's current limitations in supporting mixed-dimensionality computation, as detailed further in Section 6.

KV size \ Method	SnapKV	H2O
256	[3,7]	[0,0]
512	[5,10]	[0,0]
1024	[5,10]	[5,10]

Table 8: Selection of [a, b] pairs for Mistral-7B

C Selection of [a, b] Pairs in Experiments

To mitigate query distribution shifts during decoding, IAP preserves salient channels in the key. The specific $[a, b]$ parameter pairs that IAP employs for this strategy in experiments with LLaMA-3-8B and Mistral-7B models are detailed in Tab. 7 and 8. This configuration adheres to the principle, discussed in Section 3.3, that larger KV cache sizes generally necessitate correspondingly larger [a,b] pairs.

D More Visualization Examples

In this section, we present heatmap visualizations for the LLaMA-3-8B and Mistral-7B models on additional datasets. Fig. 4a,4b,4c illustrates these visualizations for LLaMA-3-8B using the NrtvQA dataset, while Fig.4d,4e,4f depicts them for Mistral-7B on the Qasper dataset. As can be observed, the heatmap patterns are consistent with those described in Sec. 3.2.