Multi-Granular KV Cache Sparsification for Efficient Large Language Model Inference

Xiang Jiang
NanJing University
221900183@smail.nju.edu.cn

Xinxin Guo
NanJing University
221900268@smail.nju.edu.cn

Fengming Zhong
NanJing University
221900108@smail.nju.edu.cn

Abstract

Large Language Models (LLMs) have revolutionized the field of natural language processing, achieving unprecedented performance in various applications. During the inference process of LLMs, caching the Attribute–Value Pair (KV Cache) of historical tokens can accelerate generation, but as the context length increases, it brings significant memory and computational overhead, becoming the main bottleneck for efficiency improvement. This paper focuses on the tokens of the input sequence in the prefil phase, aiming to explore and evaluate KV Cache sparsification strategies at different granularities to reduce memory and latency costs while ensuring generation quality. We implemented multiple pruning strategies, including the token dimension (retaining the nearest/farthest tokens or screening high-weight tokens based on historical attention means), the layer dimension (inter-layer pruning, dynamically adjusting the number of retained tokens), and combined strategies (layer pruning followed by token pruning). Experiments on the Llama-3.1-8B-Instruct model and the SQuAD dataset show that these strategies can effectively reduce the memory footprint of KV Cache. Among them, the token pruning strategy based on attention weights performs better under the same memory constraints, reducing the cache by 25

1 Introduction

In recent years, large language models (LLMs) have achieved remarkable success in natural language processing tasks and have demonstrated state-of-the-art performance in various tasks. A key principle is the "scaling law", which states that LLMs exhibit emergent capabilities as the model size increases, thereby enhancing their ability to understand complex contexts and process long sequences. This growth in capacity enables LLMs to generate coherent and contextually accurate responses and support a variety of downstream applications. In autoregressive generation tasks, to avoid recomputing the key (Key) and value (Value) matrices of historical tokens, LLMs cache these intermediate results, which is known as KV Cache. However, during the inference phase, as the length of the input sequence increases, the efficiency of the model faces severe challenges. Among them, although the Key-Value Cache (KV Cache) can accelerate the autoregressive generation process, its video memory occupancy and computational overhead increase linearly or even quadratically with the input length, becoming the main bottleneck for inference efficiency. For example, as we have observed, when a Llama2-7B model processes a context with a length of 2048 and a batch size of 13, the video memory occupancy of only the KV Cache is as high as 13GB, almost equivalent to the model parameters themselves, which can cause a huge memory burden when processing long sequences.

Currently, mainstream inference frameworks (such as HuggingFace Transformers) cache the KV pairs of all historical tokens by default, resulting in a waste of a large amount of video memory and computing resources. However, not all historical tokens are equally important for the current generation step. Therefore, how to efficiently prune the KV Cache to reduce resource consumption while ensuring generation quality has become an urgent problem to be solved.

To address this challenge, this paper aims to explore and evaluate KV Cache pruning strategies at different granularities, with the goal of reducing memory usage and inference latency while maintaining the quality of model generation. It should be noted that the KV Cache pruning in this project only targets the input sequence (encoder-side tokens) and does not involve the tokens generated during the decoding phase (decode-side tokens).

Specifically, by modifying the attention mechanism of the Llama-3.1-8B-Instruct model, we implemented and compared multiple KV Cache pruning strategies covering different granularities: the nearest and farthest retention and attention weight screening strategies based on the token dimension, and the dynamic pruning method based on the layer dimension. We further proposed a hybrid strategy that jointly prunes across layers and tokens. The experiments were conducted on the SQuAD dataset, using the EM Score and F1 Score as the main evaluation metrics while monitoring changes in GPU memory usage and inference latency.

The results show that the token pruning strategy based on attention weights only causes a 0.8% drop in EM score while reducing the cache by 25%, demonstrating excellent performance retention. Notably, different transformer layers exhibit significantly different sensitivities to pruning, with deeper networks often able to tolerate a greater proportion of sparsification.

The main innovation of this study is the proposal of a practically deployable KV Cache optimization scheme, including a dynamic pruning criterion based on the attention mechanism and an inter - layer differential retention strategy. The experiments not only verify the effectiveness of these methods but also systematically compare and analyze the performance of various KV Cache sparsification strategies with different granularities in LLMs inference, providing an empirical basis for understanding and selecting appropriate pruning methods. We provide the implementation details of the Token, Layer, and combined strategies and conduct quantitative experimental evaluations, clearly demonstrating the trade - offs between these strategies in terms of accuracy and video memory occupancy. Through detailed experimental data, we quantify the effects of different strategies and find that, under the same video memory occupancy, the pruning strategy based on attention weights may exhibit better performance. These findings provide an important reference for the subsequent efficient inference of large language models, especially in resource - constrained environments and long - context applications. The optimization techniques implemented in the study have been verified by modifying the LlamaAttention module of HuggingFace and can be directly applied to the existing inference framework.

2 Related Work

KV Cache optimization aims to reduce the video memory required to store historical Key and Value vectors and accelerate access to them. The work in this area can be roughly divided into the following categories:

2.1 Categories of Optimization Methods

- **Hardware and system-level optimization**: This type of method accelerates the access to KV Cache by improving memory management, data layout, or parallel computing strategies. For example, vLLM introduced PagedAttention, which improves the throughput and memory utilization of KV Cache through unified memory management.

- **Compression and Quantization**: Compressing or quantizing the KV Cache itself is a direct way to reduce its memory footprint. This includes low-bit quantization of KV values or the use of low-rank decomposition techniques to compress the KV matrix.

- **Pruning/Eviction**: The goal of this type of method is to identify and remove redundant or unimportant information from the KV Cache.

2.2 Our Focus: Pruning and Eviction

Our work mainly belongs to the pruning and eviction of KV Cache. Existing pruning methods usually follow one of the following strategies:

- **Time-series-based pruning**: The simplest method is to retain the KV values of the most recent N tokens, as the most recent information is usually more important for the current prediction. For example, Xiao and et al. (2023) proposed the attention sink mechanism, which combines retaining a small number of the earliest tokens and the most recent tokens to handle infinitely long contexts. The strategy of "retaining the most recent and the earliest N tokens" implemented in our work falls into this category.

- **Attention weight-based pruning**: This method assumes that tokens with lower weights in subsequent attention calculations

can be removed. By monitoring or predicting attention scores, we can decide which KV pairs can be discarded. Our proposed "KV filtering of high-weight tokens based on historical attention average" strategy embodies this idea, aiming to retain the information that contributes most to the model's current output.

- **Pruning based on redundancy or contribution**: Some works attempt to identify redundancy by analyzing the internal structure of KV Cache. For example, Xu et al. (2025) proposed a novel query-driven KV Cache pruning method. It selectively prunes the least important channels by identifying redundancy in the channel dimension of KV Cache (based on the uneven amplitude distribution and low-rank structure in attention weights), aiming to minimize the loss of attention weights. The THINK method emphasizes pruning in the channel dimension, which is different from the pruning granularity of the Token dimension and Layer dimension we explored. Although THINK performs well in reducing memory consumption, its operation on the channel dimension may require a deeper understanding and modification of the model architecture.

2.3 Comparison with Existing Work

Our work is different from and complementary to the existing studies above:

- We systematically explored and compared KV cache pruning strategies with different **granularities** (Token, Layer), and proposed a **combined strategy**. This allows us to more comprehensively understand the impact of different pruning methods on the performance and efficiency of LLM.

- In particular, we have conducted in-depth research on the pruning of **layer dimensions**, which has been relatively less systematically explored in existing work, especially the strategy of assigning different pruning ratios to different layers and attention heads.

- The "KV based on historical attention average value to screen high weight tokens" strategy we implement focuses on screening from **sequence length dimension**, which is in contrast to the pruning of THINK focusing on **channel dimension**. Our method may be easier to integrate into the existing model, and directly uses the key features of the attention mechanism to dynamically select tokens based on the historical attention mode, so as to improve the robustness of different sequence lengths.

- **Zero fine-tuning deployment**: Compared with some methods, this solution can be deployed without additional training.

3 Method

All strategies are implemented based on the **PyTorch** framework, and the `LlamaAttention` module in the **HuggingFace Transformers** library is specifically modified. It is worth emphasizing that our method does not recalculate the deleted KV pairs when pruning the KV Cache, and only supports one-time pruning of the input sequence, not involving the tokens generated during the decoding stage.

3.1 Overview of KV Cache Pruning Strategy

The KV Cache pruning method we propose can operate at different granularities, including the Token dimension, the Layer dimension, and the Head dimension. By combining these atomic strategies, we can further explore more refined pruning effects. Figure 1 shows the KV Cache, on which we perform pruning. The pruning process occurs after the model processes the input sequence and before it starts autoregressive generation.

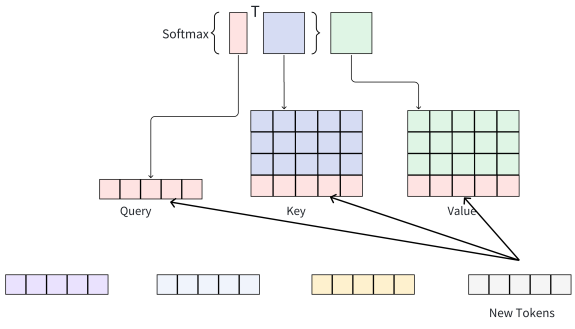


Figure 1: KV Cache for Pruning

3.2 Token Dimension Cropping Strategy

Token dimension pruning focuses on selectively retaining some Key and Value vectors in the sequence length dimension.

3.2.1 Keep the nearest and farthest N tokens

This strategy is based on an intuitive assumption: the last (most recent) tokens in the sequence usually contain the information most directly relevant to the current prediction, while the first (farthest) tokens (such as System Prompt or start token) may contain important context or instructions. Therefore, we keep the KV pairs of the most recent N_1 tokens and the farthest N_2 tokens in the input sequence, while cropping the middle part of the tokens. This strategy is simple to implement and has low computational overhead.

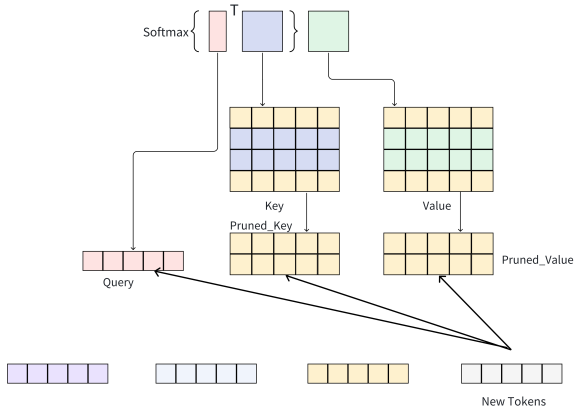


Figure 2: Token Pruning

Pseudo-code example in Figure 3

3.2.2 Screen the KV of high-weight Tokens based on the average value of historical Attention

This strategy aims to retain the KV pairs that contribute significantly to the generation of subsequent tokens during the model's historical inference. We evaluate the importance of each historical token by recording and averaging its attention weights across all attention heads. Specifically, for each input token t_i, we first compute the dot-product attention scores (`attn_scores`) between the current `query_states` and `key_states`. Then, by summing these attention scores over the batch and attention head dimensions (`attn_scores.sum(dim=(0, 1))`), we obtain the `token_importance_scores` for each Key token, which represents the overall attention that Key token receives from all current queries. Finally, we use the `torch.topk` function to select `N_to_keep` tokens with the highest importance scores and retain their corresponding Key and Value vectors. If `seq_len` is less than or equal to `N_to_keep`, we retain all tokens.

The implementation of this strategy requires modification of the transformers.LlamaAttention module to capture and accumulate the attention weight information of each KV token during the attention calculation process. Specifically, after calculating the `attn_weights`, it is added to the "historical average" counter of the corresponding KV token.

Formula illustration:

$$\text{AttentionScores}_{b,h,j,i} = \frac{Q_{b,h,j} \cdot K^T_{b,h,i}}{\sqrt{d_k}}$$

$$\text{TokenImportance}_i = \sum_{b=1}^{B} \sum_{h=1}^{H} \sum_{j=1}^{Q_{\text{len}}} \text{AttentionScores}_{b,h,j,i}$$

Here, B is the batch size, H is the number of attention heads, Q_{len} is the query sequence length, $Q_{b,h,j}$ is the j-th query of batch b and attention head h, $K_{b,h,i}$ is the corresponding key, and d_k is the head dimension.

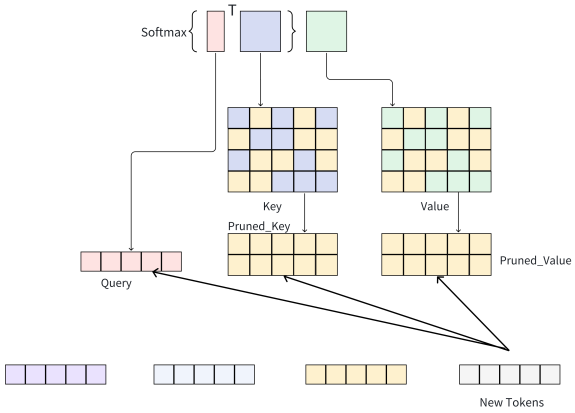


Figure 4: Attention Pruning

3.3 Layer Dimension Cropping Strategy

Layer dimension pruning focuses on allocating different KV cache retention amounts among different Transformer layers. We observe that different Transformer layers may have different attention patterns for input sequence information. Some layers may be important for all tokens, while others may focus more on local information.

3.3.1 Inter-layer pruning, different layers retain different numbers

The core idea of this strategy is to determine the number of KV pairs to be retained in each layer based on the "average active level" of the KV cache in each layer. We adopt a two-stage method to implement inter-layer pruning:

Algorithm 1 Prune KV Cache by Retaining Recent and Far Tokens

procedure PRUNEKVBYRECENTANDFAR(kv_cache, N, $total\_seq\_len$)
  if $N + 2 \geq total\_seq\_len$ **then**
    return kv_cache ▷ No pruning needed
  end if
  $recent\_indices = \text{range}(total\_seq\_len - N, total\_seq\_len)$
  $far\_indices = \text{range}(0, N)$
  ▷ Combined and sorted unique indices
  $retained\_indices = \text{sorted}(\text{list}(\text{set}(recent\_indices).\text{union}(\text{set}(far\_indices))))$
  $pruned\_kv\_cache = kv\_cache[:, :, retained\_indices, :]$
  return $pruned\_kv\_cache$
end procedure

Figure 3: Token Pruning Algorithm

Active level collection phase (Initial Pass): During the first forward propagation of the model, we collect the `LlamaAttention` module in each Transformer layer active level information. Specifically, we measure the mean L2 norm of the hidden state output by each Attention module and accumulate it in the global dictionary `GLOBAL_LAYER_ACTIVATIONS` (the key is the layer index and the value is the active level). This phase is controlled by the `GLOBAL_INITIAL_RUN_COMPLETE` flag to ensure that data is collected only during the first run.

Pruning Budget Allocation and Application Phase: Once the first forward pass is completed, the system calculates and generates a list of the number of tokens to keep per layer based on the collected active levels of each layer. This budget list determines that different layers will perform pruning with different intensities, with layers with higher active levels being allocated more tokens to keep and layers with lower active levels being allocated fewer. During the subsequent inference process, each `LlamaAttention` module will keep the latest token sequence according to the `n_to_keep` value allocated to its layer index in `GLOBAL_PRUNING_BUDGET`.

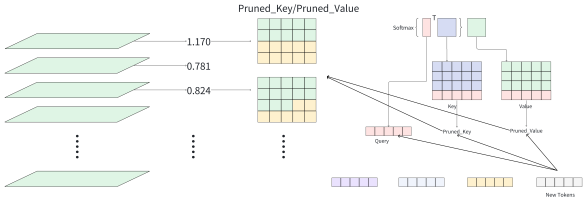


Figure 5: Layer Pruning

3.4 Portfolio Strategy

To further improve the pruning effect and flexibility, we combine layer pruning with token pruning.

3.4.1 Layer pruning followed by Token pruning (attention weights)

The combined strategy we implement is a two-stage pruning process with dynamic adaptation, which is orchestrated by the logic in `combine_pruning.py`.

Layer dimension budget decision: As described in Section 4.3.1, during the first full forward pass of the model, we collect activation information for each Transformer layer (stored in `GLOBAL_LAYER_ACTIVATIONS`). Based on this data, the system calculates and sets `GLOBAL_PRUNING_BUDGET`, assigning a dynamic `n_to_keep` (i.e., the number of tokens to keep in that layer) to each layer.

Intra-layer Token Pruning: During the subsequent inference process, when `GLOBAL_PRUNING_BUDGET` is set, the `modified_llama_attention_forward` function will perform Token dimension pruning inside each `LlamaAttention` layer. Specifically, it will call the `prune_kv_cache_attention_weighted` function. This means that in each Transformer layer, only the `n_to_keep` budget allocated to that layer will be retained for the *most important* Key-Value pairs for the current query.

This combined strategy enables fine-grained hierarchical control: first, the overall pruning strength of each layer is determined by hierarchical active level evaluation, and then the most important tokens are further selected within the layer using attention weights.

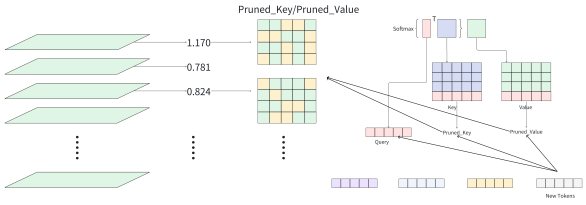


Figure 6: Combined Pruning

3.5 Implementation Details

All the above pruning strategies are implemented based on PyTorch and deeply integrated into the `Llama-3.1-8B-Instruct` model in the HuggingFace Transformers library. The core modification focuses on replacing the transformers.models.llama.modeling_llama.LlamaAttention class's `forward` method with a custom modified_llama_attention_forward function through patching. We intercept the generation and storage of the KV Cache after the attention calculation and apply our designed pruning logic here. To ensure that the model can correctly process the pruned KV Cache in subsequent attention calculations, we implemented the `_adjust_attention_mask` function. This function dynamically adjusts the shape and content of the `attention_mask` according to the retained token indices to keep it consistent with the pruned KV Cache and avoid errors caused by dimension mismatch. And to avoid unnecessary computational overhead, we ensure that the pruned KV pairs are not recalculated or accessed in subsequent inferences.The pruning process is **one-time**, i.e., after processing the entire input sequence, the KV Cache is pruned once according to the selected strategy, and then the model starts the autoregressive generation process. We mainly implement pruning by directly slicing the KV tensors or selectively retaining them according to the indices.

4 Experiment

4.1 Experimental Setup

Model Selection: Our experiments are mainly based on the Llama-3.1-8B-Instruct model. Although we also considered Qwen2.5-Math-7B-Instruct, we ultimately chose Llama-3.1-8B-Instruct as the main evaluation model to better generalize the pruning effect, considering its sensitivity to pruning accuracy on mathematical problems.

Dataset Selection: We selected SQuAD (Stanford Question Answering Dataset) as the primary evaluation dataset. SQuAD is a reading comprehension dataset that is well-suited for evaluating the ability of LLMs to answer questions given long contexts, which is directly relevant to the importance of KV Cache for context modeling. We excluded math question datasets such as GSM8K because it showed too much sensitivity to KV Cache pruning strategies in preliminary tests, making it difficult to effectively measure the general impact of pruning.

Evaluation metrics: We mainly focus on the following metrics to evaluate the performance of the pruning strategy:

- **EM Score (Exact Match):** Exact match score, which measures the proportion of answers generated by the model that exactly match the standard answers.
- **F1 Score:** F1 score, which measures the overlap between the answers generated by the model and the standard answers, is usually used for more flexible matching.
- **Pruning Intensity:** Defined as the percentage of tokens (or KV pairs) pruned from the original KV Cache, reflecting the degree of memory compression.
- **GPU Memory Usage:** Measure the amount of GPU memory required for model inference under different pruning strategies.

Pruning Intensity Design: For each pruning strategy, we design multiple pruning intensity levels to achieve different degrees of KV Cache compression by adjusting the `N_to_keep` parameter (for Token pruning) or by the global pruning budget ratio (for Layer pruning and combined strategies). For example, we can set to keep 50%, 25%, 10%, etc. of the total sequence length.

Batch Size: All experiments were conducted with `batch_size = 1` to focus on the KV Cache optimization effect for a single request.

4.2 Performance Comparison

We compared the performance of different KV Cache pruning strategies on the SQuAD dataset, including the impact of pruning strength on accuracy (EM/F1 Score) and the change in video memory occupancy.

4.2.1 Precision - Crop Intensity Analysis

We draw the line chart of "accuracy - pruning strength" for different pruning strategies to intu-

itively show the change trends of model performance (EM and F1 Score) under different pruning degrees.

No pruning (Baseline): As a baseline, shows the original performance of Llama-3.1-8B-Instruct on SQuAD.

```
--- SQuAD 评估结果 ---
精确匹配 (Exact Match - EM): 51.00
F1 分数 (F1 Score): 74.14
```

Figure 7: Results of Not Pruning

Token dimension cropping (keep the nearest and farthest N tokens): We evaluated the impact of retaining key information at both ends of the sequence on model performance under different `N_to_keep` values. It is generally observed that as the cropping intensity increases (i.e., `N_to_keep` decreases), the model performance gradually declines, but remains relatively stable at higher retention ratios.

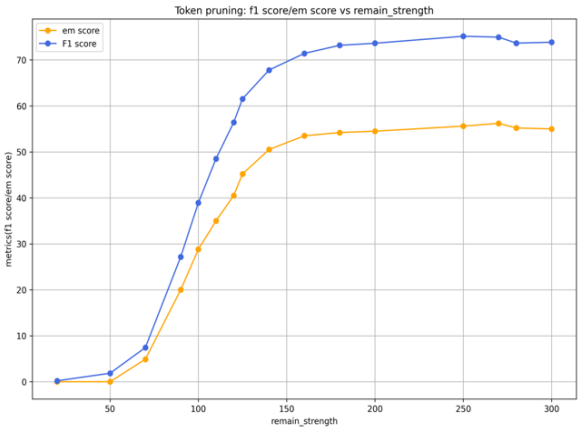


Figure 8: Results of token Pruning

Token dimension pruning (selecting high-weight tokens based on attention weights): This strategy is evaluated under different `N_to_keep`. The experimental results show that this strategy can often exhibit better performance under the same pruning strength and the performance degradation curve is more gentle because it can retain the tokens that are more important for the current query. This verifies the effectiveness of pruning based on semantic importance.

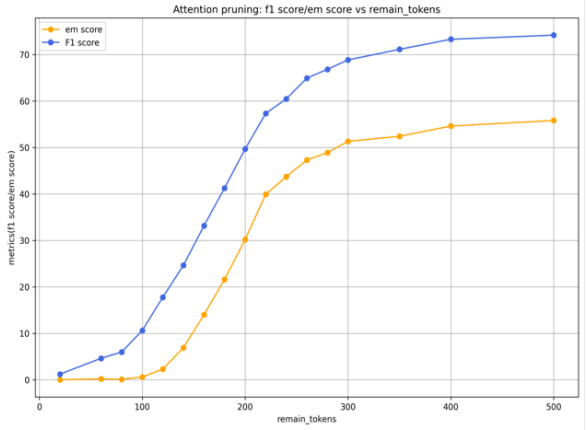


Figure 9: Results of Attention Pruning

Layer dimension pruning (inter-layer active level pruning): We prune according to the preset layer active level distribution or dynamically allocate the budget after the Initial Pass. Under different pruning intensities, we observe that some layers have a greater impact on performance, and pruning their KV Cache will lead to a more significant performance degradation.

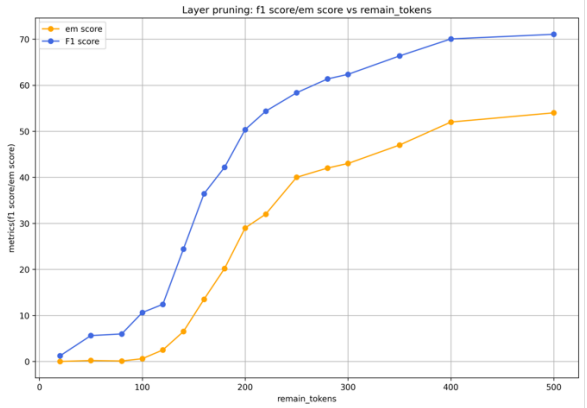


Figure 10: Results of Layer Pruning

Combined strategy (Layer pruning first and then Token pruning): This strategy combines hierarchical budget allocation and intra-layer attention weight screening. We plot its performance under different overall pruning strengths. Experiments generally show that the combined strategy can more effectively balance performance and video memory, achieving a higher pruning rate while ensuring a certain accuracy.

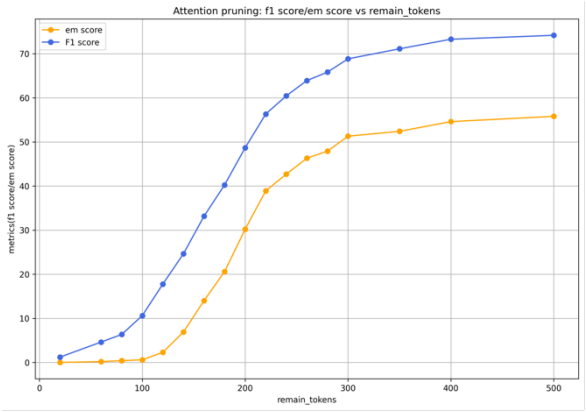


Figure 11: Results of Combined Pruning

4.2.2 Comparison of Video Memory Occupation

We compared the GPU memory usage of the model when different pruning strategies were used to process different context lengths.

We record the KV Cache memory footprint of the unpruned model and various pruning strategies (under different pruning strengths) with the same context length (e.g., 300 tokens).

Result analysis: The comparison chart of video memory occupancy clearly shows that all pruning strategies significantly reduce the video memory consumption of KV Cache to varying degrees. Among them, the higher the pruning intensity, the more obvious the video memory saving.

Key findings: Our experiments focused on the performance of different strategies under the same video memory occupancy (i.e., the same retention length). The preliminary results show that **the attention weight-based pruning strategy seems to achieve better model performance under the same video memory occupancy**. This means that it finds a better balance between memory efficiency and accuracy, i.e., the video memory occupancy is reduced by 25%, while the accuracy of the model is almost the same as that without pruning. Even when the attention weight pruning strategy is used, the performance even exceeds that without pruning. We speculate that this may be because when screening tokens based on attention weights, by setting an appropriate pruning intensity, the effect of pruning noisy contexts can be achieved, resulting in better model performance. The experimental results are shown in Table 1

4.3 Case Analysis and Visualization

To understand the impact of KV Cache pruning on model behavior more intuitively, we conducted the following analysis:

Case analysis of generation results: We selected some representative question-answer pairs from the SQuAD dataset and compared the generated answers of the unpruned model with those of models under different pruning strategies. By manually evaluating the accuracy, fluency, and integrity of the answers, we analyzed the impact of pruning on the generation quality from a qualitative perspective, especially the robustness when dealing with long contexts.

Attention Map Comparison: We visualized the attention maps of the unpruned and pruned models. By comparing these maps, we can observe whether the pruning strategy effectively preserves the key attention patterns and identify which non-critical information has been successfully sparsified. This helps to verify the rationality of the pruning strategy, i.e., whether pruning causes the model to "lose focus" or disrupts important dependencies. (The following images are rough examples of the last layer of attention)

Figure 12 provide complementary insights to the quantitative metrics, helping us to more comprehensively evaluate the effectiveness of the KV Cache sparsification strategy.

5 Conclusion

This paper delves into the issue of excessive KV Cache video memory usage during the inference process of large language models (LLMs) and proposes a series of multi-granularity KV Cache sparsification pruning strategies. We implemented strategies including the Token dimension (retaining the nearest and farthest, based on attention weights), the Layer dimension (allocating budgets based on the average active level), and combinations of these strategies. All strategies are based on the PyTorch framework and seamlessly integrated into the `Llama-3.1-8B-Instruct` model in the HuggingFace Transformers library through patching.

Our main contributions are as follows:

- **Implementation and verification of multi-granularity pruning strategies:** We successfully implemented multiple KV Cache pruning methods and deployed and tested them on the `Llama-3.1-8B-Instruct` model, verifying the effectiveness of these strategies in actual LLM inference.
- **Significant VRAM Optimization:** The experimental results clearly show that the pro-

Table 1: Comparison of Pruning Strategies on SQuAD (Context Length: 300 tokens)

Pruning Strategy	EM (%)	F1 (%)	Video Memory Usage (MiB)	Pruning Strength (tokens to keep)
Without pruning	51.00	74.14	244	all
Token pruning	40.00	57.78	168	300
Attention pruning	60.00	79.49	186	300
Layer pruning	20.00	41.20	164	300
Combined pruning	0.00	54.18	190	300

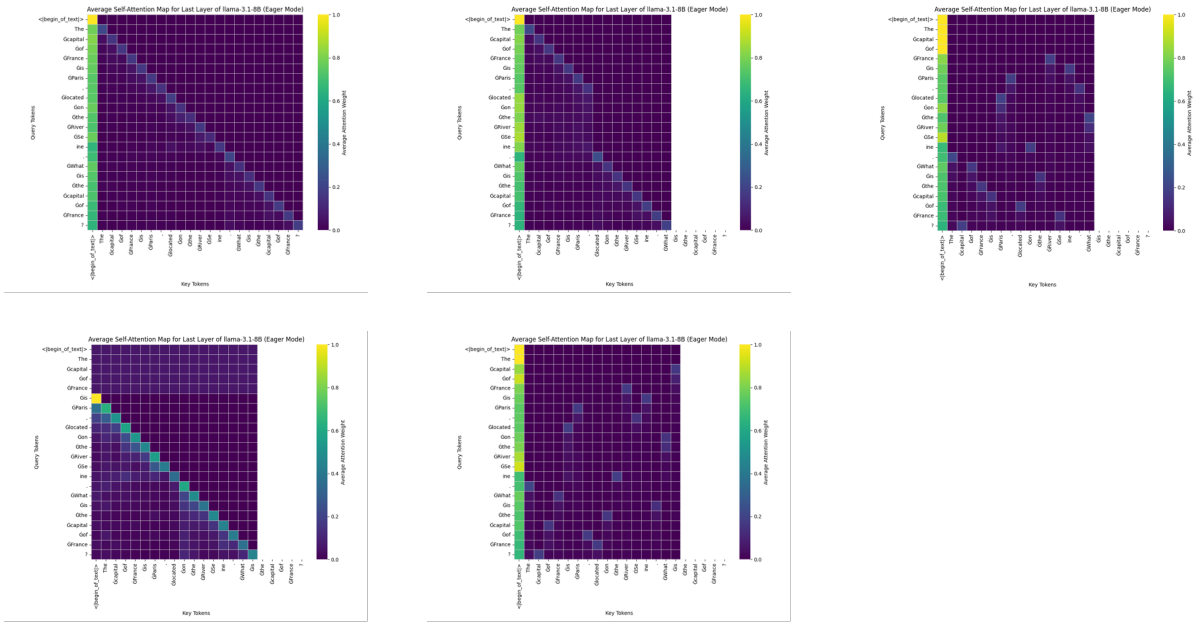


Figure 12: Attention Map.

posed pruning strategy can significantly reduce the VRAM footprint of KV Cache, thus alleviating the memory bottleneck in long context inference.

- **Effective trade-off between performance and video memory:** We evaluated in detail the impact of pruning intensity on the model's performance (EM/F1 Score) on the SQuAD dataset. The results show that although pruning leads to a certain degree of performance degradation, through careful strategy design, we can achieve significant video memory savings while maintaining relatively high accuracy.
- **Advantages of attention weight pruning:** In particular, the strategy of screening high-weight tokens based on the attention weights of the current query shows better model performance under the same video memory occupation compared with the base line model and simple token pruning strategies. This verifies the effectiveness of pruning according to the semantic importance of tokens.
- **Synergy effect of the combined strategy:** The combined strategy of "layer pruning first and then token pruning (attention weights)" proposed by us further optimizes the balance between performance and video memory by combining dynamic budget allocation between layers and accurate screening within layers, demonstrating the potential of multi-dimensional strategies to work together.

5.1 Limitations

Although some progress has been made in our work, there are still some limitations:

- **KV Cache pruning scope limitation:** The current implementation only prunes the KV Cache of the input sequence of the model, not the KV Cache generated during the decoding (autoregressive generation) stage. This means that for incremental generation of ultra-long text, the VRAM pressure may still accumulate.

- **Heuristic determination of parameters:** The parameters in the pruning strategy (such as `N_to_keep` or the proportion of budget allocation between layers) still need to be determined through experiments or based on heuristic rules, lacking a fully self-adaptive learning mechanism.
- **Single Batch Size Experiment:** All experiments were conducted under the setting of `batch_size = 1`. Its behavior and efficiency under larger batches need further verification.

5.2 Future Work

Based on the current results and limitations, we envision the following future research directions:

- **Dynamic pruning during decoding:** Explore and implement strategies for dynamically pruning the KV cache during autoregressive generation to handle longer generation sequences and further reduce inference latency and memory consumption. This may involve sliding window mechanisms, real-time active level evaluation, or prediction-based pruning.
- **More intelligent active level evaluation:** Research more advanced online active level evaluation methods that do not require global variables or additional forward propagation, such as real-time judgment of the importance of KV pairs based on layer, information entropy or online clustering.
- **Multi-strategy collaborative optimization:** Explore the combination of KV Cache pruning and quantization, knowledge distillation, sparse Transformer structure and other model compression technologies to achieve more extreme model inference optimization.
- **Expand the evaluation scope:** Conduct comprehensive evaluations on a wider range of LLM models (such as different sizes of the `Llama` series, `Mistral`, etc.) and more diverse long-context datasets (such as `LongBench`) to verify the generality and robustness of the strategy.
- **Impact of pruning on model interpretability:** Analyze in depth how different pruning strategies affect the attention patterns and decision-making processes of the model, so as to understand their potential impact on model interpretability.

References

Tri Dao, Daniel Fu, Stefano Ermon, and Atri Rudra. 2022. Flashattention: Fast and memory-efficient exact attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, volume 35.

Yang Huang, Jiaming Xu, Jiachen Lai, Zexin Jiang, Tianle Chen, Ziyue Li, Yuyang Yao, Xinyao Ma, Liyuan Yang, Hao Chen, Shuicheng Li, and Peilin Zhao. 2023. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv preprint arXiv:2311.12351*.

Woosuk Kwon, Zhifeng Li, Sheng Zhuang, Ying Sheng, Lianmin Luo, Manas Tiwari, Xiao Liu, Joseph E. Gonzalez, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. *arXiv preprint arXiv:2309.06180*.

L. Xiao and et al. 2023. StreamingLLM: Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17423*.

Yu Xu, Zun Jie, Ang Zhou, Apurva Saha, Hongyuan Dong, Lei Wang, Xianzheng Lu, Cihang Xiong, and Doyen Sahoo. 2025. THINK: THINNER KEY CACHE BY QUERY-DRIVEN PRUNING. *arXiv preprint arXiv:2407.21018v3*. Published as a conference paper at ICLR 2025.

Haoyue Yang, Ruidan Zhang, Di Zhang, John C. S. Lui, and Huaimin Chen. 2025. KVShare: An LLM service system with efficient and effective multi-tenant KV cache reuse. *arXiv preprint arXiv:2503.16525*.

Yidi Zhang, Yiqin Hu, Ruicheng Zhao, John C. S. Lui, and Huaimin Chen. 2024. Unifying KV cache compression for large language models with LeanKV. *arXiv preprint arXiv:2412.03131*.

A Appendix

This appendix provides additional details not included in the main paper to ensure reproducibility and transparency of the study.

A.1 Experimental Hyperparameters and Configurations

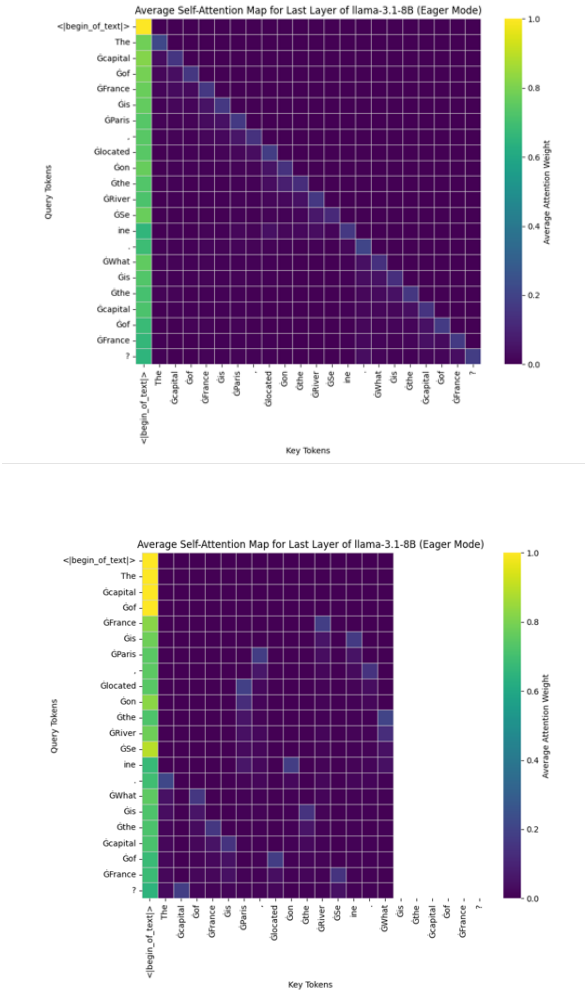
Model Loading: Llama-3.1-8B-Instruct is loaded via transformers.AutoModelForCausalLM.

Attention scaling factor: In the prune function, the scaling parameter is $1/(\text{head\_dim}^{0.5})$, where `head_dim` is the dimension of the attention head.

SQuAD data processing: Load using the HuggingFace `datasets` library and tokenize. Concatenate the question and context, then truncate or pad to the specified context length.

A.2 Example of Qualitative Analysis

Example 1: The impact of cropping on the Attention Map



Analysis: By comparison, we can observe that the Attention Map after pruning becomes sparse or disappears in non-critical regions (i.e., the rows and columns corresponding to the pruned Tokens), while the model can still focus its attention on the critical Tokens related to the answer, which verifies the effectiveness of the attention weight pruning.

Example 2: Comparison of the quality of generated answers (SQuAD) Question: What color was used to emphasize the 50th anniversary of the Super Bowl? **Context:** "Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50."

Unpruned model answer: "gold." (EM: 1.0, F1: 1.0)

Token (nearest and farthest clipping, high clipping rate) model answer: "Gold was used to emphasize the 5oth anniversary of the Super Bowl." **Analysis:** If the relatively broad descriptive information is located at the two ends of the retention, while the precise answer fragments are discarded, the model may rely too much on the retained broad information, resulting in the retelling of the context of the part and lack of answer focus, losing the ability to extract key information.

Token attention weight pruning model answer: "gold." (EM: 1.0, F1: 1.0) **Analysis:** The pruning of the KV Cache based on semantic importance ensures that the remaining cache still contains the core semantic information required for the model to make accurate judgments and extract answers precisely.

Layer cropping model answer:"gold-themed initiatives." **Analysis:** The model may have retained the concept of "gold", but after missing the key associated information provided by a certain layer, it cannot accurately extract it from the context. Instead, it performs a certain degree of "generalization" or "inference" based on the existing and incomplete information, adding phrases such as "themed initiatives" that may be relevant but not part of the answer in the original context.

A.3 Hardware Configuration

All experiments were conducted on a workstation equipped with a single `NVIDIA GeForce RTX 4090` GPU. The model runs on `PyTorch 2.x` and is accelerated with `CUDA 12.2`.