IAM: Efficient Inference through Attention Mapping between Different-scale LLMs

Yi Zhao[1,2,3], **Zuchao Li**[4,*] **and Hai Zhao**[1,2,3,*]
[1]School of Computer Science, Shanghai Jiao Tong University
[2]Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering, Shanghai Jiao Tong University
[3]Shanghai Key Laboratory of Trusted Data Circulation and Governance in Web3
[4]School of Artificial Intelligence, Wuhan University, Wuhan, China
zhao-yi@sjtu.edu.cn, zcli-charlie@whu.edu.cn, zhaohai@cs.sjtu.edu.cn

Abstract

LLMs encounter significant challenges in resource consumption nowadays, especially with long contexts. Despite extensive efforts dedicate to enhancing inference efficiency, these methods primarily exploit internal sparsity within the models, without leveraging external information for optimization. We identify the high similarity of attention matrices across different-scale LLMs, which offers a novel perspective for optimization. We first conduct a comprehensive analysis of how to measure similarity, how to select mapping Layers and whether mapping is consistency. Based on these insights, we introduce the IAM framework, which achieves dual benefits of accelerated attention computation and reduced KV cache usage by performing attention mapping between small and large LLMs. Our experimental results demonstrate that IAM can accelerate prefill by 15% and reduce KV cache usage by 22.1% without appreciably sacrificing performance. Experiments on different series of models show the generalizability of IAM. Importantly, it is also orthogonal to many existing KV cache optimization methods, making it a versatile addition to the current toolkit for enhancing LLM efficiency. Our code is available at https://github.com/QQQ-yi/IAM

1 Introduction

Large language models (LLMs) like GPT4 (OpenAI, 2024a) have emerged with remarkable natural language understanding capabilities and broad prospects in application. While subsequent advancements such as Chain-of-Thought (CoT) (Wei et al., 2022; Yao et al., 2024) have revitalized the landscape of applications, they also introduce significant computation and memory consumption due to exceedingly long contexts. Recent developments in reasoning models, exemplified by ChatGPT-o1 (OpenAI, 2024b) and DeepSeek-R1 (DeepSeek-AI, 2025), have exacerbated this issue because of their extensive internal reasoning processes.

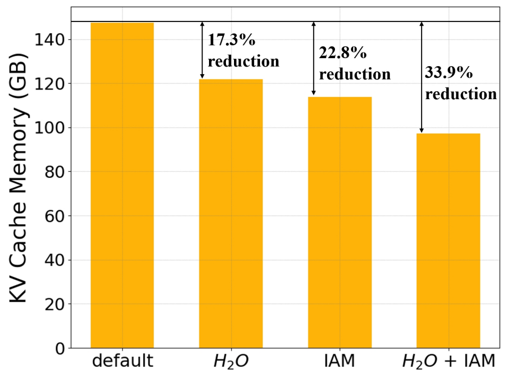

Figure 1: IAM is orthogonal to the existing KV cache eviction methods and can further reduce the KV cache usage in long context scenarios.

To address the aforementioned challenges and achieve efficient inference, various methodologies have been proposed, including optimizing model architectures (Sun et al., 2024; Gloeckle et al., 2024), prompt compression (Jiang et al., 2023; Pan et al., 2024), and KV cache optimization (Zhang et al., 2023; Yang et al., 2024b; Hooper et al., 2024), among others. A specific approach within KV cache optimization, known as KV cache eviction, achieves efficiency improvements by exploiting characteristics and sparsity within attention mechanisms. Nevertheless, these methodologies predominantly concentrate on leveraging intrinsic sparsity of LLM itself, without considering external information to facilitate better optimization.

Previous study (Chen et al., 2021) has indicated a significant similarity in attention patterns between small and large models within the BERT architecture. In Appendix B, we further substantiate that this similarity is also present in LLMs. Building

[*] Corresponding author. This research was supported by the Joint Research Project of Yangtze River Delta Science and Technology Innovation Community (No. 2022CSJGG1400), the National Natural Science Foundation of China (No. 62306216) and Xiaomi Open-Competition Research Program.

upon this characteristic, this paper introduces an efficient inference technique through attention mapping between small language model (SLM) and the larger one. It is important to note that our proposed method achieves dual benefits of accelerated attention computation and reduced KV cache comsumption, and is orthogonal to most existing KV cache optimization methods. As depicted in Figure 1, the utilization of H_2O (Zhang et al., 2023) facilitates KV cache compression at the token level, whereas our method can achieve further compression at the model layer level.

In this study, we investigate the methodology to achieve attention mapping between SLM and LLM while maintaining the performance of the original LLM. Initially, we evaluate the impact of various similarity metrics applied to attention matrices on language modeling efficacy. Following this, we explore how mapping at different layers of the LLM influences its language-understanding capabilities, thereby identifying the most appropriate layers for mapping. Finally, we demonstrate that the established mapping relation remains consistent throughout the inference process, enabling it to be constructed during the prefill stage and subsequently utilized in subsequent decode stage. This method of establishing mapping also facilitates dynamic adaptation to evolving contexts.

Based on these experiments and observations, we introduce the efficient **I**nference through **A**ttention **M**apping (**IAM**) framework. This framework effectively captures the similarities of attention patterns between SLM and LLM to dynamically establish mapping across varying contexts. Consequently, LLM can perform efficient inference without calculating portions of attention matrices, thereby achieving dual benefits of reduced GPU memory usage for KV cache and decreased computational requirements in attention mechanisms. We first comprehensively evaluate the performance preservation of IAM across four kinds of scenarios. Experimental results indicate that with a 30% mapping ratio, the model maintains performance close to lossless, while at a 50% mapping ratio, it also retains high capability levels. Efficiency evaluations across different inference scenarios demonstrate that IAM achieves an average reduction of 22.1% in KV cache usage and an average acceleration of 11% in inference speed. Our other experimental results indicate that the IAM is generalizable on other series of LLMs and compatible with existing KV cache optimization methods.

2 Related Work

Due to the widespread adoption of technologies such as RAG and the recent emergence of reasoning models like DeepSeek-R1, the demand for handling long contexts has significantly increased. The self-attention mechanism necessitates computing attention between the current token and every preceding token, leading to the common practice of storing previous tokens' KV states (KV cache) to avoid recomputation. However, this approach has become a primary bottleneck for managing long contexts.

One category of methods focuses on efficiently storing and transmitting large amounts of KV cache with constrained hardware. For instance, tensor parallelism (Shoeybi et al., 2020) distributes attention heads and pipeline parallelism (Huang et al., 2019) distributes attention layers across multiple GPUs, enabling horizontal scaling of KV cache storage capacity by adding more GPUs. When GPU HBM is insufficient, some techniques (Sheng et al., 2023) concentrate on efficiently offloading the KV cache to CPU memory. Mooncake (Qin et al., 2024) advances this concept further by proposing a multi-level caching strategy centered around the KV cache. At the CUDA optimization level, FlashAttention (Dao, 2023) reduces the number of read/write operations between GPU HBM and GPU cache, while PagedAttention (Kwon et al., 2023) employs virtual memory management techniques to minimize memory fragmentation in GPU HBM. These approaches primarily aim to optimize hardware capabilities and KV cache requirements from a system perspective, without addressing KV cache reduction from an algorithmic perspective.

Another category of methods focuses on leveraging the inherent similarities and sparsity within the attention mechanism. Techniques such as KVQuant (Hooper et al., 2024) exploit redundancies in numerical representations to propose quantization of the KV cache, thereby reducing storage requirements and enhancing load speeds. StreamingLLM (Xiao et al., 2024) achieves closely unlimited input by reserving both the initial and the most recent KV cache of tokens. H_2O (Zhang et al., 2023) observed that the accumulated attention scores of all tokens follow a power-law distribution, indicating that only a small subset of tokens is highly significant in the generation. Scissorhands (Liu et al., 2023) revealed the persistence of importance, indicating that tokens identified as important in initial remain significant throughout subsequent stages of inference. PyramidInfer (Yang et al., 2024b) further explores the distinct attention characteristics across different layers within LLM, and identified that deeper layers exhibit greater redundancy. These findings help design effective KV cache eviction approaches. Meanwhile, KVMerger (Wang et al., 2024) leverages the high similarity of KV states observed across different datasets to design a Gaussian kernel weighted merging algorithm for merging KV Cache, thereby minimizing the loss introduced by dropping methods. However, these approaches primarily exploit the inherent similarities and sparsity within the model's attention mechanism and do not incorporate external information to enhance KV cache optimization, as IAM does. In addition, IAM is orthogonal to most of the aforementioned methods, due to its innovative approach of utilizing entire attention matrices of SLM and avoiding computations of attention mechanism of mapped layers in LLM.

There are also some works about speculative decoding (Xia et al., 2023; Cai et al., 2024), which employs a SLM as a draft model, followed by validation from a LLM to determine whether to adopt its predictions. Similar to our method, this approach aims to accelerate inference through the collaboration of small and large models. However, the core mechanisms differ: speculative decoding focuses on aligning the next-word prediction probability distributions between the SLM and LLM, while IAM relies on the high similarity of attention matrices between the SLM and LLM. Additionally, the LLM in speculative decoding can also benefit from the IAM method to enhance performance, achieving a synergistic effect.

3 Methodology

In this section, we begin by conducting a series of experiments aimed at providing valuable insights into attention mapping. Building on these findings, we then introduce the IAM framework.

3.1 How to Measure Similarity

A proper metric of similarity is crucial in attention mapping. It helps to capture the significant pattern within the attention matrix, thus minimizing the bias introduced by mapping in the forward process of LLM. For vectors $\mathbf{x}$ and $\mathbf{y}$, formed by flattening the lower triangular part of attention matrices of SLM and LLM, we consider the following similarity measure:

Cosine Similarity. The cosine similarity is defined as:

$$cos < \mathbf{x}, \mathbf{y} >= \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{1}$$

Minkowski Distance. The Minkowski Distance is defined as:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{2}$$

where x_i and y_i is the element in vector $\mathbf{x}$ and $\mathbf{y}$, and p is the hyperparameter.

Pearson correlation. The Pearson correlation is a common measure of linear correlation between two variables. It is defined as:

$$r_{\mathbf{xy}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} \tag{3}$$

where x_i and y_i is the element of vector $\mathbf{x}$ and $\mathbf{y}$, and $\bar{x}$ and $\bar{y}$ represents mean of $\mathbf{x}$ and $\mathbf{y}$.

We utilize Qwen2-72B as the target LLM and consider two SLMs with different scales, Qwen2-0.5B and Qwen2-7B, as sources of attention matrices for mapping. We measure the log perplexity of the LLM on WikiText-v2 (Merity et al., 2016) after performing attention mapping using different metrics of similarity. The experimental results are presented in Table 1. It is noted that attention mapping is applied to the layers in the latter half of the LLM.

From the experimental results, it can be observed that cosine similarity achieves the best performance for both mappings from Qwen2-0.5B to Qwen2-72B and from Qwen2-7B to Qwen2-72B. Minkowski Distance with p = 1 also performs nearly as well as the best metric. Pearson correlation is proven unsuitable as the similarity metric due to its significantly higher perplexity. We also explore compensating the differences of the L2 norm after selecting the mapping matrix based on cosine similarity, which is named "Cosine with Norm" in Table 1. However, the experimental results indicate that this approach does not consistently and significantly enhance the performance of cosine similarity. Moreover, introducing the variable of L2 norm adds complexity to the mapping in the inference process and is unfavorable for the generalization across different contexts. Therefore, we finally choose cosine similarity as the similarity metric.

Mapping	Metric	Perplexity (log)
Qwen2-0.5B to Qwen2-72B	Cosine	2.577
	Pearson	4.976
	Minkowski (p = 1)	2.589
	Minkowski (p = 2)	2.878
	Cosine with Norm	**2.562**
Qwen2-7B to Qwen2-72B	Cosine	**2.414**
	Pearson	5.376
	Minkowski (p = 1)	2.427
	Minkowski (p = 2)	2.514
	Cosine with Norm	2.421
Qwen2-72B (Original)		2.136

Table 1: The log perplexity values after conducting attention mapping using different similarity metrics. The original log perplexity is also provided in the bottom for comparison.

3.2 Which Layers to Map

In this section, we explore the methodology for selecting suitable layers to map, with the objective of retaining model performance. For our purposes, we utilize Qwen2-72B as the target LLM for mapping which comprises 80 transformer layers. We partitioned these layers into 10 sub-blocks (8 layers each) and make them mapped sequentially, followed by an evaluation of its performance using the MMLU benchmark.

The experimental results are shown in Figure 2. The experimental results indicate the presence of two optimal mapping regions within the Qwen2 model architecture: one situated in the final two sub-blocks (layers 64 - 80) and another located in No.2 to No.4 sub-blocks (layers 16 - 40). These regions maintain performance levels comparable to those of the original model after attention mapping. Conversely, mapping from the beginning (layers 0 - 16) is entirely impractical due to the significant performance degradation it induces.

We also examine the effectiveness of the mapping strategy on other tasks, using linear correlation to verify whether a well-performing strategy derived from one task (e.g., MMLU) can generalize to others. The experimental results are shown in Table 3. Noting that lower log perplexity indicates better performance, we take the reciprocal of this metric to align its trend with that of other tasks for consistency. As can be seen, the results of mapping different sub-blocks on the MMLU task show strong similarity with those on other tasks, as evidenced by the high Pearson correlation coefficients between them.

Based on these observations, IAM adopts the following mapping strategy for Qwen2-72B: according to the user-specified mapping ratio, mappings are first performed sequentially from the last layer of the model backward to layer 64. If the specified mapping ratio is still not achieved, the mapping continues from layer 16 onward, proceeding backwards until the requirement is met. This approach ensures a balance between maintaining model performance and achieving the desired mapping efficiency, leveraging the identified optimal regions for layer mapping.

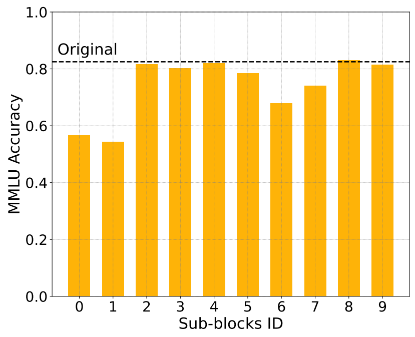

Figure 2: The performance on MMLU benchmark after mapping different attention layers of Qwen2-72B.

3.3 Consistency of Mapping

IAM establishes a mapping relationship between the SLM and the LLM based on similarity during the prefill stage, which is then utilized in the subsequent decode stage. Therefore, an essential guarantee is that the mappings established during the prefill stage remain consistent, which means mapping relationship should not significantly change as the inference progresses. To evaluate the consistency, we observe the proportion of times that the mapping does not change as consistency rate during the autoregressive generation. We also use the WikiText-v2 datasets as the prompt and set the max output tokens as 500.

The experimental results are shown in Figure 3, where we averaged the consistency rate within layers. It can be seen that a high level of consistency rate is maintained across layers. This indicates that the mapping established during the prefill stage remains stable and reliable throughout the decode stage, ensuring relatively small imprecision of mapping during the dynamic generation.

Benchmark	Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8	Block 9	Pearson
MMLU	0.567	0.544	0.816	0.802	0.819	0.784	0.678	0.741	0.83	0.815	-
1/Perplexity (log)	0.218	0.288	0.326	0.350	0.364	0.374	0.383	0.388	0.395	0.403	0.741
HotpotQA (Norm.)	0.386	0.415	0.924	1.278	1.311	0.560	0.779	1.108	0.907	1.032	0.763
GovReport (Norm.)	0.160	0.259	0.797	0.996	1.007	1.090	0.639	0.806	1.014	1.047	0.950

Table 2: The performance on different benchmarks after mapping different attention layers of Qwen2-72B. The high Pearson correlation coefficients indicate the consistency of the impact on different tasks with the mapping strategy.

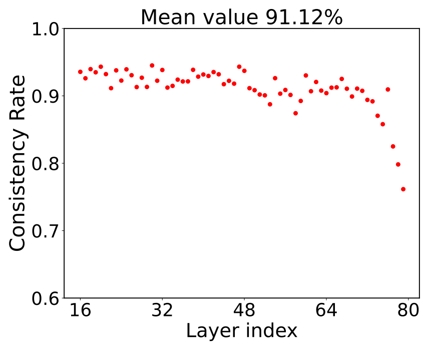

Figure 3: The average consistency rate from the 16th layer onward. A higher consistency rate indicates that the mapping established in prefill stage remains more stable and undergoes fewer changes during decoding.

3.4 IAM Framework

Based on the preceding experiments and observations, we present the framework of IAM in this section, as shown in Figure 4. IAM initially establishes mappings during the prefill stage. Given a LLM M and a SLM M_s, the prompt is first passed through both models to obtain the attention matrices respectively. Specifically, the attention matrices of M are denoted as $A_i \in \mathbb{R}^{N\times N}$, $0 < i < L \cdot D$, and those in M_s are denoted as $A'_j \in \mathbb{R}^{N\times N}$, $0 < j < l \cdot d$. L and l represent the number of layers in the M and M_s. D and d denote the number of attention heads per layer in each model. N represents the number of tokens in the prompt. Due to differences in model architectures, we have $l \cdot d < L \cdot D$. And then, pairwise similarity between the attention matrices from M and M_s is computed by:

$$S(A_i, A'_j) = \frac{\mathrm{Tr}(A_i^T A'_j)}{\|A_i\|_F \|A'_j\|_F} \tag{4}$$

$\mathrm{Tr}(\cdot)$ denotes the trace of the matrix and $\|\cdot\|_F$ denotes the Frobenius norm. Finally, the mapping function is obtained by:

$$f(i) = \arg\max_j S(A_i, A'_j) \tag{5}$$

In the decode stage, IAM utilizes the established mapping $f(i)$ to perform the attention mapping, which can be represented as:

$$A_i \leftarrow A'_{f(i)} \tag{6}$$

Finally, the forward process of the model based on IAM can be represented as:

$$\mathbf{Y} = M(\mathbf{X}; \theta^M, A_i \leftarrow A'_{f(i)}) \tag{7}$$

where $\mathbf{X}$ and $\mathbf{Y}$ denotes the input and output respectively.

After establishing the IAM framework, we perform instruction tuning to mitigate the performance degradation caused by differences in the attention matrices. Specifically, we utilized the Alpaca dataset (Taori et al., 2023) for this purpose. More details about training can be found in Appendix A. The optimization objective for M_s can be formulated as:

$$\min_{\theta^{M_s}} \mathbb{E}_{(\mathbf{X},\mathbf{T})\sim\mathcal{D}} \left[\mathcal{L}(M(\mathbf{X}; \theta^{M_s}, A_i \leftarrow A'_{f(i)}), \mathbf{T})\right]$$

It is also noted that to prevent instability in the mapping established during the prefill stage due to overly short prompt, we employ a delayed establishment mechanism. This means that mapping is only initiated and performed once the number of preceding tokens N exceeds a specified threshold. Similarly, for scenarios involving long contexts, based on observations of consistency in Section 3.3, we truncate the sequences when calculating similarity to avoid inaccuracies that arise from high-dimensional data. The detailed procedure of IAM can be referred to Algorithm 1.

4 Experiments

4.1 Experiments Setup

Datasets We comprehensively evaluate the utility preservation of IAM from four kinds of scenarios: 1) Language modeling: we measure the perplexity on WikiText-v2 (Merity et al., 2016). 2) Language understanding: We evaluate performance on

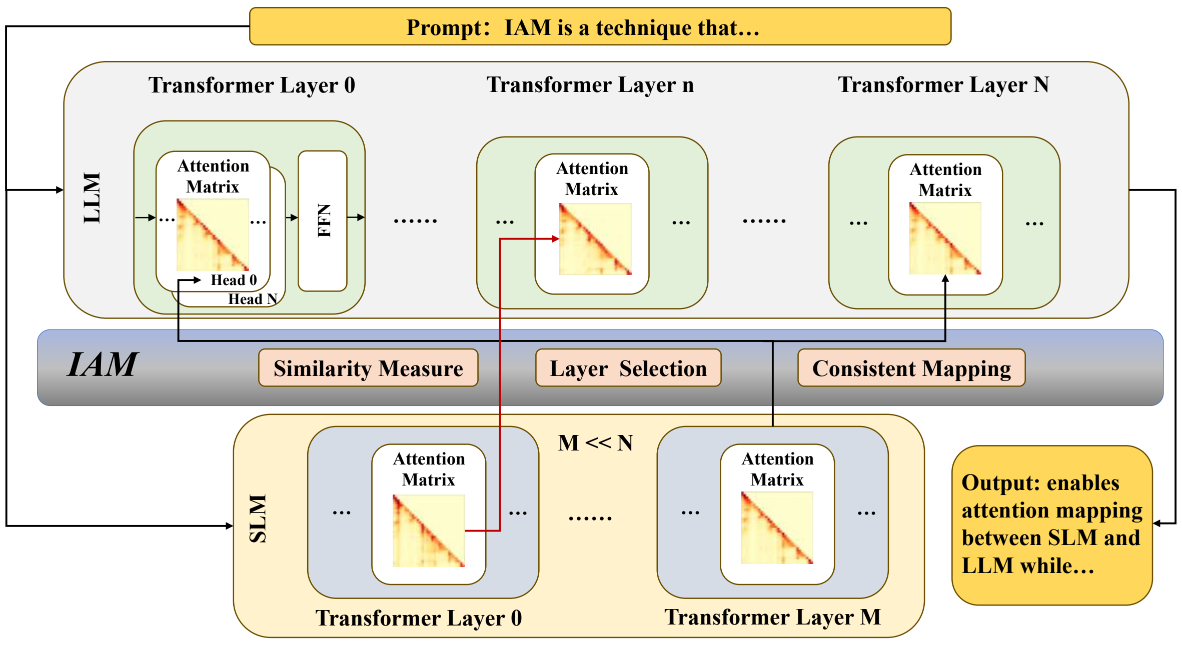

Figure 4: Framework of IAM.

the MMLU (Hendrycks et al., 2021). 3) QA task: we assess QA performance using the HotpotQA dataset (Yang et al., 2018) and adopt F1 score as evaluation metric. 4) Long context: we utilize Gov-Report benchmark (Huang et al., 2021) to test the long text summarization capacity.

Implementation Details Our experiments are conducted on two types of LLMs. The main results are obtained using the models of the Qwen2 series[1]. Specifically, we use Qwen2-72B as target LLM and two SLMs with different scales: Qwen2-0.5B and Qwen2-7B. We also test the LLaMA3 series models[2] to study the IAM's adaptation to different series LLMs in section 4.4. For these experiments, the SLM is LLaMA 3.2-1B, and the LLM is LLaMA 3.1-70B.

All experiments are performed on 8 NVIDIA A100 (80GB) GPUs. We use greedy decoding to ensure the stability of experimental results. For generative tasks, we limit the maximum output tokens to 512 and set the repetition penalty as 1.2. We also set the establishing threshold τ_e and the truncation threshold τ_t in IAM algorithm as 20 and 100 respectively. The other experimental environment includes the following configurations: CUDA version 12.0, PyTorch version 2.4.0, and HuggingFace's Transformers[3] with version 4.45.1.

4.2 Benchmark Results

In Figure 5, we evaluate the performance of IAM in four kinds of scenarios with mapping ratio varying from 0 to 50%. It can be seen that IAM maintains high language understanding and generation quality with much less GPU memory consumption. Specifically, at a mapping ratio of 30%, IAM achieves almost the same performance as the original model. At a mapping ratio of 50%, it still retains high capability levels.

It is also important to note that using Qwen2-7B as the SLM yields better performance. We conjecture that this is because using the bigger model with more source attention matrices can improve maximum similarity between mapping layers, which will reduce the loss of mapping. This implies that there is a trade-off between efficiency and model performance.

4.3 Efficiency Analysis

To evaluate the efficiency of IAM, we consider two common scenarios: 1) Multi-user Concurrency: In this scenario, the context length is set to a prefill length of 512 tokens and a maximum generation length of 512 tokens, with a batch size of 64. 2) Long Context: In this scenario, the context length

[1]https://github.com/QwenLM/Qwen
[2]https://github.com/meta-llama/llama3
[3]https://github.com/huggingface/transformers

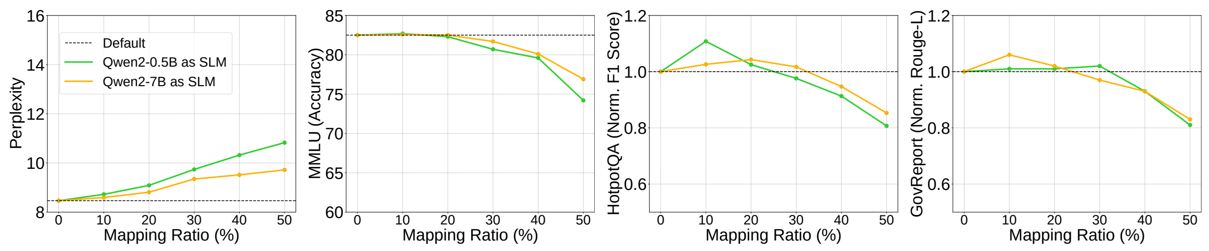

Figure 5: Benchmark results of IAM with mapping ratio varying from 0 to 50%. Default represents using the original LLM without mapping.

Algorithm 1 IAM

Input: A target large language model M; A small language model M_s; Prompt $\boldsymbol{x}$; Establishing threshold τ_e; Truncation threshold τ_t; Mapping ratio R; Max output lenth L_{max}.

```
1:  Calculate token lenth of prompt as $len(\boldsymbol{x})$
2:  if $len(\boldsymbol{x}) > \tau_t$ then
3:      Truncate lenth of prompt to $\tau_t$
4:  else if $len(\boldsymbol{x}) < \tau_e$ then
5:      Autoregressive generation $\boldsymbol{y} = \{y_i\}_{i=1}^{k}$ by
        $M$ until $len(\boldsymbol{x}) + k >= \tau_e$
6:  end if
7:  Forward process by $M(\boldsymbol{x})$ and $M_s(\boldsymbol{x})$ and get
    attention matrices respectively
8:  Calculate pairwise similarity via Eq.(4)
9:  Select mapping layers according to $R$
10: Establish mapping via Eq.(5)
11: while $y_i \neq eos$ and $L < L_{max}$ do
12:     Replace attention matrix of $M$ via Eq.(6)
13:     Autoregressive generation $\boldsymbol{y} = \{y_i\}_{i=k}^{L}$ by
        $M(\boldsymbol{x}, \boldsymbol{y})$
14: end while
```

Output: Output token sequences $\boldsymbol{y} = \{y_i\}_{i=1}^{L}$

is set to a prefill length of 8192 tokens and a maximum generation length of 512 tokens, with a batch size of 8. In each scenario, we analyze both memory efficiency and compute efficiency, and thus derive a detailed view of the impact of IAM on end-to-end performance. The LLM used in this part of the experiment is Qwen2-72B while the SLM is Qwen2-0.5B, and the mapping ratio is set to 0.5.

Memory Efficiency With IAM, the mapped layers in the LLM do not need to store the K cache because they do not compute attention matrices. This results in a reduction of memory usage. Although the introduction of SLM with its parameters and kv cache also incurs additional overhead. It is acceptable since the SLM's memory is significantly smaller than that of the LLM. Referring to Table 3, even when accounting for the KV cache of the SLM, IAM achieves a 21.7% reduction in KV cache usage in the multi-user concurrency scenario and a 22.5% reduction in the long context scenario, which indicates the memory efficiency of IAM.

Compute Efficiency Similarly, in IAM, the mapped layers of LLM do not need to perform Q projection, K projection, and the multiplication of the QK matrix with softmax. These computations are particularly intensive during the prefill stage and often become a performance bottleneck. By using IAM, we can significantly reduce this computational load. Although introducing the SLM adds some computational overhead, this additional workload is significantly lower compared to the computations avoided in the LLM. As a result, the overall system still sees substantial benefits: In the multi-user concurrency scenario, the reduction in computational load leads to a 12% decrease in TTFT. In the long context scenario, where the computational complexity of the QK matrix multiplication increases quadratically with sequence length, IAM provides even greater computational savings, resulting in a more significant 17% reduction in TTFT.

End-to-end Efficiency The resulting memory and compute savings contribute to improved overall system efficiency. Benefit from the substantial reduction in K cache usage, the LLM experiences notable improvements in TPOT during the decode stage, with decreases of 11% and 10% in the multi-user concurrency and long context scenarios, respectively. Additionally, the significant reduction in computational load leads to improved TTFT during the prefill stage, with decreases of 12% and 17% in the same scenarios. These enhancements translate into improved end-to-end throughput performance. Compared to the default method, IAM

Table 3: Evaluation on efficiency of IAM. **Bsz**: batch size. **Lenth**: prefill length + decode lenth. **KV Mem.**: GPU memory usage (GB) of the KV cache. **TPOT**: average time (s) per output token in decode stage. **TTFT**: time (s) to first token. **Thr.**: End-to-end throughput (token/s)

Method	Bsz	Lenth	KV Mem.	TPOT	TTFT	Thr.
Default	64	512+512	158.7 (100%)	0.196 (1.0x)	2.67 (1.0x)	636 (1.0x)
IAM			**124.2 (78.3%)**	0.176 (1.11x)	2.39 (1.12x)	**708 (1.11x)**
Default	4	8192+512	187.4 (100%)	0.216 (1.0x)	6.72 (1.0x)	297 (1.0x)
IAM			**143.1 (77.5%)**	0.197 (1.10x)	5.74 (1.17x)	**326 (1.10x)**

achieves throughput improvements of 11% and 10% in the multi-user concurrency and long context scenarios, making it both practical and beneficial for various deployment scenarios.

4.4 Different Series Models

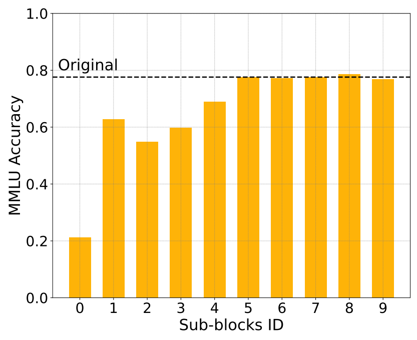

Figure 6: The performance on MMLU benchmark after mapping different attention layers of LLaMA 3.1-70B.

To evaluate the generalizability of the IAM, we conduct experiments on different series of models. In this experiment, the LLM is LLaMA 3.1-70B and the SLM is LLaMA 3.2-1B. The procedure of IAM for different series of models is largely followed the procedures detailed in the methodology section, with the primary modification being the re-identification of suitable layers within LLaMA 3.1-70B for mapping. Following the same approach described in Section 3.2, we test the suitable layers within LLaMA 3.1-70B and show results in Figure 8. An interesting observation is that, unlike the Qwen2 series models which exhibit two optimal mapping regions, the LLaMA series models only have a single optimal mapping region located at the end of the model. Consequently, we adjusted our mapping strategy for the LLaMA series models: as the mapping ratio increases, mappings are sequentially established from the last layer backward toward earlier layers.

After selecting the appropriate layers for mapping in the LLaMA series models, we also evaluated their performance on the MMLU benchmark under various mapping ratios. The experimental results are shown in Figure 7. Even better than the performance with the Qwen2 series, IAM demonstrates excellent language comprehension capabilities on the LLaMA series models.

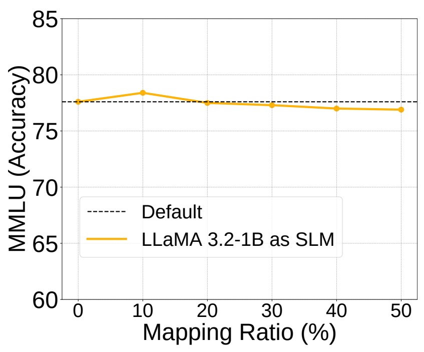

Figure 7: The MMLU benchmark results of IAM on LLaMA series models.

4.5 Compatible with KV cache compression

IAM is orthogonal to most existing KV cache compression methods due to its innovative approach of utilizing entire attention matrices of SLM for mapping, whereas existing methods mainly focus on token-level attention importance. To illustrate this, we take H_2O (Zhang et al., 2023) as an example. In Figure 1, we demonstrate that, after the KV cache optimization of H_2O, IAM can further reduce KV cache consumption. We also evaluate the impact of using H_2O on IAM's performance in terms of perplexity on the WikiText-v2 dataset, as shown in Figure 3. The KV cache budget of H_2O is set to 80%. Experimental results indicate that IAM is compatible with KV cache compression methods like H_2O without compromising the model's ability of language modeling.

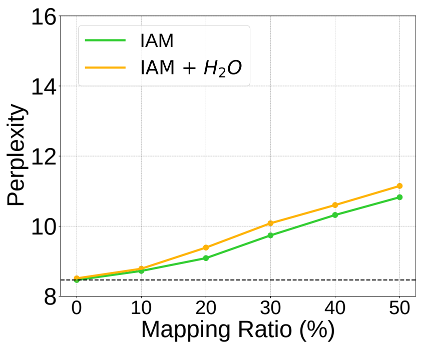

Figure 8: The perplexity of using IAM in combination with KV cache compression method H_2O.

5 Conclusion

We address the challenges faced by LLMs in serving long context scenarios by introducing IAM. This approach effectively reduces attention computation and KV cache usage by performing attention mapping between different-sized models with same series, with minimal impact on model performance. We also demonstrate that IAM is effective across different series of models. Furthermore, it is compatible with many existing KV cache optimization methods, making it a highly promising solution for deploying LLMs in resource-constrained environments.

Limitations

There are also some limitations in our approach: the current analysis and implementation of IAM are conducted at the granularity of model layers. If the granularity is further refined to the attention head, it could potentially yield additional improvements in model performance and resource savings. Additionally, IAM is not compatible with high-efficiency attention methods (e.g., Flash Attention) as it requires calculating attention scores. Besides that, IAM can not achieve total lossless for model performance.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2021. bert2bert: Towards reusable pretrained language models. *Preprint*, arXiv:2110.07143.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *Preprint*, arXiv:2307.08691.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better faster large language models via multi-token prediction. *Preprint*, arXiv:2404.19737.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Preprint*, arXiv:2401.18079.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism*. Curran Associates Inc., Red Hook, NY, USA.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time.

In *Advances in Neural Information Processing Systems*, volume 36, pages 52342–52364. Curran Associates, Inc.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

OpenAI. 2024a. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI. 2024b. Introducing openai o1-preview. `https://openai.com/index/introducing-openai-o1-preview/`.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 963–981, Bangkok, Thailand. Association for Computational Linguistics.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *Preprint*, arXiv:2407.00079.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. *Preprint*, arXiv:2303.06865.

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.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *Preprint*, arXiv:2407.08454.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. *Preprint*, arXiv:2203.16487.

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

An Yang, Baosong Yang, Binyuan Hui, et al. 2024a. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Preprint*, arXiv:1809.09600.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710. Curran Associates, Inc.

A Training Setting

The model is trained using 8 NVIDIA A100 80GB GPUs. During training, only the parameters of the small model are updated through gradient descent. We set the batch size to 128 and train for a maximum of five epochs. The learning rate is set to 2e-5, with a weight decay of 0.01 applied during training. These training parameters remain consistent for both the Qwen series and Llama series models. We observe that the training loss stops decreasing significantly after two epochs; therefore, we adopt the small model trained after two epochs for subsequent evaluation experiments.

We compared the perplexity of Qwen2-72B using IAM on the WikiText-v2 dataset before and after instruction tuning in Table 4. The results show that the perplexity decreases significantly after fine-tuning. Moreover, as the mapping ratio increases, the decrease becomes more pronounced. This indicates that appropriate fine-tuning at higher Mapping Ratios can help mitigate the performance degradation caused by IAM.

Mapping Ratio	10%	20%	30%	40%	50%
w.o. tuning	8.47	8.72	9.09	9.74	10.32
w. tuning	8.53	8.84	9.18	10.34	11.99

Table 4: The ablation study on the impact of fine-tuning on IAM method.

B Extended Analysis of Attention Similarity

What is the degree of attention similarity between different-scale LLMs? A fundamental premise of this paper is the high similarity of attention matrices across different-sized models within same series. We first demonstrate the visualization of average attention matrices across all layers and all heads under a given context in WikiText-v2 using four different-size models from the Qwen2 series, as shown in Figure 9. It can be observed that these matrices exhibit strong similarities in their attention patterns.

We also provide a quantitative analysis of this similarity. Considering the WikiText-v2 dataset, we use the SLM of Qwen2-7B and the LLM of Qwen2-72B to compute the average cosine similarity of all pairwise most similar matrices. The result yield a average cosine similarity of 0.954.

Why is such high similarity exhibited? The attention similarities between different-scale LLMs within same series are from our empirical observations. We speculate that it may be caused by the following factors: 1) They share the same model architecture, with differences only in parameters such as the number of layers and hidden dimensions. 2) They are pre-trained on the same corpus (Yang et al., 2024a). 3) Some smaller models are distilled from larger models (Abdin et al., 2024). 4) The attention patterns in larger LLM exhibit more redundancy, especially in deeper decode layers, which means these layers tend to have simpler attention patterns for mapping by SLM.

Can different series models show the same similarity? Due to differences in training strategies and model architectures, the similarity between models from different series would drop significantly. For example, we test the case where the SLM is llama3.2-1B and the LLM is Qwen2-72B. Due to the difference of tokenizers, we employ the pairwise longest common subsequence method to align tokens between the two models. In this situation, the average cosine similarity is only 0.568. More critically, discrepancies in tokenizers result in attention matrices with different dimensions, which prevents direct attention mapping using IAM method.

C Detailed Analysis of Mapping Strategy

In this section, we provide additional analysis of the mapping strategy. There is a natural consideration of mapping the layers with the highest average cosine similarity, aiming to minimize the loss introduced by mapping. To this end, we first calculate the average cosine similarity for each layer of Qwen2-72B, as shown in Figure 10. It is evident that the layers in the first half of the model, excluding the first two layers, exhibit more strong similarity.

We consider three mapping strategies: From the Front-End (mapping only the initial layers), From the Back-End (mapping only the latter layers), and the Most Similar (mapping the layers with the highest mean cosine similarity). Using the same evaluation methodology as described in section 3.1, the experimental results are illustrated in Figure 11, where the x-axis represents the percentage of layers mapped relative to the total number of layers. From experimental results, it is evident that the From the Front-End strategy is entirely unviable. The From

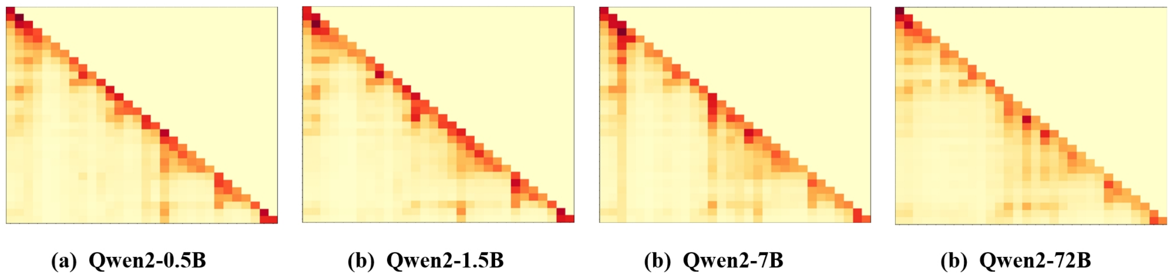

Figure 9: The average attention matrices across all layers and heads from four different-scale models of Qwen2 series.

the Back-End strategy yields the best results, even outperforming the Most Similar approach. We conjecture that this can be attributed to the cumulative error introduced by front-end mapping, which progressively accumulates through each subsequent layer. Consequently, introducing mappings at the back-end leads to fewer alterations from the original model, thereby maintaining a relatively greater capability of the LLM. This also explains why mapping the latter layers consistently yields better performance, both in the Qwen2 series models and the LLaMA3 series models.

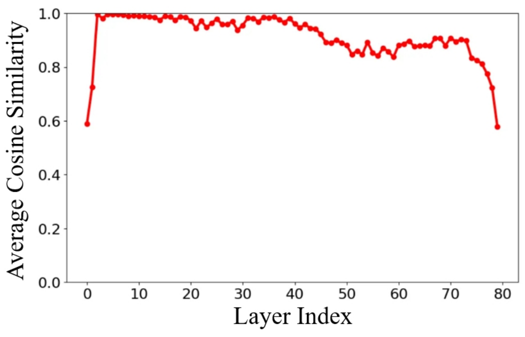

Figure 10: The average cosine similarity for each layer of Qwen2-72B by calculating the most simular attention matrix between Qwen2-7B and Qwen2-72B.

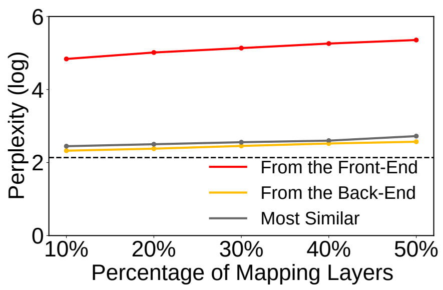

Figure 11: The log perplexity of LLM according to different mapping layer selecting strategies.

D Statistical Analysis of Mapping Relations

We also conduct a statistical analysis of the most frequent mapping relations on the WikiText-v2 dataset. Specifically, we record the mapping relations for each context. For each attention head of the LLM, we identify the mode of its mapping index. In this experiment, we used a SLM of Qwen2-0.5B and a LLM of Qwen2-72B, resulting in a mapping from 5120 to 360. The experimental results are shown in Figure 12.

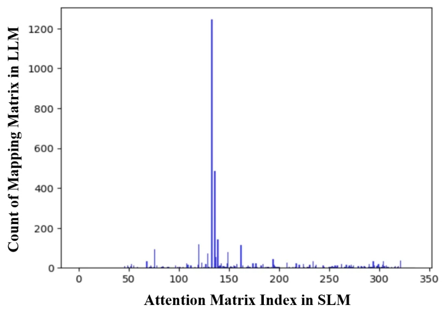

Figure 12: The statistical analysis of the most frequent mapping relations between Qwen2-0.5B and Qwen2-72B.

First, it can be observed that most matrices from the SLM are utilized in the mapping process, with the exception of the first forty matrices, which are rarely used. Additionally, the mapping relations exhibit a strong imbalance across different matrices. The most frequently used matrix is utilized more than 1200 times, significantly more than any other matrix. This finding also suggests that the attention patterns within the LLM have substantial sparsity. Such redundancy may partly explain why the LLM can extract key attention patterns from the more refined SLM, thereby maintaining performance without degradation.