MoQAE: Mixed-Precision Quantization for Long-Context LLM Inference via Mixture of Quantization-Aware Experts

Wei Tao[♠♡], **Haocheng Lu**[♠♡], **Xiaoyang Qu**[♡*], **Bin Zhang**[♠♡], **Kai Lu**[♠*], **Jiguang Wan**[♠], **Jianzong Wang**[♡]
[♠]Huazhong University of Science and Technology,
[♡]Ping An Technology (Shenzhen) Co., Ltd.
Correspondence: quxiaoy@gmail.com, kailu@hust.edu.cn

Abstract

One of the primary challenges in optimizing large language models (LLMs) for long-context inference lies in the high memory consumption of the Key-Value (KV) cache. Existing approaches, such as quantization, have demonstrated promising results in reducing memory usage. However, current quantization methods cannot take both effectiveness and efficiency into account. In this paper, we propose MoQAE, a novel mixed-precision quantization method via mixture of quantization-aware experts. First, we view different quantization bit-width configurations as experts and use the traditional mixture of experts (MoE) method to select the optimal configuration. To avoid the inefficiency caused by inputting tokens one by one into the router in the traditional MoE method, we input the tokens into the router chunk by chunk. Second, we design a lightweight router-only fine-tuning process to train MoQAE with a comprehensive loss to learn the trade-off between model accuracy and memory usage. Finally, we introduce a routing freezing (RF) and a routing sharing (RS) mechanism to further reduce the inference overhead. Extensive experiments on multiple benchmark datasets demonstrate that our method outperforms state-of-the-art KV cache quantization approaches in both efficiency and effectiveness.

1 Introduction

In recent years, large language models (LLMs) have become a cornerstone in many fields, including natural language processing (Dubey et al., 2024), computer vision (Lin et al., 2024a), time series data (Tao et al., 2025a) and so on. As these models continue to evolve, the need to handle longer and more intricate texts has also grown significantly. Some complicatrd tasks often require models capable of handling extended contexts that span thousands of tokens. Although the newest LLM can handle up to 2 million input tokens (Team et al., 2024), the long-context inference still presents substantial challenges in memory consumption and computational efficiency. We have plotted the composition of the memory usage of the Llama2-13B model in relation to the context length in Figure 1 (The part beyond the device memory limit is our estimation). The memory occupied by the weights is fixed, while the memory occupied by the Key-Value (KV) cache is proportional to the context length. When the context length is small, the memory usage is still dominated by the weights. However, as the context length increases, it quickly shifts to being dominated by the memory usage of the KV cache. Ultimately, when the context length reaches 128k, the memory usage of the KV cache can reach 100GB, far beyond the memory capacity of most commodity GPUs. Obviously, during long-context inference, the main bottleneck in memory usage lies in the KV cache. Furthermore, the frequent transfer of large KV caches between CPU and GPU memory for computation exacerbates the problem, leading to significant inference latency.

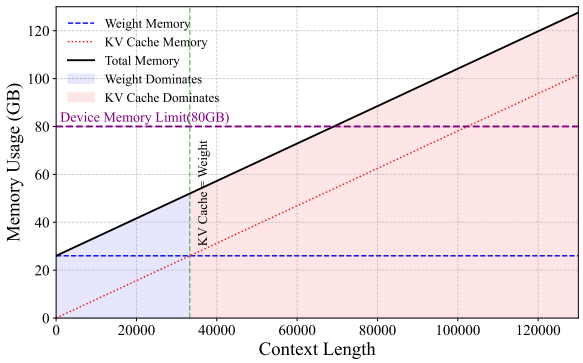

Figure 1: The composition of LLM inference memory under different context lengths on an NVIDIA A100 GPU with 80GB memory capacity.

Researchers have proposed various methods to optimize LLMs for long-context inference, including pruning, knowledge distillation, and quantization. Among them, quantization is the easi-

*Xiaoyang Qu (email: quxiaoy@gmail.com) and Kai Lu (email: kailu@hust.edu.cn) are the corresponding authors.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 10810–10820

est method to implement and can reduce memory consumption the most. Some researchers propose uniform quantizing models to low bit-width, which achieve great performance on memory reduction but can cause drastic accuracy degradation. Other researchers design mixed-precision quantization, which keeps the important tokens in high bit-width to maintain the model accuracy. However, these mixed-precision methods require complex and time-consuming quantization search processes to determine the bit-width configuration.

Inspired by MoICE (Lin et al., 2024b), which employs the experts in the mixture of experts (MoE) module as the bases of rotary position embedding (RoPE), we leverage the advantages of the mixture of experts (MoE) approach's fast training and inference speed to propose MoQAE, a novel mixed-precision KV cache quantization method via mixture of quantization-aware experts. Our main innovation is to creatively use MoE technology to learn the quantization bit-width configuration. Specifically, our contributions consist of three components. (1) We treat each kind of quantization bit-width configuration as an expert (which is also the origin of the name "quantization-aware expert") and leverage the router in the MoE method to select the most suitable quantization bit-width. That is, we input a token into a router, which identifies the most suitable expert for that token. The quantization bit-width corresponding to that expert is the bit-width to which we need to quantize the token. We input tokens chunk-by-chunk instead of using the token-by-token manner in traditional MoE methods. (2) We design a lightweight fine-tuning process. Instead of training the entire LLM, we freeze the pre-trained LLM's parameters and perform minimal fine-tuning on the MoE routers using a calibration dataset. During fine-tuning, we introduce a comprehensive loss that balances model accuracy and memory usage. (3) We propose a routing-freezing (RF) and a routing sharing (RS) mechanism. The RF mechanism freezes the quantization strategy of initial chunks to keep model accuracy, while the RS mechanism allows the quantization strategy to be shared across different LLM blocks.

2 Background

2.1 Preliminaries

LLM Inference. Modern LLM architectures are predominantly based on a decoder-only structure, where inference is divided into two distinct stages: the prefill stage and the decoding stage. In the prefill stage, all input tokens are processed by the LLM to generate the first output token. Subsequently, during the decoding stage, a sequence comprising all input tokens and the tokens already generated is processed by the LLM to generate the next output token. This process repeats iteratively, with each newly generated token appended to the sequence for subsequent processing, until the entire output sequence is completed. A significant drawback of this approach is that, at each step, the key (K) and value (V) matrices corresponding to the input tokens and all previously generated tokens must be recomputed, leading to inefficiencies. To address this, modern LLMs utilize a KV cache, which stores the K and V matrices of both input and generated tokens, eliminating redundant computations and substantially reducing inference latency. However, when processing long input texts, the size of the KV cache grows dramatically, consuming a large amount of GPU memory and making model deployment infeasible on resource-constrained hardware. Moreover, the frequent transfer of the KV cache between CPU and GPU memory becomes more time-consuming as its size increases, turning the KV cache into a bottleneck for inference latency.

Mixture of Experts. MoE is a model architecture designed to divide computational tasks among multiple experts (sub-models) and dynamically select a subset of experts to process a given input using a routing mechanism. Recently, MoE architectures have been widely adopted in LLMs, such as Switch Transformer (Fedus et al., 2022) and GLaM (Du et al., 2022). Traditionally, MoE treats each feed-forward network (FFN) layer in the LLM as an expert, and a router dynamically activates only a small subset of these FFN layers based on the input, while the inactive layers remain idle. This strategy has since been extended to self-attention layers as well (Zhang et al., 2022). Compared to dense models, MoE's sparse activation mechanism significantly reduces computational overhead while maintaining excellent scalability in parameter size. In this work, rather than viewing LLM layers as experts, we innovatively treat the quantization bit-width configurations of the KV cache in LLMs as experts and propose quantization-aware experts.

2.2 Related Works

KV Cache Optimization. Researchers have proposed various methods to optimize the KV cache in

LLMs. Some (Zhang et al., 2023; Xiao et al., 2024; Han et al., 2024; Liu et al., 2024a; Ge et al., 2024; Pagliardini et al., 2023) have introduced pruning techniques to eliminate the KV cache of less important tokens. For example, Zhang et al. propose H_2O (Zhang et al., 2023), which removes tokens whose sum of vertical attention scores in the attention weight matrix is the lowest. StreamingLLM (Xiao et al., 2024) proposes an "attention sink" mechanism, and only keeps the initial tokens and the most recent tokens. Others (Song et al., 2024; Xue et al., 2024; He and Zhai, 2024; Kwon et al., 2023; Dao et al., 2022; Yu et al., 2022; Cai et al., 2024; Jin et al., 2023) have focused on memory management strategies, addressing KV cache fragmentation from a system-level perspective. For instance, vLLM (Kwon et al., 2023) constructs a page table that maps the continuous logical pages of the KV cache to non-contiguous physical memory pages, while also employing a copy-on-write mechanism to reduce memory usage. Jin et al. propose S3 (Jin et al., 2023), which predicts the output sequence length during inference and allocates KV cache memory space according to the prediction result, avoiding memory waste caused by over-allocating KV cache space. Additionally, quantization (Liu et al., 2024b; Hooper et al., 2024; Zhao et al., 2024; Frantar et al., 2023; Yang et al., 2024; Kim et al., 2024) has been explored as a promising approach to convert KV cache data from high-precision to low-precision formats, thereby saving memory. KIVI (Liu et al., 2024b) identifies the presence of many outlier channels in the key cache. Therefore, it proposes quantizing the key cache on a per-channel basis, while the value cache is quantized in the standard per-token manner. Atom (Zhao et al., 2024) applies asymmetric and 4-bit group quantization to the KV cache and performs dequantization before the KV cache computes with the query vector. Among these methods, quantization stands out as one of the most effective and straightforward solutions. However, traditional quantization often incurs significant performance degradation. In this paper, we propose a novel mixed-precision quantization method that achieves near-lossless model performance, addressing the limitations of existing techniques while optimizing KV cache memory usage.

Mixed-Precision Quantization. To mitigate the performance degradation caused by quantization, researchers have proposed mixed-precision quantization methods (Hooper et al., 2024; Yang et al., 2024; Zhang et al., 2024b; Kim et al., 2024; Lin et al., 2024c; Tao et al., 2025b). These approaches assign higher bit-widths to tokens of greater importance and lower bit-widths to less critical tokens, thereby maintaining model performance more effectively. In the beginning, researchers apply mixed precision quantization to the weights and activation values of LLM. For example, SqueezeLLM (Kim et al., 2024) divides the weights of LLM into a dense matrix and a sparse matrix, and then uses INT8 quantization on the sparse matrix while keeping the precision of the dense matrix at FP16. AWQ (Lin et al., 2024c) proposes an activation-aware weight quantization, which finds 1% of salient weights through the distribution of activation values and reorders the weights to ensure hardware efficiency. Gradually, as the problems on the KV cache became increasingly prominent, mixed precision quantization has also been extended to the KV Cache. For example, MiKV (Yang et al., 2024) uses the same method as H_2O to determine important tokens, but uses lower-bit quantization instead of evicting them. KVQuant (Hooper et al., 2024) retains high precision of the outlier value (value in large magnitude) in the KV cache during quantization, and designs a new data type nuqX to represent the KV cache after mixed precision quantization. However, most of these methods require a prohibitively long search time to determine the quantization bit-width. In this paper, we propose a novel mixed-precision quantization method via quantization-aware experts. This approach adopts the efficient routers in the MoE method to quickly and effectively learn the optimal quantization configuration for the KV cache.

3 Method

3.1 Overview

Figure 2 shows the overview of MoQAE. The input text is first divided into several equal-length chunks, which are then processed by the LLM. In each block of the LLM, we use a quantization search module to determine the quantization strategy (i.e. quantization bit-width configuration) for the input chunks. Subsequently, these chunks are quantized using the bit-width configuration just determined, and proceeds with the formal calculation in the block (attention and feed-forward computations). Finally, the output chunk is passed to the next block, where the process is repeated. Notably, we apply a routing-freezing mechanism to the first chunk,

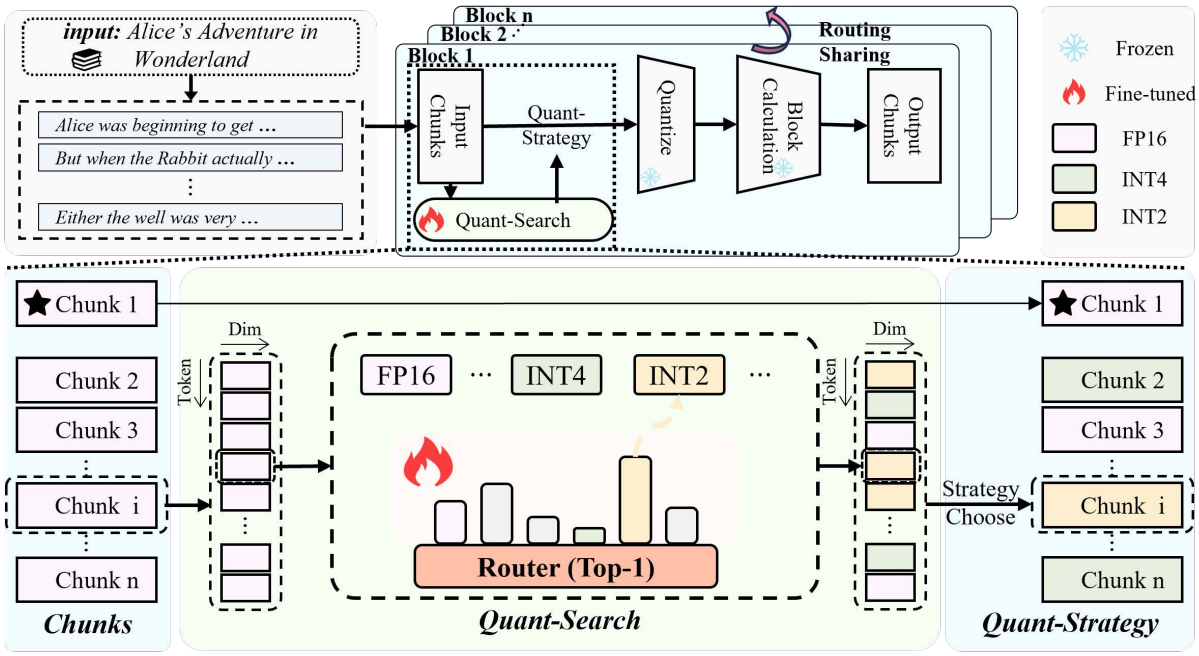

Figure 2: The overview of MoQAE. We use the router in MoE technology to learn the optimal quantization strategy.

preventing it from entering the router and fixing its bit-width to FP16. Additionally, we adopt a routing sharing mechanism between blocks, allowing different blocks to use the same quantization strategy.

3.2 Quantization-Aware Experts

In the quantization search module, we introduce a router and several attention-aware experts. These experts represent different quantization bit-width configurations, such as FP16, INT4, INT2, and so on. The input text is divided into several equal-length chunks, and for the residual part that do not meet the chunk size, we directly retain their precision as FP16. Within each block of the LLM, the chunks are first passed into a router, where the router network is implemented using an MLP with the function:

$$\mathcal{P} = f(CW_1 \cdot CW_2)W_3 \tag{1}$$

Here, $C \in \mathbb{R}^{N\times D}$ is the input chunk, $f()$ is the activation function, $W_1, W_2 \in \mathbb{R}^{D\times M}$ and $W_3 \in \mathbb{R}^{D\times M}$ are weight matrices, where D is the embedding dimension size within each attention head, N is the chunk size, M is the expert amount. The output $\mathcal{P} \in \mathbb{R}^{N\times M}$ reflects the probabilities of all the chunks about selecting which expert.

For each token in the chunk, the expert with the highest selection probability is chosen as the selected expert for that token. Subsequently, we find out the expert that is selected the most times within the chunk and denote it as the quantization strategy for the entire chunk. The equation is as follows:

$$\mathcal{R} = \arg\max_{1\le k\le M}\left(\sum_{i=1}^{N}\mathbb{I}\left(\arg\max_{1\le j\le M} p_j^i = k\right)\right) \tag{2}$$

Where $\mathcal{R} \in \{1, 2, ..., M\}$ is the quantization strategy, p_j^i means the probability of selecting expert j for chunk i, $\mathbb{I}()$ operator means that the result is 1 if the condition is satisfied otherwise 0. Finally, we integrate all the selected experts, generating the quantization strategy for all the chunks, and the input text will be quantized with this quantization strategy.

3.3 Fine-Tuning Process

To accelerate the training process, we design an efficient training method: freezing the parameters of the LLM itself and fine-tuning only the router's parameters. Additionally, our fine-tuning is conducted on a subset of the original dataset called the calibration dataset.

We further design a novel loss in the fine-tuning process. The goal of this loss is to achieve a trade-off between the accuracy of the LLM and memory usage during long-context inference. The design details of this loss are as follows:

On one hand, to optimize the model's accuracy, we incorporate the model's negative log-likelihood

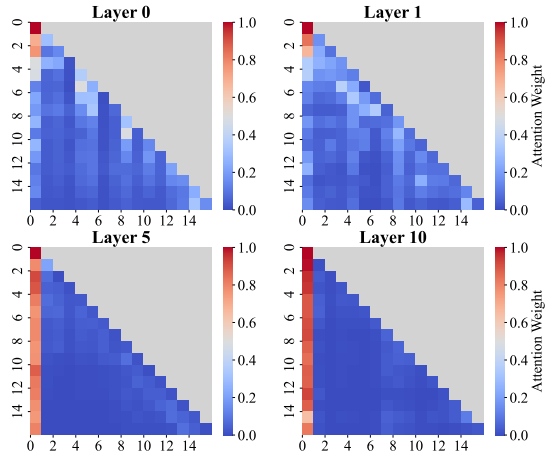

Figure 3: Attention weights of the first few tokens in different layers of Llama2-7b.

loss L_{nll} as part of the final loss. However, we cannot directly apply L_{nll} because it does not involve operators directly related to the router's weights, making it unable to train the router's weights. Therefore, we define a new loss called L_{model}, which is obtained by multiplying L_{nll} by the mean value of the expert selection probabilities output by the router. To reflect the varying importance of different experts to the model's accuracy, we apply a penalty term to each component of this product. L_{model} is ultimately computed as follows:

$$L_{model} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left(\arg\max_{1\le k\le M} p_k^i = j\right)\cdot\frac{p_j^i \cdot L_{nll}}{B_j} \tag{3}$$

where p_k^i means the probability of selecting expert k for chunk i, $1/B_j$ is the penalty term for expert j and B_j means the corresponding bit-width of expert j. We choose $1/B_j$ as the penalty term because data with lower bit-width leads to higher model loss.

On the other hand, to ensure that our method also optimizes memory usage, we introduce the memory loss L_{mem}. The purpose of L_{mem} is to encourage the router to preferentially select experts that represent lower bit-widths, thereby reducing the model's GPU memory usage. We also calculate L_{mem} as the weighted sum of the mean value of the expert selection probabilities, but the penalty term is applied in an inverted manner:

$$L_{mem} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left(\arg\max_{1\le k\le M} p_k^i = j\right)\cdot\frac{16p_j^i}{B_j} \tag{4}$$

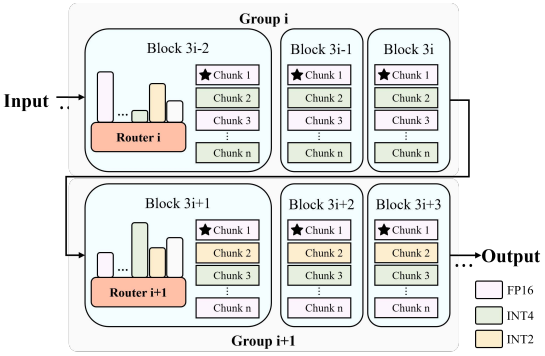

Figure 4: The routing sharing mechanism.

Here we choose $\frac{16}{B_j}$ as the penalty term. This is because data with higher-bitwidth leads to more memory consumption.

Finally, our loss is defined as follows:

$$L = \lambda L_{model} + (1-\lambda)L_{mem} \tag{5}$$

where λ is a pre-defined hyperparameter that controls the trade-off between model accuracy and memory usage. We will discuss the impact of λ on model performance in Section 4.3.

3.4 Routing Freezing and Routing Sharing

Previous researchers (Xiao et al., 2024) have demonstrated that the token at the initial position of an LLM plays a crucial role in the model's performance, significantly influencing its accuracy. In our research, we also explore this by conducting an experiment to investigate the attention weights of initial tokens of different layers within the LLM. As depicted in Figure 3, we observe that the attention weights for tokens at the initial positions are relatively higher than those for tokens in subsequent positions (except for the first two layers). This finding strongly suggests that tokens at the beginning of the sequence are highly influential, playing a critical role in determining the model's output. These initial tokens seem to capture essential contextual information, which is then propagated through the rest of the sequence.

In response to these observations, we introduce a routing freezing mechanism to ensure that the critical tokens at the initial position are not compromised during the quantization process. Specifically, we prevent the first chunk of tokens from being passed into the router and restrict it to selecting the FP16 quantization configuration. This approach guarantees that the tokens at the start of the sequence are preserved with higher precision and are

Table 1: The perplexity of MoQAE and baseline methods on Wikitext2 dataset, lower is better. **AvB** means average bit-width. Most of the data is cited from CQ (Zhang et al., 2024a).

Bit Range	Methods	AvB	LLama-7B ↓	LLama-13B ↓	LLama2-7B ↓	LLama2-13B ↓	Mistral-7B ↓
=16bits	FP16	16	5.68	5.09	5.11	4.57	5.07
4∼16bits	INT4 ①	4.00	7.40	6.82	7.31	6.59	5.91
	INT4-gs128 ①	4.16	7.16	6.67	6.87	6.20	5.76
	NF4 ②	4.00	7.27	6.74	7.09	6.45	5.85
	NF4-gs128 ②	4.16	7.16	6.66	6.86	6.20	5.77
	KVQuant-4b ③	4.00	7.13	6.65	6.70	6.11	5.75
	KVQuant-4b-1% ③	4.32	7.09	6.62	6.65	6.06	5.72
	CQ-2c8b ④	4.00	7.11	6.64	6.67	6.09	5.74
	Atom-4b-gs128 ⑤	4.00	6.16	5.46	5.98	5.26	5.67
	QoQ-4b ⑥	4.00	5.93	5.28	5.88	5.32	5.62
	QoQ-4b-gs128 ⑥	4.00	5.89	5.25	5.89	5.24	5.66
	AWQ ⑦	4.00	6.33	5.59	6.51	5.43	6.24
	AWQ-gs128 ⑦	4.00	5.93	5.36	5.92	5.27	5.66
	MiKV ⑧	5.50	6.25	5.58	5.89	5.33	5.78
	MoQAE-λ0.5	4.13	**5.76**	**5.15**	**5.22**	**4.65**	**5.14**
2∼4bits	INT2①	2.00	10892	100870	4708	4220	477
	INT2-gs128①	2.14	43.49	56.25	113.49	97.04	50.73
	NF2 ②	2.00	2850.1	4680.3	13081.2	4175.6	1102.3
	NF2-gs128 ②	2.14	248.32	118.18	420.05	499.82	191.73
	KVQuant-2b ③	2.00	10.28	9.05	15.16	43.77	8.40
	KVQuant-2b-1% ③	2.32	7.38	6.83	7.06	6.38	6.08
	CQ-4c8b④	2.00	7.52	6.96	7.23	6.52	6.17
	Atom-2b-gs128⑤	2.00	37.37	41.77	-	-	-
	MoQAE-λ0.3	3.50	8.17	**6.44**	**6.26**	7.03	**6.03**

not quantized to lower bit-widths, thus protecting the model's accuracy.

Additionally, we propose a routing sharing mechanism to optimize the inference process further. Our insight is inspired by CLA (Brandon et al., 2024), which demonstrates the feasibility of sharing key and value heads across different attention layers to reduce computational overhead. As illustrated in Figure 4, in this mechanism, we partition the different blocks within the LLM into several groups. In each group, the other blocks share the quantization strategy of the first block. The routers in other blocks are also removed. By the routing sharing mechanism, we can effectively reduce the memory usage caused by too many routers and the latency caused by router computation in most of the blocks. Although sharing routing strategies between different blocks may lead to a slight loss in model accuracy (since the quantization strategy of the KV cache in one block may not be applicable to the next block), this loss is not very severe (We will prove it in Section 4.3). At the same time, the routing sharing mechanism can significantly reduce memory usage and computation latency. Therefore, we believe that this loss is acceptable. We also explore the impact of the group size on model performance in Section 4.3.

4 Evaluation

4.1 Experimental Setup

Benchmarks.

We benchmark MoQAE on six widely-used open-source models: Llama-7B, Llama-13B(Touvron et al., 2023a), Llama2-7B, Llama2-13B (Touvron et al., 2023b), Llama3-8B (Dubey et al., 2024), and Mistral-7B (Jiang et al., 2023). To assess performance, we evaluate the perplexity of MoQAE on the WikiText2 (Merity et al., 2017) dataset. We also adopt LongBench (Bai et al., 2024) to further evaluate the long-context generation performance of our method and the baselines. We choose eight subsets from four different task types in LongBench as our practical datasets. They are single document QA task (Qasper), summarization task (QMSum, MultiNews), few-shot learning task (TREC, TriviQA, SAMSum), and code completion task (LCC, RepoBench-P). F1 score is used as the evaluation metric for Qasper and TriviaQA, while ROUGE score is used for QMSum, and MultiNews, and similarity score is used for LCC and RepoBench-P. Only TREC uses classification score as the evaluation metric. The maximum con-

Table 2: The performance of MoQAE and baseline methods on LongBench datasets, higher is better.

Method	Qasper ↑	QMSum ↑	MultiNews ↑	TREC ↑	TriviaQA ↑	SAMSum ↑	LCC ↑	RepoBench-P ↑
FP16	9.52	21.28	3.51	**66.00**	87.72	41.69	66.66	59.82
KIVI-2b ⑧	9.26	20.53	0.97	**66.00**	87.42	**42.61**	66.22	59.67
CQ-4c8b ④	9.58	20.87	1.93	**66.00**	87.72	41.13	**66.57**	59.75
MiKV ⑧	9.14	20.63	0.85	65.88	87.21	41.44	66.18	59.55
MoQAE	**9.79**	**21.23**	**3.47**	**66.00**	**87.89**	41.37	66.53	**59.94**

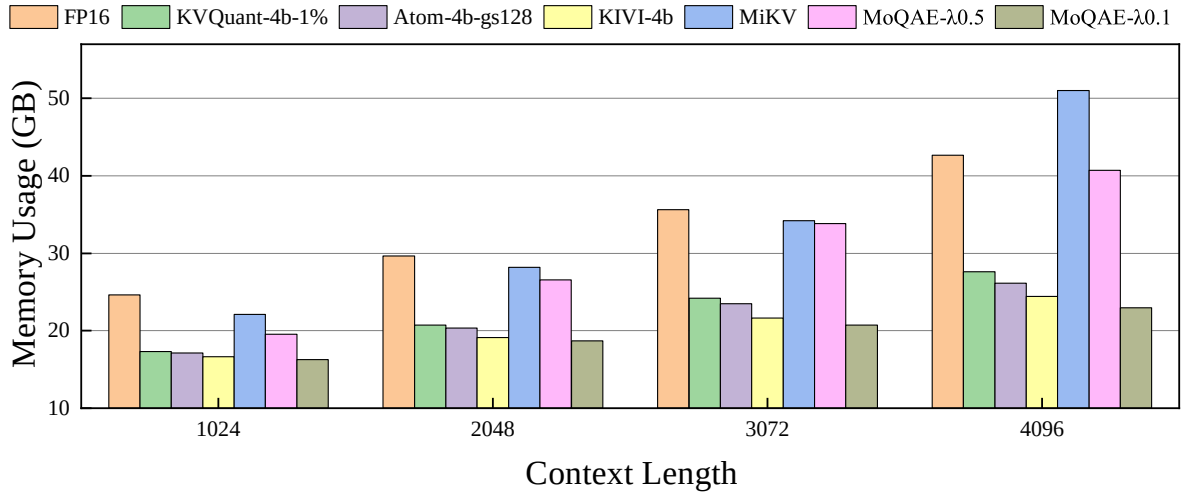

Figure 5: The memory usage of MoQAE and baseline methods under different context lengths.

text length is 2048 for Llama, 4096 for Llama-2, Llama-3, and 8192 for Mistral, respectively.

Baselines. We compare MoQAE with the FP16 full precision model and nine other state-of-the-art KV cache quantization methods as the baselines: ① INT, which means uniform integer quantization. ② NF, which means NormalFloat quantization. ③ KVQuant (Hooper et al., 2024), which keeps outlier value in high bit-width. KVQuant-[x]b-1% means 1% of the tokens is kept as FP16 precision. ④ CQ (Zhang et al., 2024a), which couples multiple key/value channels together to exploit their inter-dependency. CQ-[x]c[y]b means that each group has x channels and there are y bits in a quantized code for a group. ⑤ Atom (Zhao et al., 2024), which uses asymmetric uniform quantization with the granularity of attention head. ⑥ QoQ (Lin et al., 2025), which scales queries and keys to decrease the loss caused by quantizing the outlier values in the key cache. ⑦ AWQ (Lin et al., 2024c), which applies uniform 4-bit quantization to the KV cache. ⑧ MiKV (Yang et al., 2024), which employs mixed-precision quantization by computing the attention score sum of each token and quantizing those with low attention score sum to lower bit-width while keeping the rest at higher bit-width. ⑨ KIVI (Liu et al., 2024b), which uses per-channel quantization to the key cache and per-token quantization to the value cache. The quantization bit-width for each token is assigned based on their saliency. Among them, ①, ②, ④, ⑤, ⑥, ⑦, ⑨ are uniform quantization; ③, ⑧ are mixed-precision quantization. The suffix "gs" in the method name indicates the group size, while other method names that do not contain "gs" means that those methods do not use group quantization.

Implementation. We conduct our experiments on an NVIDIA H20-NVLink GPU containing 96 GB of memory, along with a 25-core AMD EPYC 7T83 CPU and 100GB of RAM. Chunks size is set as 32, and λ is set as 0.5. Group size in the routing sharing mechanism is set as 3. The router consists of a 2-layer MLP with a hidden dimension of expert amount. We use SiLU as the activation function and top-1 expert selection as the routing mechanism. The memory usage of the parameters of the router is about 1.6KB. As for training, we use 5% of the full training set as the calibration dataset. We use AdamW as the optimizer, with learning rate 3e-4 and batch size 8.

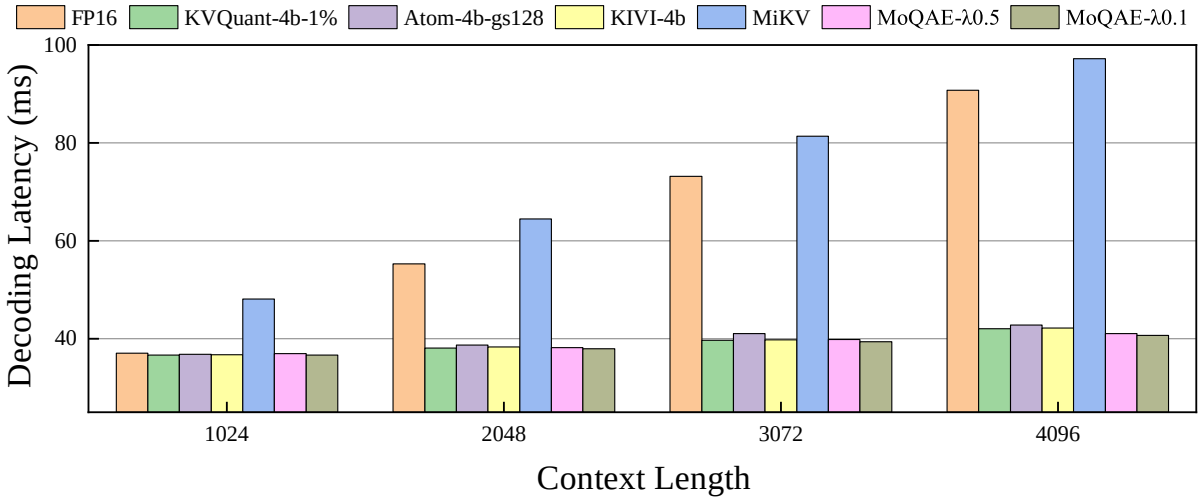

Figure 6: The decoding latency of MoQAE and baseline methods under different context lengths.

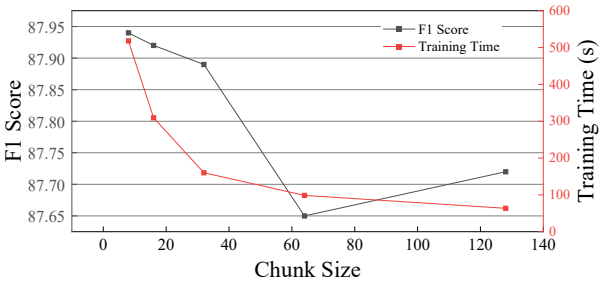

Figure 7: The impact of chunk size on model performance and training time.

Table 3: The impact of chunk size on decoding latency.

Chunk Size	8	16	32	64	128
Decoding Latency/ms	24.85	24.26	23.86	23.59	23.01

4.2 Performance

We first evaluate the perplexity on Wikitext2 dataset. The results are shown in Table 1. We additionally test the case where λ is 0.3. As can be seen from the table, simple quantization to extremely low bit-widths (2 bits) results in significant accuracy loss. Even with meticulously designed quantization methods, as the bit-width decreases, the model's accuracy rapidly declines. Compared to other methods, MoQAE is able to reduce the model's average bit-width to a relatively low level while maintaining model accuracy well. Among methods with 4-16 bits, MoQAE-λ0.5 achieves the least perplexity with similar average bit-width with baseline methods. The perplexity of MoQAE-λ0.5 is only 0.08 more than the FP16 models on average. MoQAE-λ0.3 also outperforms methods with 2-4bits on most models.

We also compare the performance of MoQAE and other methods on LongBench datasets. As shown in Table 2, MoQAE achieves the best performance on most of the datasets. The performance of MoQAE is only a little worse than baseline methods on SAMSum and LCC datasets.

Furthermore, we evaluate the memory usage and decoding latency of MoQAE and other methods under different context lengths with batch size 8. We test MoQAE under two kinds of λ. As shown in Figure 5 and Figure 6, MoQAE-λ0.1 achieves the least memory usage and decoding latency over all the context lengths.

Compared with the state-of-the-art (SOTA) quantization methods, MoQAE can reduce the memory usage by 0.79GB and reduce the decoding latency by 0.44ms, on average. The efficiency of MoQAE-λ0.5 is worse than MoQAE-λ0.1, but it still reduces the memory usage of FP16 model by 2.99GB on average and outperforms most of the baseline methods on decoding latency on decoding latency.

4.3 Ablation Study

We explore the impact of chunk size on model performance. The results are shown in Figure 7 and Table 3. As the chunk size increases, the training time decreases significantly and so does the decoding latency decreases. The model accuracy shows a trend of first decreasing and then increasing slightly. This is because when the chunk size becomes larger, some important token information will be wrapped in more unimportant token information within a chunk. Such a chunk may be misidentified as INT2

Table 4: The impact of λ on model performance.

λ	0.1	0.3	0.5	0.7	0.9
F1 Score	87.32	87.64	87.89	87.91	87.92
Average Bits	3.45	3.65	4.2	10.40	12.12
Memory Usage/GB	14.01	14.04	15.95	15.33	15.88

Table 5: The impact of our RF and RS mechanism. "gs" means group size in the RS mechanism.

Method	F1 Score	Decoding Latency/ms
FP16	87.72	9.7
MoQAE w/o RF	87.88	20.6
MoQAE w/o RS	87.92	31.7
MoQAE (gs=2)	87.92	25.7
MoQAE (gs=4)	87.81	16.1
MoQAE	87.89	20.7

quantization by the router, resulting in the loss of important information. When the chunk size is large, since we fix the first chunk to FP16, more important information is saved, which slightly improves the model accuracy.

We further conduct ablation experiments on the hyperparameter λ. As shown in Table 4, with the increase of λ, the model accuracy increases (The accuracy reaches the upper limit after λ is greater than 0.5) while average bits and memory usage decreases. This result demonstrates that λ can effectively balance model accuracy and memory usage. We also test the impact of routing freezing and routing sharing mechanisms. When routing freezing is removed from MoQAE, as can be seen from Table 5, both accuracy and inference latency are slightly reduced. This is because the first chunk of some blocks may change from the original fixed FP16 to other lower bit-widths. When routing sharing is removed, the decoding latency is significantly improved, while the accuracy is slightly increased. This is because after removing routing sharing, we need to perform more router calculations, but the calculated bit-width configuration will also be more accurate. At the same time, we test the impact of different group sizes in the routing sharing mechanism. It can be seen that as the group size increases, the decoding latency is significantly reduced, but the accuracy also slightly decreases.

5 Conclusion

In this paper, we introduce MoQAE, a novel mixed-precision quantization method based on mixture of quantization-aware experts. First, we treat different quantization bit-width configurations as experts and apply the traditional MoE method to select the optimal configuration. To avoid the inefficiency of inputting tokens one by one in the conventional MoE method, we feed the tokens into the router chunk by chunk. Second, we propose a lightweight router-only fine-tuning process and design a novel loss that enables the model to learn the trade-off between model accuracy and memory usage. Finally, we introduce the RS and RF mechanisms, which further reduces the inference overhead caused by the routers. Extensive experiments on benchmark datasets show that our method outperforms SOTA mixed-precision quantization techniques in terms of both efficiency and effectiveness.

6 Limitations

Since our method introduces additional routers in LLM, the parameters of these routers will occupy a part of the memory, and the calculation of the router will also slow down the inference time of the model. Although we have adopted methods such as chunk input and routing sharing to optimize, these overheads still exist.

In addition, in order to ensure the accuracy of the attention calculation results, since softmax has high precision requirements when calculating the attention weight, we will dequantize the quantized key vector to FP16 and calculate it with the FP16 query vector. This dequantization operation will also cause additional delays.

7 Acknowledgements

This work was sponsored by the Key Research and Development Program of Guangdong Province under Grant No.2021B0101400003, the National Key Research and Development Program of China under Grant No.2023YFB4502701, the National Natural Science Foundation of China under Grant No.62172175, the China Postdoctoral Science Foundation under Grant No.2024M751011, the Postdoctor Project of Hubei Province under Grant No.2024HBBHCXA027.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3119–3137.

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*.

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. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5209–5235.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 16344–16359.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. In *The Twelfth International Conference on Learning Representations*.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008.

Jiaao He and Jidong Zhai. 2024. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421*.

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

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2024. Squeezellm: dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, pages 23901–23923.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. 2024a. Video-llava: Learning united visual representation by alignment before projection. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5971–5984.

Hongzhan Lin, Ang Lv, Yang Song, Hengshu Zhu, Rui Yan, et al. 2024b. Mixture of in-context experts enhance llms' long context awareness. *Advances in Neural Information Processing Systems*, 37:79573–79596.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024c. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2025. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. In *Proceedings of Machine Learning and Systems*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024a. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32332–32344.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Matteo Pagliardini, Daniele Paliotta, Martin Jaggi, and François Fleuret. 2023. Faster causal attention over large sequences through sparse flash attention. *arXiv preprint arXiv:2306.01160*.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 590–606.

Wei Tao, Xiaoyang Qu, Kai Lu, Jiguang Wan, Guokuan Li, and Jianzong Wang. 2025a. Madllm: Multivariate anomaly detection via pre-trained llms. *arXiv preprint arXiv:2504.09504*.

Wei Tao, Bin Zhang, Xiaoyang Qu, Jiguang Wan, and Jianzong Wang. 2025b. Cocktail: Chunk-adaptive mixed-precision quantization for long-context llm inference. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 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, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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

Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. 2024. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024a. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Advances in Neural Information Processing Systems*, 37:3304–3331.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2022. Mixture of attention heads: Selecting attention heads per token. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4150–4162.

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, et al. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.