QPruner: Probabilistic Decision Quantization for Structured Pruning in Large Language Models

Changhai Zhou[1,3], **Yuhua Zhou**[2], **Yibin Wang**[1],
Shijie Han[4], **Qian Qiao**[5], **Hongguang Li**[3],
[1]Fudan University, [2]Zhejiang University, [3]JF SmartInvest Holdings,
[4]Columbia University, [5]Soochow University.
zhouch23@m.fudan.edu.cn zhouyuhua@zju.edu.cn yibinwang1121@163.com
sh4460@columbia.edu qqiao@stu.suda.edu.cn harvey2@mail.ustc.edu.cn

Abstract

The rise of large language models (LLMs) has significantly advanced various natural language processing (NLP) tasks. However, the resource demands of these models pose substantial challenges. Structured pruning is an effective approach to reducing model size, but it often results in significant accuracy degradation, necessitating parameter updates to adapt. Unfortunately, such fine-tuning requires substantial memory, which limits its applicability. To address these challenges, we introduce quantization into the structured pruning framework to reduce memory consumption during both fine-tuning and inference. However, the combined errors from pruning and quantization increase the difficulty of fine-tuning, requiring a more refined quantization scheme. To this end, we propose QPruner, a novel framework that employs structured pruning to reduce model size, followed by a layer-wise mixed-precision quantization scheme. Quantization precisions are assigned to each layer based on their importance to the target task, and Bayesian optimization is employed to refine precision allocation strategies, ensuring a balance between model accuracy and memory efficiency. Extensive experiments on benchmark datasets demonstrate that QPruner significantly outperforms existing methods in memory savings while maintaining or improving model performance.

1 Introduction

The advent of large language models (LLMs) has revolutionized various natural language processing (NLP) tasks, such as machine translation (Zhang et al., 2023a; Sato et al., 2020), sentiment analysis (Zhang et al., 2023b; Deng et al., 2023), and speech recognition (Min and Wang, 2023). Despite their impressive capabilities, the resource consumption required to obtain a fine-tuned model suitable for specific tasks remains substantial due to the large number of parameters and high computational demands of LLMs (Frantar and Alistarh, 2023). To address these issues, various compression techniques, including pruning (Molchanov et al., 2019; Liu et al., 2018), quantization (Shao et al., 2023; Lee et al., 2023), and distillation (Gu et al., 2023; Tan et al., 2023), have been proposed.

Structured pruning (Ma et al., 2023; Xia et al., 2023) is a widely used approach that reduces model size by removing less important parameters in a structured manner, preserving the overall architecture compatibility with hardware requirements. However, the disruption of computational graph uniformity and the removal of parameters can significantly reduce the accuracy of LLMs, which are inherently information-dense networks. To mitigate this degradation, fine-tuning is often used to recover the accuracy of pruned models. This fine-tuning step, while effective, is memory-intensive and presents substantial challenges in terms of resource consumption.

To further reduce memory usage during the fine-tuning and inference phases, we introduce quantization into the structured pruning framework. Specifically, after performing structured pruning, we quantize the pruned model and then apply different fine-tuning strategies. Quantization effectively reduces the bit-width of model parameters, thereby lowering the resource consumption during both fine-tuning and inference. However, integrating quantization with structured pruning introduces additional complexities. Structured pruning applies different pruning intensities across model layers, which exacerbates the uneven distribution of layer importance, making some layers more critical for maintaining model performance. Moreover, the cumulative quantization error varies across different layers, potentially amplifying the performance degradation caused by pruning. Therefore, a simple, uniform quantization scheme is suboptimal. Instead, a more nuanced, layer-wise mixed-precision quantization approach is needed. By allowing more

Findings of the Association for Computational Linguistics: NAACL 2025, pages 4276–4286
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

critical layers to maintain higher precision, we can better control the overall performance of the model.

Building upon these observations, we propose a new framework called QPruner. In QPruner, we first apply structured pruning to reduce the model size, followed by a quantization phase where different quantization precisions are assigned to each layer based on their contribution to the target task. To further improve the allocation strategy, Bayesian optimization (Frazier, 2018) is employed to explore better precision configurations. Finally, we apply parameter-efficient fine-tuning (PEFT) fine-tuning strategy, to recover model performance. This integrated approach aims to strike an optimal balance between model accuracy and memory efficiency, making it well-suited for resource-constrained scenarios. The main contributions of this work are summarized as follows:

- We propose QPruner, a novel framework that integrates structured pruning and quantization, aiming to significantly reduce the memory consumption of LLMs during both fine-tuning and inference.
- We introduce a mixed-precision quantization scheme where quantization precisions are assigned to each layer based on their importance to the target task, with Bayesian optimization used to further refine precision allocation strategies.
- We demonstrate QPruner's powerful ability to save memory and maintain performance. It can surpass baseline methods in terms of accuracy by up to 6% while saving at least 30% of memory.

2 Background and Motivation

2.1 Quantization

Quantization. Quantization is an essential technique used to reduce the computational and memory overhead of large-scale models by converting high-precision numerical values, such as a 32-bit floating-point number $X^{\text{HP}} \in \mathbb{R}$, into a lower-bit integer representation $X^{\text{INT}} \in \{0, 1, \ldots, 2^N - 1\}$. This process is mathematically expressed as:

$$X^{\text{INT}} = \text{round}\left((2^N - 1)F\left(X^{\text{HP}}\right)\right), \tag{1}$$

where $F(\cdot)\colon \mathbb{R} \to [0, 1]$ is a normalization function. A typical method is uniform quantization, where $F(X)$ is defined as $F(X) = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$. An alternative approach introduced by QLoRA Dettmers et al. (2024) is 4-bit NormalFloat Quantization (NF4), which assumes that the data follows a normal distribution $X \sim \mathcal{N}(0, \sigma^2)$ and applies $F(X) = \Phi(X/\sigma)$, with $\Phi(\cdot)$ representing the cumulative distribution function of a standard normal distribution.

Dequantization. To recover the high-precision values from their quantized forms, a lookup table $\mathcal{T}$ is used, which is defined as:

$$\mathcal{T}[i] = F^{-1}\left(\frac{i}{2^N - 1}\right), \quad i = 0, 1, \ldots, 2^N - 1, \tag{2}$$

allowing the integer X^{INT} to be mapped back to its simulated high-precision counterpart $X^{\text{D}} \in \mathbb{R}$. The dequantization process can be represented as:

$$X^{\text{D}} = \mathcal{T}[X^{\text{INT}}]. \tag{3}$$

Simulated Quantization for Matrices. In practice, it is often more efficient to use simulated quantization for matrices rather than directly operating on quantized values (Bai et al., 2020; Shen et al., 2020). In this method, quantized weight matrices are stored as encoded integers and are temporarily dequantized into simulated high-precision matrices during multiplication operations. This process is denoted by $q_N(\cdot)\colon \mathbb{R}^{m\times n} \to \mathbb{R}_N^{m\times n}$, where $\mathbb{R}_N : \{\mathcal{T}[i] \in \mathbb{R} | 0 \leq i < 2^N\}$.

2.2 The Motivating Example

Efficient fine-tuning of LLMs on resource-constrained devices requires effective model compression and fine-tuning techniques. After applying structured pruning and quantization, more efficient fine-tuning methods are needed to recover accuracy. One approach is to use LoRA-based methods, as done in LLM-Pruner (Ma et al., 2023), which employs LoRA for quick recovery after structured pruning. Among the LoRA series methods, LoftQ Li et al. (2023) is a method for fine-tuning quantized models. Before fine-tuning, LoftQ iteratively updates the low-rank matrices such that the quantized matrix $\mathbf{Q} + \mathbf{AB}$ approximates the full-precision matrix $\mathbf{W}$, thereby improving the fine-tuning performance, particularly in low-bit settings.

Simply combining pruning, quantization, and LoRA can lead to suboptimal results. Structural pruning reduces model size by removing less important parameters, but due to the varying importance of different layers, it often results in uneven

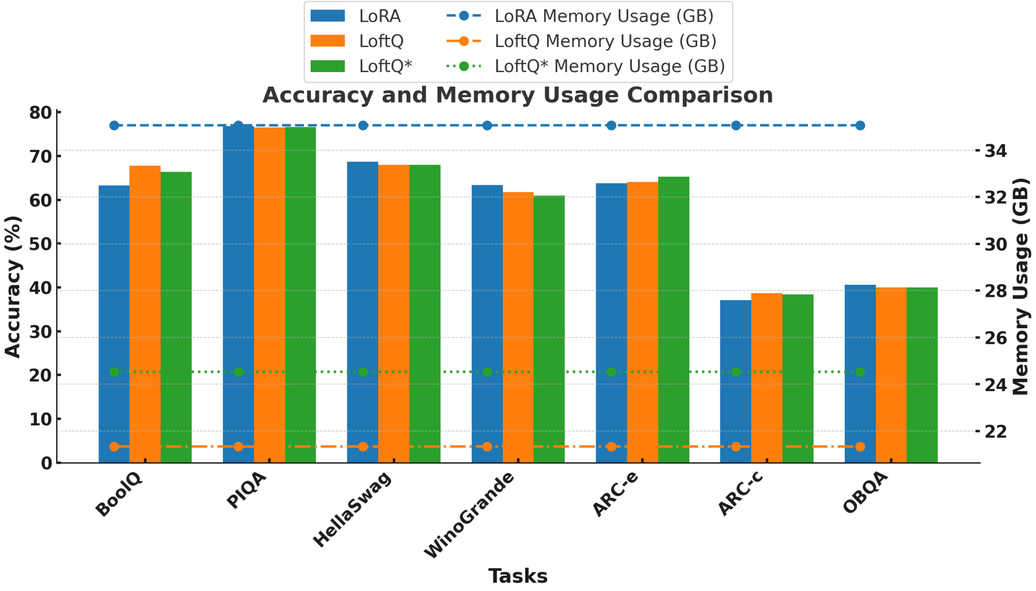

Figure 1: Comparison of accuracy and memory usage across different fine-tuning configurations for multiple tasks. The bars represent the accuracy of three different methods (LoRA, LoftQ, LoftQ*) on each task, while the markers indicate the memory usage for each corresponding method.

pruning across layers. This uneven pruning leads to a complex and unbalanced network structure, and standard quantization typically applies a uniform configuration across all layers. To explore a better trade-off between performance and memory, we adopted mixed-precision quantization, assigning different computational resources and complexities to different layers, with the goal of allowing more important layers to learn with finer granularity.

We conducted experiments using the LLaMA-7b model with a pruning rate of 20%. The pruning was performed using the optimal strategy determined by LLM-Pruner. The methods compared were as follows: LoRA with a uniform 16-bit configuration, LoftQ with a uniform 4-bit quantization, and LoftQ* with a mixed-precision setting of 4 or 8 bits per layer. As shown in Figure 1, the quantized models (LoftQ) achieved performance comparable to the original precision models (LoRA), with significantly lower memory usage (21.33 GB versus 35.06 GB). On some tasks, there was a slight drop in performance, but the mixed-precision model (LoftQ*) demonstrated the potential to further enhance performance while maintaining efficient memory usage.

3 QPruner

Structured pruning, while effective in reducing model size, can disrupt the balance of layer importance, leading to performance degradation. Therefore, parameter adjustments are often necessary to mitigate this imbalance and restore model performance. However, parameter updates require significant memory, which is why we employ quantization techniques to reduce memory consumption. As demonstrated in the motivating example, simply combining pruning and quantization is not always the best choice, as the importance of different layers in a pruned model can vary greatly. We need finer-grained layer-wise quantization bit-width control, which introduces a challenging bit-width allocation problem. To address this, we designed a two-stage allocation strategy to effectively balance these trade-offs.

Based on these insights, we propose QPruner, an integrated framework tailored for efficient or low-resource NLP tasks. It employs structured pruning, mixed-precision quantization, and efficient fine-tuning to solve the challenges of balancing memory efficiency and model performance.

3.1 Structured Pruning

Our framework does not impose specific requirements on the pruning method; as new technologies evolve, the pruning method can be replaced. The only requirement for this step is to produce a smaller model. Although some methods can achieve good performance without fine-tuning (An et al., 2024), most real-time systems require dynamic adaptation, which means that the pruned model must be fine-tuned to improve performance.

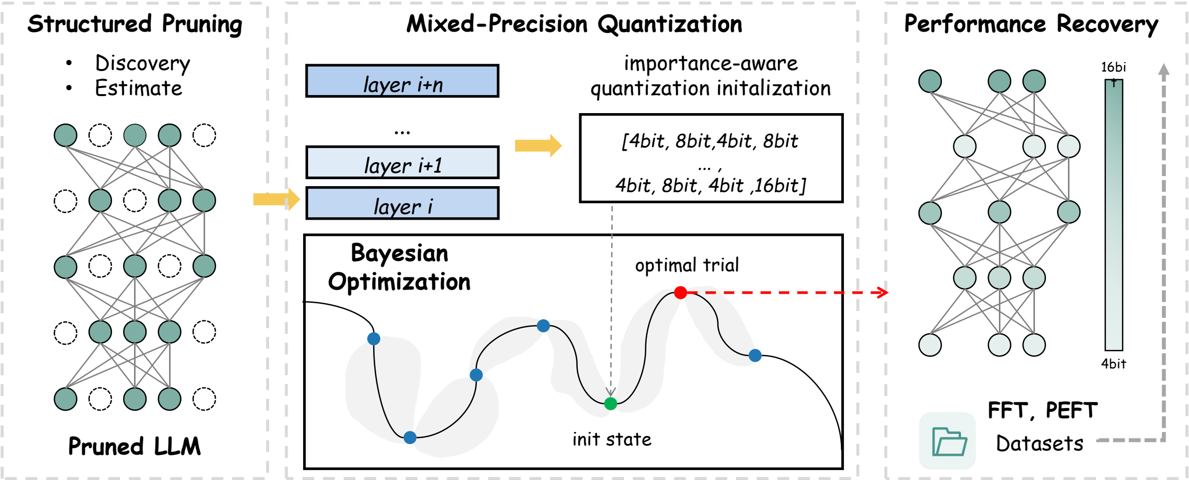

Figure 2: Overview of the QPruner framework.

A popular structured pruning method is LLM-Pruner (Ma et al., 2023), which first identifies dependencies between neurons and groups them, then removes weights based on their importance. Let N_i and N_j be two neurons in the model. If $N_j \in \text{Out}(N_i)$ and $\text{Deg}^-(N_j) = 1$, then N_j is dependent on N_i. Similarly, if $N_i \in \text{In}(N_j)$ and $\text{Deg}^+(N_i) = 1$, then N_i is dependent on N_j. Based on this principle, a dependency graph can be constructed to iteratively identify all coupled structures.

Next, these coupled structures are grouped, and their importance is estimated to effectively perform pruning. For a group of coupled structures $\mathbf{G} = \{\mathbf{W}_i\}_{i=1}^{M}$, its importance can be expressed as:

$$I_{\mathbf{W}_i} = |\mathcal{L}_{\mathbf{W}_i}(\mathcal{D}) - \mathcal{L}_{\mathbf{W}_i=0}(\mathcal{D})|, \tag{4}$$

where $\mathcal{L}$ represents the prediction loss.

Using a second-order Taylor expansion, the importance can be approximated as:

$$\left| \frac{\partial \mathcal{L}(\mathcal{D})}{\partial \mathbf{W}_i} \mathbf{W}_i - \frac{1}{2} \mathbf{W}_i^\top \mathbf{H} \mathbf{W}_i \right|, \tag{5}$$

where $\mathbf{H}$ is the Hessian matrix of the loss function.

For each parameter W_k^i, its importance is defined as:

$$\left| \frac{\partial \mathcal{L}(\mathcal{D})}{\partial W_k^i} W_k^i - \frac{1}{2} (W_k^i)^2 H_{kk} \right|, \tag{6}$$

where H_{kk} is the k-th diagonal element of the Hessian matrix.

Finally, we aggregate the importance of each structure into group-level importance using methods such as summation, multiplication, taking the maximum, or using only the last item. Groups with the lowest importance are selected for pruning, thereby reducing the model size while maintaining performance as much as possible.

3.2 Mixed-Precision Quantization

After pruning, we apply mixed-precision quantization to further reduce memory usage while maintaining model performance. Instead of assigning a uniform bit-width across all layers, different bit-widths are allocated based on each layer's contribution to the final model output. The contribution of each layer is quantified using mutual information between the layer's output and the model's prediction.

To compute mutual information, we first run representative data samples through the pruned model. For each layer, we record its output X and the final prediction Y. The mutual information $I(X;Y)$ between the output of layer X and prediction Y is computed as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}, \tag{7}$$

where $p(x,y)$ is the joint probability distribution of X and Y, while $p(x)$ and $p(y)$ are the marginal distributions. A higher mutual information value indicates that the layer is more important for the final output and should therefore be assigned a higher bit-width. Once the mutual information is computed, an average bit-width B_{avg} is determined based on the available memory budget. Layers

Algorithm 1 Mixed-Precision Quantization

```
Compute mutual information $I(X_i; Y)$
Initialize bit-width configuration $\mathbf{b}_0$ based on $I(X_i; Y)$ and memory constraint
$\mathcal{D} \leftarrow \{(\mathbf{b}_0, P(\mathbf{b}_0), M(\mathbf{b}_0))\}$
while not converged do
    Train GP model on $\mathcal{D}$
    $\mathbf{b}_{t+1} \leftarrow \arg\max_{\mathbf{b}} \alpha(\mathbf{b})$
    Apply $\mathbf{b}_{t+1}$ to pruned model and fine-tune
    Measure $P(\mathbf{b}_{t+1})$, $M(\mathbf{b}_{t+1})$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{b}_{t+1}, P(\mathbf{b}_{t+1}), M(\mathbf{b}_{t+1}))\}$
end while
```

with higher importance receive more bits, and the allocation is performed in discrete bit-widths (e.g., 4-bit, 8-bit), constrained by the total memory limit.

Although the initial bit-width configuration derived from mutual information offers a reasonable starting point for fine-tuning, the complex interactions between layers, particularly in LLMs, mean that the importance of individual layers may shift after fine-tuning. As a result, the initial bit-width assignment might not represent the optimal configuration. To further refine the precision configuration, we employ Bayesian optimization.

The objective of Bayesian optimization is to maximize model performance while minimizing memory usage. Let $\mathbf{b} = [B_1, B_2, \ldots, B_L]$ represent the bit-width configuration across L layers. The optimization problem is formulated as:

$$\mathbf{b}_{\text{opt}} = \arg\max_{\mathbf{b}} \ \alpha(\mathbf{b}), \tag{8}$$

where $\alpha(\mathbf{b})$ is an acquisition function that balances exploration (of less well-understood configurations) and exploitation (of known promising configurations). The memory usage $M(\mathbf{b})$ is constrained by M_{max}, the total available memory.

The process starts by initializing a dataset $\mathcal{D}$ with the initial bit-width configuration $\mathbf{b}_0$, along with its corresponding performance $P(\mathbf{b}_0)$ and memory usage $M(\mathbf{b}_0)$. A Gaussian Process (GP) model is then trained on the data to predict model performance and the uncertainty for new configurations. Based on this model, the acquisition function $\alpha(\mathbf{b})$ is used to select the next bit-width configuration to evaluate.

Once a new configuration $\mathbf{b}_{t+1}$ is selected, it is applied to the pruned model, fine-tuned, and its performance $P(\mathbf{b}_{t+1})$ and memory usage $M(\mathbf{b}_{t+1})$ are measured. These results are then added to the dataset $\mathcal{D}$, and the GP model is updated with the new data. This iterative process continues until a stopping criterion is met, such as convergence or a maximum number of iterations. Over time, this method refines the bit-width configuration to achieve an optimal balance between model performance and memory efficiency.

3.3 Performance Recovery

After the steps of structured pruning and mixed-precision quantization, significant memory savings are achieved. However, model performance typically needs to be restored through fine-tuning. Full-parameter fine-tuning is often impractical due to the large memory footprint it requires, but our compression technique makes full model fine-tuning feasible by reducing both memory and computational costs.

In addition to traditional full-parameter fine-tuning, efficient fine-tuning techniques such as LoRA (Low-Rank Adaptation) (Hu et al., 2021) have proven especially effective, particularly in scenarios with limited data. LoRA significantly reduces the number of trainable parameters by freezing the original weight matrix $\mathbf{W}_0$ and only updating the low-rank approximation of the weight matrix, represented as $\Delta\mathbf{W} = \mathbf{AB}$, where $\mathbf{A} \in \mathbb{R}^{d\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times d}$. Here, r (the rank) is much smaller than the original dimension d, leading to a substantial reduction in the number of trainable parameters.

The forward computation in this approach can be written as:

$$\mathbf{Y} = \mathbf{W}_0\mathbf{X} + \Delta\mathbf{W}\mathbf{X} = \mathbf{W}_0\mathbf{X} + \mathbf{ABX}, \tag{9}$$

There are also LoRA-like methods specifically designed for quantized models, such as QLoRA (Dettmers et al., 2023) and LoftQ (Li et al., 2023). LoftQ iteratively updates the low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ such that the quantized matrix $\mathbf{Q} + \mathbf{AB}$ approximates the original full-precision matrix $\mathbf{W}$ during fine-tuning. The objective is defined as:

$$\min_{\mathbf{A},\mathbf{B}} \|\mathbf{W} - (\mathbf{Q} + \mathbf{AB})\|^2. \tag{10}$$

where $\mathbf{Q}$ is the quantized matrix.

By combining structured pruning, mixed-precision quantization, and performance recovery techniques, QPruner is able to achieve robust adaptability with minimal computational overhead.

Table 1: Zero-shot performance and peak memory usage on LLaMA-7B and Vicuna-7B with varying pruning rates. LLM-Pruner represents the currently widely used half-precision model. The performance is reported in percentage (%), and the memory usage is in gigabytes (GB).

		Method	BoolQ	PIQA	HellS	WinoG	ARC-e	ARC-c	OBQA	Memory (GB)
LLaMA-7B	Rate = 0%	w/o tuning	73.09	78.35	72.98	67.09	67.42	41.38	42.40	-
	Rate = 20%	LLM-Pruner	63.30	76.82	68.68	**63.38**	63.76	37.11	40.60	35.06
		QPruner^1	67.77	76.55	68.03	61.80	64.06	38.65	40.00	21.78
		QPruner^2	68.60	76.79	68.43	62.78	65.50	38.74	40.40	23.05
		QPruner^3	**69.11**	**77.23**	**68.80**	63.17	**66.16**	**39.20**	**41.00**	23.32
	Rate = 30%	LLM-Pruner	62.45	74.37	**63.14**	**61.96**	**59.22**	33.70	**39.60**	31.38
		QPruner^1	58.96	71.22	58.10	58.88	52.19	32.34	38.40	20.12
		QPruner^2	62.20	72.88	60.64	60.50	55.61	33.56	38.40	22.87
		QPruner^3	**66.50**	**74.43**	61.14	61.40	58.12	**34.47**	39.20	22.15
	Rate = 50%	LLM-Pruner	43.76	68.88	44.85	50.99	45.20	28.75	34.60	23.89
		QPruner^1	45.14	68.34	44.39	52.96	43.86	29.01	35.80	15.47
		QPruner^2	47.08	68.85	45.53	53.65	44.31	29.36	36.20	16.85
		QPruner^3	**48.37**	**69.20**	**45.19**	**54.45**	**45.28**	**29.70**	**36.40**	16.65
Vicuna-7B	Rate = 0%	w/o tuning	75.69	77.75	71.06	67.80	69.07	40.78	42.20	-
	Rate = 20%	LLM-Pruner	57.77	77.56	67.16	63.14	67.30	37.71	40.40	35.25
		QPruner^1	57.95	76.82	66.42	62.51	66.62	37.37	40.60	21.65
		QPruner^2	59.70	77.20	66.31	62.66	67.12	37.48	40.80	22.95
		QPruner^3	**59.85**	**77.59**	**67.31**	**63.20**	**67.84**	**37.85**	**41.20**	23.10
	Rate = 30%	LLM-Pruner	**58.81**	74.37	60.70	**60.62**	59.01	33.79	38.80	31.83
		QPruner^1	53.85	74.76	60.65	60.06	59.72	34.30	38.20	19.95
		QPruner^2	55.64	75.07	61.65	60.31	59.54	34.47	38.60	21.65
		QPruner^3	57.23	**75.90**	**62.00**	60.37	**60.81**	**34.79**	**39.40**	21.80
	Rate = 50%	LLM-Pruner	59.51	66.87	43.18	52.01	48.40	26.45	34.00	24.55
		QPruner^1	59.51	67.90	43.30	50.83	48.82	27.49	34.60	14.50
		QPruner^2	61.31	68.56	44.54	53.02	49.50	28.13	35.40	15.90
		QPruner^3	**61.56**	**68.80**	**43.72**	**53.39**	**49.66**	**27.98**	**35.80**	15.35

4 Experiments

LLMs and Benchmarks. To demonstrate how QPruner performes on different model, we test it on three open source large language models: LLaMA-7B[1] (Touvron et al., 2023), LLaMA-13B[2] (Touvron et al., 2023) and Vicuna-7B[3] (Zheng et al., 2024). We conduct these LLMs on zero-shot classification tests for commonsense reasoning datasets, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), ARC-challenge (Clark et al., 2018), and OpenbookQA (Mihaylov et al., 2018).

Software and hardware configuration. We utilize the following configurations: *PyTorch* version 2.1.2, *BitsandBytes* library version 0.43.1, *Transformers* library version 4.41.0, *PEFT (Parameter-Efficient Fine-Tuning)* library version 0.11.1, *Optuna* library version 3.6.1, *CUDA* version 12.4, *GPU:* NVIDIA L20 GPU with 48GB of memory.

Implementation Details. The pruning method follows LLM-Pruner (Ma et al., 2023), and the dataset uses 50k publicly available samples from the Alpaca (Taori et al., 2023). All experiments were conducted with a LoRA matrix rank of 8, and LoftQ initialization with one iteration. We utilized BitsandBytes for quantization configuration, for memory considerations, we keep the number of 8-bit layers below 25%. For 4-bit quantization, we employed NF4 (Dettmers et al., 2024), and since 2-bit quantization does not reduce memory usage, each layer's quantization configuration only considered 4-bit and 8-bit options.

Hyperparameters. In the optimization of the pruned LLaMA-7B model, a comprehensive hyperparameter configuration was employed to ensure an optimal balance between model performance and computational efficiency. The model was fine-

[1] https://huggingface.co/baffo32/decapoda-research-llama-7B-hf

[2] https://huggingface.co/yahma/llama-13b-hf

[3] https://huggingface.co/lmsys/vicuna-7b-v1.5

Table 2: Performance comparison (%) of ablation studies on seven tasks at 20% pruning rate on LLaMA-7B. It appears that QPruner captures potential resource allocations without relying on other settings.

Benchmark	**Dtype of 4-bit**		**Adapter Initialization Method**			**Adapter Iteration Count**			**Importance Estimation**	
	NF4	**FP4**	**LoftQ**	**Gaussian**	**PiSSA**	**iter=1**	**iter=2**	**iter=4**	**Element^1**	**Element^2**
ARC-e	**65.49**	62.84	**65.49**	64.77	64.44	**65.49**	64.31	64.18	**65.49**	62.50
ARC-c	**38.99**	36.77	**38.99**	**38.99**	38.40	**38.99**	38.05	38.14	**38.99**	37.80
WinoGrande	61.40	**63.22**	61.40	**61.96**	61.48	**61.40**	60.46	60.69	**61.40**	59.43
OBQA	**40.20**	39.80	40.20	39.00	**40.40**	**40.20**	39.40	39.60	**40.20**	38.60
BoolQ	**67.22**	66.48	67.22	64.43	**68.20**	67.22	**67.55**	66.85	**67.22**	65.44
PIQA	**76.82**	**76.82**	**76.82**	76.44	76.39	**76.82**	76.44	76.55	**76.82**	76.39
HellaSwag	**67.97**	67.88	67.97	67.80	**68.01**	**67.97**	**67.97**	67.93	**67.97**	66.93

tuned with a learning rate of 3×10^{-4}, utilizing a batch size of 8, further divided into micro batches of 4 to manage memory constraints effectively. Sequences were standardized to a maximum length of 256 tokens, and a dropout of 0.05 was applied specifically to the LoRA layers targeting projections such as query, key, value, and output, alongside gate, down, and up projections. Quantization was dynamically applied at 4-bit and 8-bit levels according to layer requirements to optimize memory use without compromising computational accuracy. The training employed the paged AdamW optimizer with 32-bit precision, enhancing stability and efficiency. These settings were methodically tested and optimized through the Optuna framework to ensure robust model performance and resource utilization.

4.1 Main Results

In this section, we present experimental results to demonstrate the capability of our proposed QPruner framework in balancing performance while reducing memory usage through integrating quantization and structured pruning. Through further iterative optimization, it can even achieve better performance than high-precision models. Although pruning methods are very important, the pruning method itself is not our focus; therefore, we adopt the popular LLM-Pruner (Ma et al., 2023) as our baseline, which is a widely used structured pruning method that directly removes weights.

We evaluate the model performance and peak memory usage of LLM-Pruner and QPruner under different pruning rates. Due to the lack of specific test prompts in the LLaMA paper, we utilize open-source prompts provided by Gao et al. (2023) for benchmarking. Results for the LLaMA-7B and Vicuna-7B models are shown in Table 1. Although our method is expected to have greater advantages on larger models (e.g., 70B parameters or more), due to hardware limitations, we focus only on models within 13B parameters.

In our experiments, **QPruner^1** denotes the use of uniform quantization across all layers, **QPruner^2** represents the mixed-precision configuration based on mutual information, and **QPruner^3** refers to the mixed-precision quantization after further optimization using Bayesian methods based on QPruner^2. Theoretically, full-parameter fine-tuning would perform better than PEFT methods; however, it performs poorly on the Alpaca dataset commonly used in model compression. If we perform individual training according to each benchmark, only the pruned models after quantization can be fully fine-tuned, which is an advantage of our framework, but this would lead to unfair comparisons. Therefore, for unquantized models, we use LoRA (Hu et al., 2021) fine-tuning, and for quantized models, we use LoftQ (Li et al., 2023) fine-tuning.

From Table 1, we observe that our method demonstrates more significant advantages at higher pruning rates. For instance, at a pruning rate of 50% on the LLaMA-7B model, **QPruner^3** outperforms LLM-Pruner by achieving a higher accuracy on the BoolQ dataset (48.37% vs. 43.76%) while reducing memory usage from 23.89 GB to 16.65 GB—a reduction of approximately 30%. This highlights the effectiveness of our framework in maintaining or even improving performance under aggressive compression.

These results demonstrate that our QPruner framework effectively balances memory efficiency and model accuracy by integrating quantization with structured pruning. By employing finer-grained quantization strategies and a combined performance recovery phase, we mitigate the detrimental effects that pruning and quantization individually impose on LLMs. This integration not only reduces memory consumption but can also enhance model performance, especially at higher pruning rates.

4.2 Ablation Study

We conducted ablation experiments using LLaMA-7B with a 20% pruning rate, based on results obtained by **QPruner**[3]. All results are presented in Table 2. We tested different quantization data types (NF4, FP4), LoRA matrix initialization methods (Gaussian, PiSSA (Meng, 2024), LoftQ), varying iteration counts in LoftQ (more iterations represent better error fitting), and different importance estimation methods.

Our experiments show that the choice of quantization data type slightly affects performance, but our method is effective across different types. Similarly, different LoRA initialization methods yield comparable results, indicating robustness to initialization strategies. Interestingly, increasing the number of iterations in LoftQ does not necessarily improve performance, suggesting that fitting residuals with low-rank matrices may not always be beneficial. Finally, using first-order Taylor approximations for importance estimation outperforms second-order ones, highlighting the complexity of LLMs and the limitations of higher-order approximations.

Additional experiments on different Bayesian optimization iteration counts and resource consumption are provided in Appendix **??**. The Pareto frontier demonstrates that more iterations can lead to better configurations, albeit at increased computational cost.

5 Related Work

5.1 Efficient Compression of LLMs

LLM-Pruner (Ma et al., 2023) uses structured pruning to eliminate non-essential interconnected structures by leveraging gradient information. This technique enables compressed models to maintain good performance across multiple tasks with basic fine-tuning. Santacroce et al. (2023) proposes Globally Unique Movement (GUM), a novel pruning technique focusing on the sensitivity and uniqueness of LLMs' network components. GUM prunes neurons that uniquely contribute to the model output and are sensitive to loss changes, thus preserving high accuracy. This method optimizes the trade-off between information retention and computational efficiency. Quantization-Aware Training (QAT) combines quantization with full model fine-tuning to adapt models for downstream tasks (Peri et al., 2020; Liu et al., 2023). Although QAT is effective, it requires substantial computational resources, such as gradient calculations and optimization states, and it complicates the gradient computation for quantized weights. However, by leveraging LoRA, these challenges can be bypassed during task adaptation. Post-Training Quantization (PTQ) frameworks, such as GPTQ and SmoothQuant (Frantar et al., 2022; Xiao et al., 2023), use a small subset of training data to calibrate high-precision models, enabling the generation of task-specific quantized models without the need for gradient backpropagation. This makes PTQ more cost-efficient than QAT, although it generally results in lower accuracy. Xiao et al. (2023) proposed SmoothQuant, a post-training quantization framework that employs a mixed-precision strategy to calibrate large language models, enabling accurate and efficient deployment without the need for retraining.

5.2 Parameter Efficient Fine-Tuning

LLM-Adapters (Hu et al., 2023) integrate small adapters with few extra parameters into LLMs for efficient fine-tuning, allowing smaller models to perform as well as larger ones on specific tasks. Unlike the serial approach of adapters, low-rank adaptation (LoRA) (Hu et al., 2021) uses a parallel method to insert trainable rank decomposition matrices into each layer of the model's architecture. LoRA adds trainable matrices to each layer while keeping the pre-trained weights unchanged, reducing the number of trainable parameters and making model adaptation faster and less resource-intensive. QLoRA (Dettmers et al., 2024) combines low-rank adapters and quantized 4-bit weights for efficient LLM fine-tuning, significantly reducing GPU memory requirements while achieving performance comparable to full 16-bit fine-tuning. LoftQ (Li et al., 2023) applies quantization and low-rank approximation alternately to achieve a good initialization for LoRA fine-tuning, mitigating the discrepancy between quantized and pre-trained weights, and enabling efficient fine-tuning of quantized models, particularly in challenging low-bit regimes.

6 Conclusion

We propose QPruner, an innovative framework that combines structured pruning and quantization for efficient model compression. Given that structured pruning and quantization typically require performance recovery steps, integrating them provides a more holistic approach to mitigating the errors introduced by both techniques while further compressing the model. To address the uneven importance distribution across layers and precision loss caused by pruning and quantization, we adopt a fine-grained method to preserve the capacity of critical layers, enhancing their performance further during the fine-tuning process. After pruning, we first allocate mixed-precision quantization based on task relevance, followed by Bayesian optimization to iteratively refine decisions and probabilistically select the optimal quantization configuration. Experimental results demonstrate that QPruner significantly outperforms baseline models in terms of memory efficiency while achieving superior accuracy across multiple NLP benchmarks. By striking a balance between efficiency and performance, shows that QPruner is a powerful solution for deploying LLM in resource-limited environments.

Limitation

One of the current limitations of QPruner is the significant precision loss caused by structured pruning, which still impacts the overall model performance. In future work, we aim to further optimize the pruning process to minimize this precision degradation. Additionally, the use of Bayesian optimization requires real data to guide the process, which can be time-consuming. While this method improves quantization configurations, the iterative nature of Bayesian optimization introduces additional computational overhead that may not be ideal for all deployment scenarios.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10865–10873.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Xiang Deng, Vasilisa Bashlovkina, Feng Han, Simon Baumgartner, and Michael Bendersky. 2023. What do llms know about financial markets? a case study on reddit market sentiment analysis. In *Companion Proceedings of the ACM Web Conference 2023*, pages 107–110.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Peter I Frazier. 2018. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*, pages 255–278. Informs.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276.

Janghwan Lee, Minsoo Kim, Seungcheol Baek, Seok Hwang, Wonyong Sung, and Jungwook Choi. 2023. Enhancing computation efficiency in large language models through weight and activation quantization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14726–14739.

Yixiao Li, Yifan Yu, Chen Liang, Nikos Karampatziakis, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Loftq: Lora-fine-tuning-aware quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Meng. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Zeping Min and Jinbo Wang. 2023. Exploring the integration of large language models into automatic speech recognition systems: An empirical study. In *International Conference on Neural Information Processing*, pages 69–84. Springer.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

Dheeraj Peri, Jhalak Patel, and Josh Park. 2020. Deploying quantization-aware trained networks using tensorrt. In *GPU Technology Conference*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. 2023. What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*.

Shoetsu Sato, Jin Sakuma, Naoki Yoshinaga, Masashi Toyoda, and Masaru Kitsuregawa. 2020. Vocabulary adaptation for domain adaptation in neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4269–4279.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Shicheng Tan, Weng Lam Tam, Yuanchun Wang, Wenwen Gong, Shu Zhao, Peng Zhang, and Jie Tang. 2023. Gkd: A general knowledge distillation framework for large-scale pre-trained language model. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 134–148.

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`.

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

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Biao Zhang, Barry Haddow, and Alexandra Birch. 2023a. Prompting large language model for machine translation: A case study. In *International Conference on Machine Learning*, pages 41092–41110. PMLR.

Boyu Zhang, Hongyang Yang, Tianyu Zhou, Muhammad Ali Babar, and Xiao-Yang Liu. 2023b. Enhancing financial sentiment analysis via retrieval augmented large language models. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 349–356.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.