ATOM: LOW-BIT QUANTIZATION FOR EFFICIENT AND ACCURATE LLM SERVING

Yilong Zhao [1 2 *] **Chien-Yu Lin** [2] **Kan Zhu** [2] **Zihao Ye** [2] **Lequn Chen** [2] **Size Zheng** [2 3 *]
Luis Ceze [2 4] **Arvind Krishnamurthy** [2] **Tianqi Chen** [4 5] **Baris Kasikci** [2]

ABSTRACT

The growing demand for Large Language Models (LLMs) in applications such as content generation, intelligent chatbots, and sentiment analysis poses considerable challenges for LLM service providers. To efficiently use GPU resources and boost throughput, batching multiple requests has emerged as a popular paradigm; to further speed up batching, LLM quantization techniques reduce memory consumption and increase computing capacity. However, prevalent quantization schemes (e.g., 8-bit weight-activation quantization) cannot fully leverage the capabilities of modern GPUs, such as 4-bit integer operators, resulting in sub-optimal performance.

To maximize LLMs' serving throughput, we introduce Atom, a low-bit quantization method that achieves high throughput improvements with negligible accuracy loss. Atom significantly boosts serving throughput by using low-bit operators and considerably reduces memory consumption via low-bit quantization. It attains high accuracy by applying a novel mixed-precision and fine-grained quantization process. We evaluate Atom on 4-bit weight-activation quantization in the serving context. Atom improves end-to-end throughput (token/s) by up to $7.73\times$ compared to the FP16 and by $2.53\times$ compared to INT8 quantization, while maintaining the same latency target.

1 INTRODUCTION

Large Language Models (LLMs) are increasingly being integrated into our work routines and daily lives, where we use them for summarization, code completion, and decision-making. Studies report that ChatGPT has over 100 million users, with more than 1 billion website accesses per month (Duarte, 2023). Furthermore, the size and capabilities of LLMs continue to grow to accommodate a broader range of tasks. The high inference demand and model complexity have significantly increased the operational costs, i.e., compute/memory and energy, for LLM service providers to near $1 million daily (Elimian, 2023).

Unsurprisingly, optimizing LLM serving is becoming a pressing concern. Most efforts have focused on improving LLM serving throughput, which is typically achieved by batching requests from various users (Yu et al., 2022; Chen, 2023; Kwon et al., 2023). Batching multiple requests increases compute intensity and amortizes the cost of loading weight matrices, thereby improving throughput. Prior work has explored LLM quantization techniques to further improve batching efficiency. These techniques employ smaller data types to replace 16-bit floating point (FP16) values, thereby reducing memory consumption and accelerating computation (Lin et al., 2023; Xiao et al., 2023).

However, current quantization schemes do not leverage the full extent of capabilities provided by emerging efficient low-bit hardware support (e.g., Nvidia Ampere (Abdelkhalik et al., 2022) and Qualcomm Hexagon (Wikipedia contributors, 2023)). For instance, several prior approaches have explored weight-only quantization (Lin et al., 2023; Frantar et al., 2023). In these quantization schemes, weights are quantized to a low-bit representation (e.g., INT3), whereas activations remain in a floating point representation (e.g., FP16). Consequently, weights must be dequantized to the appropriate floating point representation (e.g., FP16) before being multiplied with activations using floating point representation. Therefore, even though weight-only quantization reduces memory consumption, it still requires costly floating-point arithmetic, which is inefficient, especially for large batch sizes.

Another prominent quantization scheme is weight-activation quantization, where both weights and activations are quantized to low-bit representations. In this scheme, weights and activations can be directly multiplied using low-precision

[*]Work done at UW. [1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China [2]School of Computer Science & Engineering, University of Washington, Seattle, United States [3]School of Computer Science, Peking University, Beijing, China [4]OctoAI [5]School of Computer Science, Carnegie Mellon University, Pittsburgh, United States. Correspondence to: Yilong Zhao <zhaoyilong217@sjtu.edu.cn>.

Proceedings of the 7^{th} MLSys Conference, Santa Clara, CA, USA, 2024.

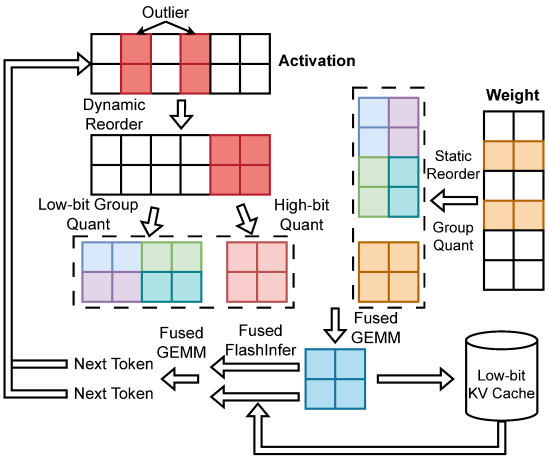


Figure 1. Overview of Atom's design. For activation matrices, we dynamically reorder the channels to pick out the outliers. Then, we apply low-bit group quantization to the normal values while using high-bit precision for outliers. For weight matrices, the quantization process can be done statically. We perform fused GEMM and fused FlashInfer (Ye et al., 2024) to boost throughput. We also adopt a quantized KV-cache to reduce memory movement.

arithmetic units. This quantization approach has greater potential to achieve higher inference throughput than weight-only quantization due to the efficient low-bit hardware support. For example, A100 GPUs can reach 1248 TOPS of INT4 and 624 TOPS of INT8 as opposed to only 312 TFLOPS for FP16 with Tensor Cores (NVIDIA, a). Prior works such as LLM.INT8() (Dettmers et al., 2022) and SmoothQuant (Xiao et al., 2023) explored INT8 weight-activation quantization and achieved near no accuracy loss. However, INT8 quantization still cannot utilize lower bit arithmetic such as INT4 Tensor Cores (NVIDIA, b). In addition, INT8 quantization remains sub-optimal for reducing the large memory consumption in LLM serving, where both model parameters and batched KV-cache consume large memory (Sheng et al., 2023; Zhang et al., 2023). For lower-bit weight-activation quantization, recent works such as OmniQuant (Shao et al., 2023) and QLLM (Liu et al., 2023a) have proposed to quantize LLMs down to 4-bit. However, their techniques still show a significant perplexity increase compared to the FP16 baseline as shown in Figure 2. Therefore, determining how to accurately quantize LLMs into low-bit representations while maintaining hardware efficiency remains an open area of research.

In this work, we introduce Atom, an accurate low-bit weight-activation quantization for LLMs that efficiently use modern hardware. To maintain accuracy, Atom incorporates three key quantization designs: (1) It adopts mixed-precision quantization, which retains a small but salient number of activations and weights in high precision to preserve accuracy. (2) It employs fine-grained group quantization on both weights and activations, which naturally reduces quantization errors. (3) Instead of pre-calculating quantization parameters for activations, Atom dynamically quantizes activations to best capture the distribution of each input.

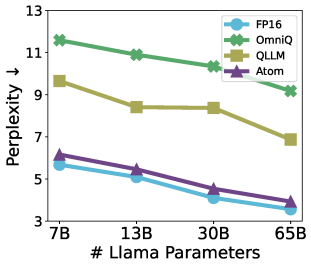


Figure 2. WikiText2 perplexity on Llama models with different 4-bit weight-activation quantization mechanisms. Atom maintains perplexity results close to the FP16 baseline across all model sizes.

Although these quantization optimizations can improve quantization accuracy, they may not utilize the underlying hardware efficiently without a bespoke design. For example, the mixed-precision technique could lead to irregular memory accesses and performance slowdown (Guo et al., 2023); matrix multiplications with group quantization are not well-supported in kernel libraries; and dynamic quantization of activations incurs extra computation (Xiao et al., 2023). To ensure high hardware efficiency and minimize quantization overheads, Atom: (1) reorders activations and weights to maintain regular memory accesses for mixed-precision operations, (2) fuses quantization and reordering operations into existing operators to mitigate the overheads, (3) further quantizes outliers into 8-bit to keep a balance between accuracy and efficiency and (4) quantizes the KV-cache into low-bit representations to reduce memory movement. We illustrate Atom's quantization workflow in Figure 1.

To validate Atom's feasibility, we integrate it into an end-to-end serving framework (Chen et al., 2023). For our special matrix multiplications with mixed-precision and group quantization, we implement customized CUDA kernels that utilize low-bit tensor cores. Experiments on popular datasets show that Atom has negligible accuracy loss (1.4% average zero-shot accuracy drop, 0.3 WikiText2 perplexity increase for Llama-65B) when quantizing models to 4-bit (for both weights and activations), while prior works suffer larger accuracy loss under the same precision (see Table 1).

When comparing end-to-end serving throughput to different precisions and quantization schemes, Atom improves throughput by up to 7.7×, 5.5×, and 2.5× relative to FP16, W4A16, and W8A8, respectively, while achieving similar latency (see Figure 10). These results show that Atom can accurately quantize LLMs into low-bit precision while achieving high serving throughput.

In summary, we contribute the following:

- A comprehensive performance analysis of LLM serv-

ing workloads that pinpoints the efficiency benefit of low-bit weight-activation quantization.
- Atom, an accurate low-bit weight-activation quantization algorithm that combines (1) mixed-precision with channel reordering, (2) fine-grained group quantization, (3) dynamic activation quantization to minimize quantization errors, and (4) KV-cache quantization.
- An integrated LLM serving framework for which we codesign an efficient inference workflow, implement low-bit GPU kernels and demonstrate practical end-to-end throughput and latency of Atom.
- A comprehensive evaluation of Atom, which shows that it improves LLM serving throughput by up to 7.7× with only a slight accuracy loss.

2 BACKGROUND

Quantization techniques use discrete low-bit values to approximate high-precision floating points. Since integers represent a uniform range, quantizing floating point values into integers is widespread due to simplicity and hardware efficiency (Jacob et al., 2017; Han et al., 2016). Typical quantization involves two steps: determining the quantization parameters (which consist of scale and zero point) and calculating the quantized tensor. For uniform asymmetric quantization, the scale s and zero point z are determined by (Nagel et al., 2021):

$$s = \frac{\max(X) - \min(X)}{2^n - 1} \cdot c, z = \lfloor \frac{-\min(X)}{s} \rceil, \quad (1)$$

where X is the input tensor, n is the quantization bit-width, and c is the clipping factor used to reduce the dynamic range of quantization to mitigate the effect of outlier values. The elements in quantized tensor can be calculated by:

$$\bar{X} = \text{clamp}(\lfloor \frac{X}{s} \rceil + z, 0, 2^n - 1).$$

We can further simplify this equation for symmetric quantization:

$$s = \frac{2 \cdot \max(|X|)}{2^n - 1} \cdot c$$

$$\bar{X} = \text{clamp}(\lfloor \frac{X}{s} \rceil, -2^{n-1}, 2^{n-1} - 1).$$

Quantization parameters s and z can be calculated either statically using calibration data or dynamically during inference time with runtime statistics. Thus, quantization approaches can be classified as *static* or *dynamic*.

For LLMs, we can apply quantization on both activation and weight matrices (weight-activation quantization) or just the latter (weight-only quantization). However, asymmetric weight-activation quantization can lead to additional calculations during matrix multiplication since:

$$W \cdot X = s_W(\bar{W} - z_W) \cdot s_x(\bar{X} - z_x),$$

where three additional cross-terms need to be calculated for using low-bit arithmetic units. Therefore, we apply symmetric quantization in this work for efficiency.

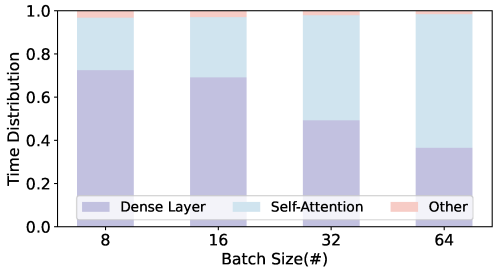


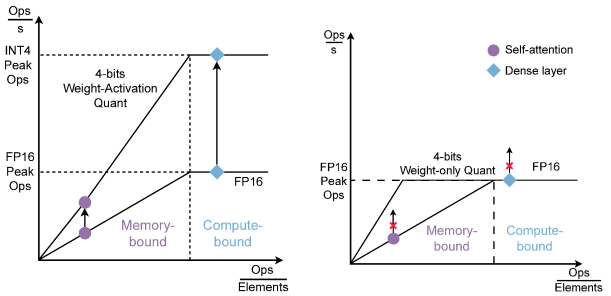
Figure 3. Runtime breakdown of Llama-7b inference with different batch sizes. The dense layer represents the batched K, Q, V generation, O projection, and MLP. The self-attention layer is implemented by FlashInfer (Ye et al., 2024) integrated with PageAttention (Kwon et al., 2023). Results indicate that the dense and self-attention layers together account for over 90% of the execution time, thereby constraining the throughput.

Different trade-offs between accuracy and efficiency can be achieved by quantization with different granularity: For **per-tensor** quantization, all the values in the tensor share one set of scale and zero-point (Nagel et al., 2021). For **per-channel (token)** quantization, we calculate scale and zero-point for a row or a column of the tensor (Xiao et al., 2023). We denote the channel as the last dimension of the input matrix. Each channel can be further divided into several sub-groups, and quantization is individually performed on each group, which is called **per-group** quantization (Lin et al., 2023). The finer the granularity, the more precise the quantization, but the higher the overhead. In this work, we adopt group quantization for higher accuracy with dedicated kernels to manage the overhead, as shown in § 4.2.

3 PERFORMANCE ANALYSIS OF LOW-BIT LLM SERVING

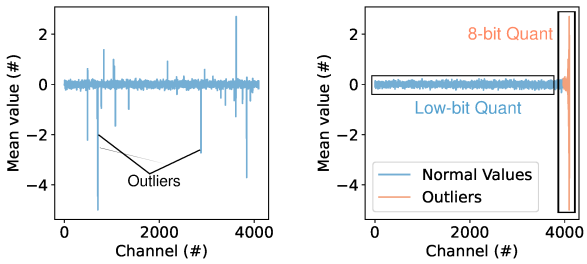
In this section, we first analyze the performance bottleneck of LLM inference in serving scenarios and then establish the importance of low-bit weight-activation quantization.

Due to high demand, LLM serving is throughput-oriented. However, the auto-regressive decode stage of LLM inference only takes one token as input and generates the next token, thus relying on matrix-vector multiplication (GEMV) (Agrawal et al., 2024). Since GEMV needs to load a large weight matrix while only performing a few multiplications, it is heavily memory-bound. It thus causes GPU under-utilization, which results in low compute inten-



(a) Weight-activation quantization (b) Weight-only quantization

Figure 4. A roofline model of different quantization approaches that characterizes operators by their arithmetic intensity, which is defined as Ops/Elements. At large batch sizes, the dense layer is compute-bound, which has a large arithmetic intensity, whereas self-attention consistently exhibits a lower arithmetic intensity.



(a) Activation mean values per channel. (b) Mean values after reordering.

Figure 5. Sampled value of an activation matrix from Llama-7b. (a) The activation matrix contains outlier channels, which result in large quantization errors. (b) Atom reorders these outlier channels to the end of the matrix and uses higher precision to quantize them while keeping regular memory access.

sity (computation-to-IO ratio) and, thereby, low throughput (Williams et al., 2009). To mitigate this problem, batching is widely used by combining the input from multiple requests to perform dense layer (K,Q,V generation, O projection, and MLP) matrix multiplications and increase compute intensity, therefore GPU utilization (Pope et al., 2022; Yu et al., 2022; Chen et al., 2023; Zhong et al., 2024).

To further exploit the batching effect and boost throughput, the input matrices of the dense layer of the decode and prefill stages are batched together to form larger matrices (Patel et al., 2023). Given large batch sizes, the dense layer ends up having compute-bound matrix-matrix multiplications (GEMM). However, though self-attention layers in the decode stage are also GEMV operations, they cannot benefit from batching. Since different inference requests do not share the KV-cache with different context histories, cross-request data cannot be batched for reuse, resulting in no efficiency benefit. Even with several optimizations such as FlashAttention (Dao et al., 2022) or Group Query Attention (Ainslie et al., 2023), the self-attention layers are still bounded by the large memory movement of KV-cache.

After applying the batching technique, we measure the time breakdown of different operators under different batch sizes. As Figure 3 shows, both the dense and self-attention layers act as bottlenecks to throughput, consuming over 90% of the processing time. Consequently, we employ quantization mechanisms to expedite both dense and self-attention layers.

We use the Roofline model (Williams et al., 2009) to evaluate the effect of different quantization approaches in serving scenarios. As Figure 4(a) shows, weight-activation quantization has higher dense layer compute throughput due to the efficient low-bit hardware arithmetic. It also increases the throughput of the self-attention layer by reducing the size of the KV-cache, thus decreasing memory movement. However, as Figure 4(b) shows, weight-only quantization fails to improve dense layer throughput since dequantization must be performed before matrix multiplications, yielding calculations still in the floating point format. On the other hand, weight-only quantization fails to quantize the KV-cache, yielding no benefit for self-attention layers. We further quantify the effect of different quantization techniques in Figures 11(a) and 11(b) in §5 with kernel profiling.

In summary, the low-bit weight-activation quantization is superior to weight-only quantization in terms of enhancing the throughput in the serving scenario because it accelerates both the dense and self-attention layers. In the following sections, we demonstrate how Atom delivers high throughput while still maintaining high accuracy with the low-bit weight-activation quantization.

4 Design

Low-bit precision enables efficient utilization of the underlying hardware, leading to increased throughput. However, it is challenging to maintain high accuracy with a low-bit representation. To quantize LLMs to extremely low-bit precision while keeping accuracy, we incorporate a suite of quantization mechanisms tailored to LLM characteristics. These mechanisms include mixed-precision quantization with channel reordering, fine-grained group quantization, and dynamic quantization. We demonstrate the accuracy gain thanks to these techniques with ablation study in Table 3. Atom also applies low-bit quantization on KV-cache, which further boosts the efficiency. The subsequent subsections delve into the specifics of each mechanism and its advantages, followed by a detailed description of the end-to-end workflow.

4.1 Mixed-precision quantization

Prior works observed that a key challenge of LLM quantization is the outlier phenomena in activations (Dettmers et al.,

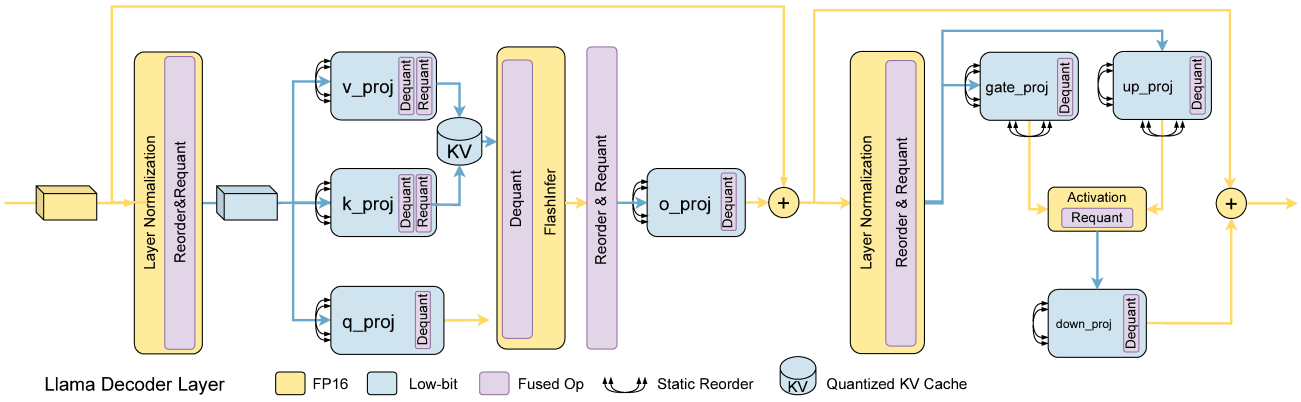


Figure 6. Overview of Atom workflow on Llama model family. Atom carefully manages the overhead of quantization operators by fusing them into existing operators. For the compute-bound operators, Atom utilizes efficient low-bit hardware support. For the memory-bound self-attention layer, Atom quantizes KV-cache to further enhance the throughput. We implement dedicated kernels for each fused operator.

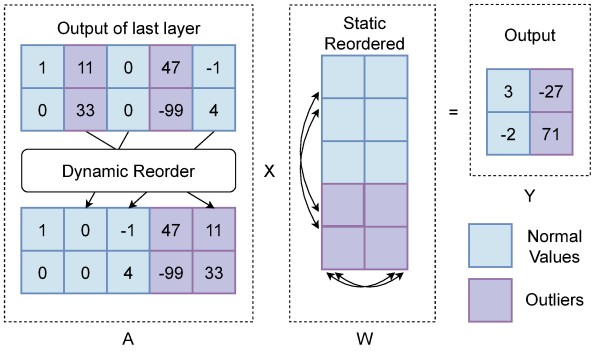


Figure 7. Atom dynamically reorders activation (A) to move the outlier channels to the end of the matrix, with the reorder indices determined in offline calibration. The weight matrix (W) is statically reordered to remain aligned with the corresponding activation channels, which guarantees the correctness of the output result.

2022; Lin et al., 2023). As Figure 5(a) shows, a few channels exhibit large magnitudes that are several orders greater than those of other channels, which are called outliers. The large dynamic range of these outliers can substantially increase the quantization error. Therefore, efficiently handling the outliers is crucial in low-bit quantization.

One intuitive way to effectively mitigate this challenge is to quantize outliers and normal values separately, into low and high bits, which is referred to as a *mixed-precision* method. As Figure 5(b) shows, after we remove the outliers, the remaining channels are much more uniform, which can be effectively expressed by low-bit values. Our results indicate that 8-bit representations, such as FP8 (Micikevicius et al., 2022) and INT8, are sufficient to express outliers (See Table 3). Since INT8 is widely supported by hardware implementations (e.g., NVIDIA Tensor Core (Abdelkhalik et al., 2022)), Atom applies INT8 quantization for outliers.

The primary concern with mixed-precision quantization is its irregular memory accesses (Dettmers et al., 2022; Guo et al., 2023), which leads to poor hardware efficiency. To apply mixed-precision quantization while maintaining regular memory access, Atom re-purposes the reordering technique introduced in RPTQ (Yuan et al., 2023), where the objective is to improve quantization accuracy. As Figure 7 shows, Atom reorders the scattered outlier channels of activations to the end of the matrix, which enables the efficient implementation of mixed-precision. To guarantee the equivalence of the computation result, the weight matrices need to be reordered with the corresponding reorder indices of activations. Since the outlier channels can be identified offline using calibration data (Dettmers et al., 2022), the reordering of weight matrices incurs a one-time cost. However, the reordering of activation matrices still needs to be performed online, which can be expensive. To mitigate this, Atom fuses the activation matrix reordering operators into prior operators, which significantly reduces the reordering overhead to less than 0.5% of runtime.

4.2 Fine-grained group quantization

Even if Atom quantizes outliers and normal values separately, the latter is still challenging to perform accurately due to the limited representation capability of 4-bit precision (Section 5.4). To further enhance accuracy, *group quantization* is widely adopted (Lin et al., 2023; Nagel et al., 2021), which divides the matrix into subgroups and performs quantization within each subgroup. For example, a group size of 128 implies that every contiguous sequence of 128 elements is treated as a single group, which is quantized independently.

Group quantization offers a trade-off between accuracy improvements and dequantization overheads, especially in weight-activation quantization. Prior works have not investigated how to efficiently incorporate group dequantization into the delicate GEMM pipeline, i.e., MMA

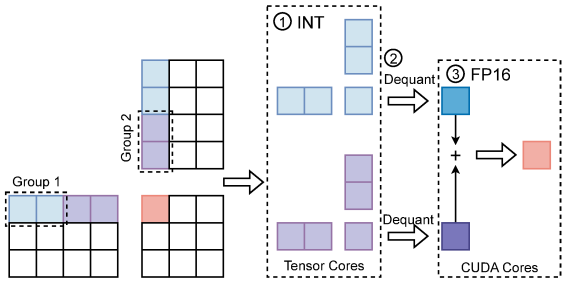


Figure 8. Overview of the fused GEMM operator. The multiplication of each group is first computed by units with efficient low-bit support, i.e., Tensor Cores (Step ①). The result is then dequantized and subsequently accumulated with typical FP16 units (Step ②, ③). Note that all operations are fused in a single pipeline.

pipeline (Thakkar et al., 2023). Atom proposes a fusion technique as shown in Figure 8, which contributes to an efficient GEMM kernel with practical speedup (See §5.3.1). Atom first calculates the matrix multiplication of the activation groups with the corresponding weight groups and obtains temporary results using efficient low-bit hardware, i.e. Tensor Cores (Step ①). Atom then adds multiple temporary results together to get the GEMM result. However, since Atom performs fine-grained quantization for each activation and weight group, each temporary result has different quantization parameters. Therefore, Atom first dequantizes all temporary results to the FP16 representation with CUDA Cores (Step ②) and then performs addition (Step ③). To manage the overhead, we fuse dequantization and summation into the GEMM kernel, to be specific, into the MMA pipeline. Therefore, the additional operations can be executed in place without extra memory movement and overlapped with the original MMA instructions. We demonstrate the efficiency of the fused GEMM operator in §5.3.1.

With a group size of 128 and a high precision channel size of 128, Atom has an effective bit of 4.25[1] on Llama-7b. The *effective bit* is defined as the average bits used for each element, including the quantization parameters. This metric is widely used in previous works on weight-only quantization (Frantar et al., 2023; Lin et al., 2023), mainly because it represents the actual compression ratio and, therefore, the speedup in the memory-bound setting. However, the main benefit of weight-activation quantization in serving scenarios is the computation efficiency of leveraging low-bit arithmetic units instead of the memory reduction. Therefore, we will not use this metric in the following discussions.

4.3 Dynamic quantization process

Although fine-grained quantization can better preserve the local variations inside each channel of activations, this advantage would diminish if we statically calculated the quantization parameters based on calibration data, as the actual input might have a different local distribution.

[1]With 4-bit for normal values, 8-bit for outliers, and 16-bit scale per group, the effective bit is calculated as $((4096-128)*4+128*8)/4096+16/128=4.25$.

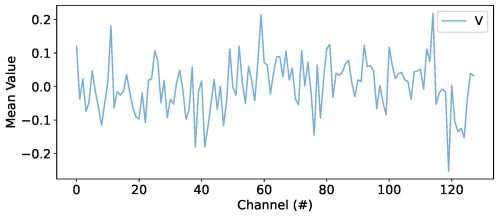


Figure 9. Sampled value of the V cache within a single attention head from Llama-7b. Compared with sampled activations shown in Figure 5(a), the V cache shows a much less dynamic range with fewer outlier channels, which is much easier to quantize.

Therefore, Atom adopts *dynamic quantization*, tailoring quantization parameters for each activation matrix during inference. To tame the overhead of dynamic quantization, we fuse quantization operations into the prior operator, akin to the implementation of ZeroQuant (Yao et al., 2022). Since the additional operator is element-wise (with a reduction and an element-wise division), the run time of the fused operator is still negligible compared to the time-consuming dense and self-attention layers, as Figure 3 shows.

However, asymmetric quantization can lead to significant run-time overhead due to considerable additional computation (as discussed in §2). To strike a balance between throughput and accuracy, we choose symmetric quantization with a carefully chosen clip threshold. We also incorporate GPTQ (Frantar et al., 2023) when quantizing the weight matrix since this is purely an offline process and offers an accuracy boost without sacrificing runtime efficiency.

4.4 KV-cache quantization

As described in §3, the self-attention layer in the decode stage is highly memory-bound. To mitigate this issue, Atom also applies low-bit quantization to the KV-cache. Atom loads the KV-cache in low-bit precision and directly dequantizes it before performing the FP16 calculation, which significantly boosts the throughput by large memory reduction. On the other hand, since the memory movement of asymmetric and symmetric quantized KV-cache are similar, they perform similarly on memory-bound self-attention layers. Therefore, Atom uses asymmetric quantization on KV-cache as it can provide accuracy benefits.

Compared with activation matrices, we argue that the KV-cache is more amenable to quantization. To perform self-attention, the Query vector of the incoming token is multiplied by the K cache. The result is normalized using Softmax and further multiplied with the V cache to obtain the output (Vaswani et al., 2023). Due to the normalization of Softmax, the quantization error of the K cache has less influence on the output. Furthermore, our profiling in Figure 9

indicates that the V cache exhibits the outlier phenomenon less frequently, rendering it more suitable for quantization. Therefore, Atom directly applies asymmetric low-bit quantization with the granularity of attention head and preserves high accuracy as shown in §5.4.

4.5 Implementation of quantization workflow

To demonstrate the feasibility of our design choices, we implement Atom on Llama models (Touvron et al., 2023a), as shown in Figure 6. To leverage the benefit of quantization, Atom manages the overhead of the additional operators by kernel fusion: Atom fuses quantization operators, including reordering, quantization, and dequantization, into existing operators. For the compute-bound dense layer, Atom utilizes the low-bit units to boost throughput. For the memory-bound self-attention layer, Atom fuses dequantization with a kernel library for LLM serving, FlashInfer (Ye et al., 2024), so that only low-bit values from KV-cache are loaded. Atom also incorporates PageAttention (Kwon et al., 2023) for efficient memory usage to enable large batch sizes.

5 Evaluation

We conduct a comprehensive evaluation of Atom's accuracy and efficiency. For accuracy, we evaluate Atom on widely used metrics, generation perplexity and zero-shot accuracy. For efficiency, we evaluate Atom from the bottom up, starting with per-kernel performance, followed by end-to-end throughput and latency. We also perform ablation studies to understand how different techniques affect Atom, which pinpoints the trade-off between the efficiency and accuracy of each design choice.

5.1 Quantization setup

Atom uses symmetric quantization on weights and activations while using asymmetric quantization on the KV-cache. We evaluate Atom using a group size of 128. To identify outlier channels, we use 128 randomly sampled sentences from WikiText2 (Merity et al., 2016) as calibration data, following prior works (Lee et al., 2023; Shao et al., 2023; Liu et al., 2023a). We select 128 channels with the highest square sum values as outlier channels and keep them in INT8. We then reorder activation and weight matrices according to the indices of outlier channels. After reordering, Atom adopts GPTQ (Frantar et al., 2023) for the quantization on weight matrices. For clipping, we use a grid search to find optimal clipping factors 0.9 and 0.85 for activation and weight quantization, respectively.

For the preprocessing of weight quantization and outlier identification, we run Atom on a single RTX Ada 6000 and quantize the model layer-by-layer. For large Llama-65B, Atom takes roughly 4 hours to complete the process.

5.2 Accuracy evaluation

Benchmarks. We evaluate Atom on popular open-sourced Llama (Touvron et al., 2023a) models. We focus on low-bit settings, INT4 and INT3 weight-activation quantization. We adopt commonly used metrics of model accuracy, perplexity, and zero-shot accuracy. For perplexity, we evaluate on WikiText2 (Merity et al., 2016), PTB (Marcus et al., 1994), and C4 (Raffel et al., 2020) datasets. For zero-shot tasks, we use lm-eval (Gao et al., 2021), based on which we evaluate Atom on PIQA (Bisk et al., 2019), ARC (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2019) tasks.

Baselines. We compare Atom to recently released post-training quantization techniques: SmoothQuant (Xiao et al., 2023), OmniQuant (Shao et al., 2023), and QLLM (Liu et al., 2023a). For SmoothQuant, we implement our own version as the official code does not support Llama models and only has W8A8 quantization. We conducted a grid search on the $alpha$ value defined in SmoothQuant and reported the best numbers for each benchmark. For OmniQuant, we use their pre-quantized weights for W4A4 evaluations and evaluate W3A3 by running their official code. To obtain the best W3A3 results for OmniQuant, we conduct a hyperparameter search and identify $lr = 1e^{-4}$ and $alpha = 0.75$ for their quantization process. We skip W3A3 OmniQuant on Llama-30B and Llama-65B due to the large resource requirement of its quantization process. For QLLM, we report the W4A4 numbers in their paper but do not evaluate W3A3 as their code was unavailable when we conducted experiments.

Zero-shot accuracy. Table 1 compares the zero-shot accuracy of six tasks between Atom and baselines on Llama models. Atom significantly outperforms the other weight-activation quantization methods. For W4A4, Atom shows only a 2.3%, 1.7%, 0.4% and 1.4% average accuracy loss for Llama at 7B, 13B, 30B and 65B sizes when compared to FP16. At the same time, previous works showed a 9.6% to 23.8% accuracy loss under the same settings.

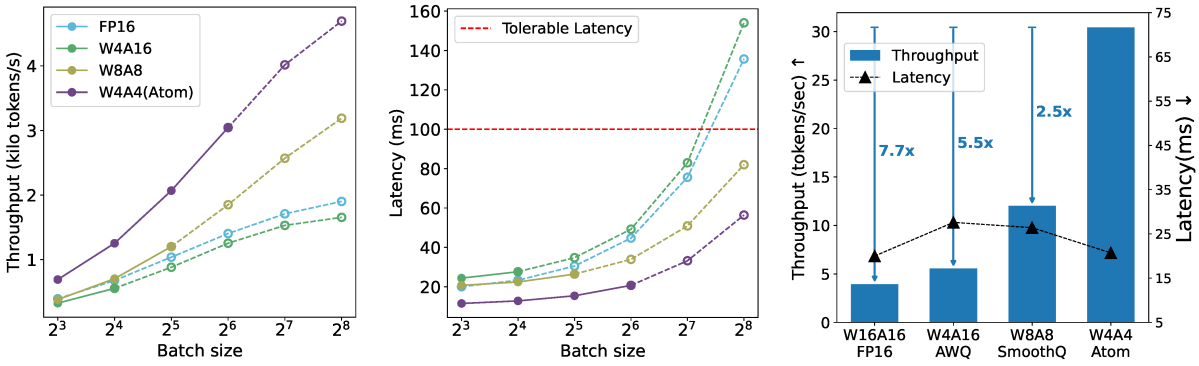
Perplexity. Table 2 reports perplexity results of Atom and baselines on Llama models. As the table shows, though recent methods such as OmniQuant and QLLM successfully reduce the perplexity of W4A4 to around 10, the accuracy loss is still significant. Atom further reduces the perplexity and achieves less than 0.4 perplexity increase on all three datasets with Llama-65b. For W3A3, Atom still largely maintains the perplexity, with an average 2.3 perplexity increase for Llama-65B. At the same time, existing works do not achieve acceptable perplexity. Note that Atom has less accuracy loss when quantizing larger models.

Table 1. **Zero-shot accuracy** of quantized Llama models on six common sense tasks.

Size	#Bits	Method	Zero-shot Accuracy ↑						
			PIQA	ARC-e	ARC-c	BoolQ	HellaSwag	Winogrande	Avg.
7B	FP16	-	77.37	52.53	41.38	73.12	72.99	66.85	64.04
	W4A4	SmoothQuant	63.11	40.03	31.57	58.47	43.38	52.80	48.23
		OmniQuant	66.15	45.20	31.14	63.51	56.44	53.43	52.65
		QLLM	68.77	45.20	31.14	-	57.43	56.67	51.84
		Atom	**76.28**	**52.10**	**38.99**	**69.79**	**69.81**	**63.69**	**61.78**
	W3A3	SmoothQuant	48.69	25.97	28.16	45.26	26.02	49.57	37.28
		OmniQuant	49.78	27.19	27.22	37.86	25.64	49.96	36.28
		Atom	65.56	41.41	30.72	61.77	53.19	55.56	51.37
13B	FP16	-	79.05	59.85	44.62	68.53	76.22	70.09	66.39
	W4A4	SmoothQuant	64.47	41.75	30.89	62.29	46.68	51.70	49.63
		OmniQuant	69.69	47.39	33.10	62.84	58.96	55.80	54.63
		QLLM	71.38	47.60	34.30	-	63.70	59.43	55.28
		Atom	**77.69**	**57.58**	**42.92**	**67.46**	**73.77**	**68.51**	**64.66**
	W3A3	SmoothQuant	47.99	26.30	27.65	46.91	25.65	49.64	37.36
		OmniQuant	50.22	26.77	27.82	37.83	25.77	51.07	36.58
		Atom	70.08	47.94	33.70	63.46	62.93	56.75	55.81
30B	FP16	-	80.20	58.92	45.31	68.38	79.23	72.69	67.46
	W4A4	SmoothQuant	59.30	36.74	28.58	59.97	34.84	49.96	44.90
		OmniQuant	71.21	49.45	34.47	65.33	64.65	59.19	57.38
		QLLM	73.83	50.67	38.40	-	67.91	58.56	57.87
		Atom	**78.73**	**58.92**	**45.82**	**68.47**	**77.40**	**73.09**	**67.07**
	W3A3	SmoothQuant	49.46	27.53	28.16	39.42	26.05	51.38	37.00
		Atom	72.47	49.54	37.80	65.75	66.99	60.14	58.78
65B	FP16	-	80.79	58.71	46.33	82.26	80.71	77.03	70.97
	W4A4	SmoothQuant	60.72	38.80	30.29	57.61	36.81	53.43	46.28
		OmniQuant	71.81	48.02	35.92	73.27	66.81	59.51	59.22
		QLLM	73.56	52.06	39.68	-	70.94	62.90	59.83
		Atom	**80.41**	**58.12**	**45.22**	**82.02**	**79.10**	**72.53**	**69.57**
	W3A3	SmoothQuant	49.56	26.64	29.10	42.97	26.05	51.14	37.58
		Atom	75.84	51.43	41.30	74.07	72.22	64.33	63.20

Table 2. **Perplexity** of quantized Llama models on WikiText2, PTB and C4 dataset.

Size	Bits	Method	Perplexity ↓			Size	Bits	Method	Perplexity ↓		
			WikiText2	PTB	C4				WikiText2	PTB	C4
7B	FP16	-	5.68	8.80	7.08	13B	FP16	-	5.09	8.07	6.61
	W4A4	SmoothQuant	22.62	40.69	31.21		W4A4	SmoothQuant	33.98	73.83	41.53
		OmniQuant	11.59	20.65	14.96			OmniQuant	10.90	18.03	13.78
		QLLM	9.65	-	12.29			QLLM	8.41	-	10.58
		Atom	**6.16**	**9.62**	**7.70**			Atom	**5.46**	**8.60**	**7.03**
	W3A3	SmoothQuant	2.7e4	3.5e4	2.6e4		W3A3	SmoothQuant	1.3e4	1.6e4	1.5e4
		OmniQuant	3.4e3	7.5e3	6.3e3			OmniQuant	7.2e3	1.6e4	1.3e4
		Atom	11.77	20.84	15.43			Atom	8.40	15.84	10.81
30B	FP16	-	4.10	7.30	5.98	65B	FP16	-	3.53	6.91	5.62
	W4A4	SmoothQuant	109.85	142.34	87.06		W4A4	SmoothQuant	88.89	278.76	283.80
		OmniQuant	10.34	14.91	12.49			OmniQuant	9.18	16.18	11.31
		QLLM	8.37	-	11.51			QLLM	6.87	-	8.98
		Atom	**4.54**	**7.69**	**6.35**			Atom	**3.89**	**7.22**	**5.92**
	W3A3	SmoothQuant	1.5e4	1.6e4	1.5e4		W3A3	SmoothQuant	6.6e8	3.7e8	4.4e8
		Atom	6.94	12.12	9.14			Atom	5.89	9.71	7.94



(a) End-to-end throughput (b) Decode latency per token (c) Comparison given fixed GPU memory

Figure 10. End-to-end evaluation of Atom. Solid lines are exact measurements, while dashed lines are estimations due to the limited memory capacity. (a) The number of generated tokens per second. (b) Average decode latency per token. Atom surpasses all other quantization methods for both throughput and latency. (c) Performance evaluated under a fixed amount of GPU memory. Note that Atom boosts the throughput by 2.5× more than W8A8 since it enables a larger batch size, which utilizes the batching effect.

5.3 Efficiency evaluation

To demonstrate the efficiency of Atom, we conduct experiments profiling both per-kernel and end-to-end performance. Since the highly efficient INT4 arithmetic is supported by NVIDIA GPUs, we evaluate Atom with W4A4 quantization on a 24GB RTX 4090 with CUDA 11.3.

5.3.1 *Kernel evaluation*

Matrix multiplication. We evaluate the fused GEMM operator implemented by Atom, as shown in Figure 11(a). We also implemented fused GEMM for 8-bit weight-activation quantization (W8A8) and 4-bit weight-only quantization (W4A16) following the existing work (Xiao et al., 2023; Lin et al., 2023) as baselines. For smaller batch sizes, GEMM is memory-bound; thus, weight-only quantization's memory reduction is effective. However, as the batch size increases, the efficiency of weight-only quantization diminishes in the compute-bound setting due to the expensive FP16 calculations. At the same time, 4-bit Atom outperforms all other approaches due to its hardware efficiency. At batch size 512, Atom's matrix-multiplication achieves 3.4× and 1.9× speedup over FP16 and INT8 kernels.

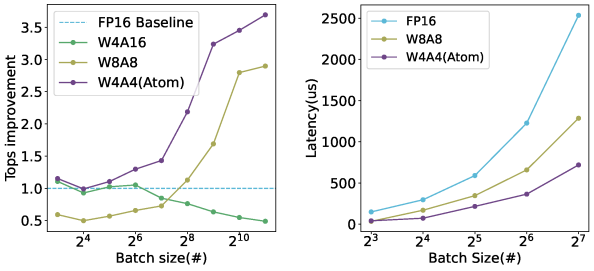
Self-attention. For the self-attention layer, we fuse different quantization methods into FlashInfer (Ye et al., 2024), which is a performant kernel library for LLMs serving. We also integrate PageAttention (Kwon et al., 2023) for efficient memory usage. We evaluate our implementation and show the results in Figure 11(b). The decrease in bits linearly reduces the memory usage of the KV-cache, therefore proportionally boosting the throughput in the memory-bound setting. At batch size 128, Atom achieves a 1.8× speedup over INT8 quantization and 3.5× over the FP16 baseline.

5.3.2 *End-to-end evaluation*

Serving setup. We integrate Atom into Punica, an LLM serving framework (Chen et al., 2023), to evaluate the performance in the end-to-end scenario. We also integrate W8A8 and W4A16 quantizations following previous works (Xiao et al., 2023; Lin et al., 2023) as baselines. To generate a representative workload, we use ShareGPT (HuggingFace, 2023) to collect the distribution of prefill and decode request length. We treat multi-round conversations as requests from multiple users. Specifically, we concatenate all previous prompts and responses and use them as the prompt for the new user request. We vary the batch size from 8 to 256, which represents the practical range in LLM serving[2]. All requests are served in a First-Come-First-Served manner. When a request is finished, we re-fill the on-the-fly batch with a new request following *continous batching* as introduced in Orca (Yu et al., 2022). Due to GPU memory limits, we only show the exact results on small batch sizes. When the memory requirement cannot be satisfied, we simulate the performance by reusing the KV-caches from a smaller batch size while preserving the data access pattern and amount of computation.

End-to-end throughput. We show the end-to-end throughput, i.e., generated tokens per second, in Figure 10(a). Solid lines represent exact evaluation results, while dashed lines represent our simulated results for the cases that exceed our GPU's memory capacity. As Figure 10(a) shows, Atom outperforms other quantization methods on all batch sizes. If we fix the available memory as shown in Figure 10(c), Atom can achieve larger batch sizes so that its throughput further surpasses all baselines while still meeting the latency

[2]With quantization, pipelining, and tensor parallelism to amortize weights, it is practical to deploy a 180B model with a 256 batch size in the serving scenario (Patel et al., 2023).



(a) Dense layer (b) Self-attention

Figure 11. Performance evaluation of different quantization approaches on Atom and baseline kernels. We set up the evaluation configuration aligned with the Llama-7b config and 1024 sequence length. Kernels are evaluated by NVBench (NVIDIA, 2024b).

target. Atom achieves 7.73× throughput compared to the FP16 baseline and 2.53× throughput compared to INT8 quantization using the same amount of memory. In contrast, weight-only quantization is bounded by FP16 computation capacity in dense layers and large memory movement of the KV-cache in the self-attention layer.

End-to-end latency. We measure the latency as the average decoding time of each token, without considering the queuing time. Atom significantly outperforms other quantization methods on every batch size. When we achieve the highest practical performance at batch size 64, our latency is lower than INT8 or FP16 implementations, even under batch size 8. Notably, even at batch size 256, our latency is still lower than 100 ms, which has been shown to be the effective reading speed of human eyes by a prior study (Trauzettel-Klosinski et al., 2012).

5.4 Ablation study of quantization techniques

In this subsection, we comprehensively evaluate the effectiveness of quantization techniques used in Atom, in terms of both accuracy and efficiency, to better illustrate our design choices and the trade-off between accuracy and efficiency.

5.4.1 Ablation study to evaluate accuracy

We examine the accuracy gain or loss of different quantization techniques used in Atom. We first use RTN and adopt per-channel quantization for weights and per-token quantization for activations, which is the standard quantization recipe (Xiao et al., 2023), to quantize the model to W4A4. We then apply other quantization techniques used in Atom, i.e., mixed-precision, quantizing outliers, group quantization, clipping, GPTQ, and KV-cache quantization, and examine the perplexity case by case. As shown in Table 3, keeping outlier channels in FP16 significantly reduces the perplexity. Further quantizing outliers into INT8 only results in a very minor 0.05 perplexity increase, which indicates mixed precision effectively addresses the outlier issue.

Table 3. Ablation study on different quantization techniques used in Atom. The model used in this table is Llama-7B.

Quantization method	WikiText2 PPL↓
FP16 baseline	5.68
W4A4 RTN	2315.52
+ Keeping 128 outliers in FP16	11.34 (2304.2↓)
+ Quantizing outliers to INT8	11.39 (0.05↑)
+ Group size 128	6.22 (5.17↓)
+ Clipping	6.13 (0.09↓)
+ GPTQ	6.04 (0.09↓)
+ Quantizing KV-cache to INT4	6.16 (0.12↑)

Besides, fine-grained group quantization brings another major perplexity reduction. Furthermore, using clipping and GPTQ lowers perplexity by 0.09 each. After all, quantizing KV-cache results in a slight 0.12 perplexity increase, which echoes our finding in Section 4.4.

5.4.2 Ablation study to evaluate efficiency

We then showcase the GEMM kernel throughput with different fused quantization techniques[3]. A pure INT4 GEMM implementation without any quantization operation achieves nearly 980 TOPS. Fusion of mixed precision, which keeps 128 channel calculations in INT8 Tensor Cores, leads to 8% overhead, with 900 TOPS throughput. Fine-grained group quantization contributes to the major overhead since it deeply affects the compute pipeline. The fusion of group dequantization decreases the performance to 770 TOPS. However, the fused GEMM kernel still outperforms the theoretical limit of INT8 throughput by nearly 18%.

Besides, to demonstrate the efficiency of channel reordering, we also conduct an ablation study on Atom and baseline. The baseline is implemented following the previous work (Dettmers et al., 2022), with matrix decomposition for mixed precision quantization. At the same time, Atom fuses quantization operators, including reordering and quantization, into existing operators. We evaluate batch sizes from 16 to 256 and measure the inference latency of a layer norm and a GEMM operation. Results show that Atom consistently outperforms the baseline from 25% to 35%.

6 Discussion

With innovations of model architectures like Mixture of Experts (MoE) (Jiang et al., 2024; Dai et al., 2024), State Space Models (SSMs) (Gu et al., 2022; Gu & Dao, 2023), and evolvement of hardware accelerators (e.g., NVIDIA Blackwell GPU (NVIDIA, 2024a)), it's important that Atom can be used for new models and hardware. In this section, we provide evaluations on more LLMs and data formats.

Generality on models. Atom's main techniques to achieve

[3]Kernel performance is profiled by NVBench (NVIDIA, 2024b) with the Llama-7b config and a batch size of 4096 on RTX 4090.

Table 4. WikiText2 perplexity for Llama-2 and Mixtral.

# Bits	Method	Llama2 7B	Llama2 13B	Llama2 70B	Mixtral 8x7B
FP16	-	5.47	4.88	3.32	3.84
W4A4	SmoothQuant	83.12	35.88	-	-
	OmniQuant	14.61	12.3	-	-
	Atom (INT)	**6.03**	**5.27**	**3.68**	**4.41**
	Atom (FP)	**6.14**	**5.35**	**3.78**	**4.50**

high accuracy are mixed precision for outliers and fine-grained quantization for normal values. We empirically find these are generalizable to newer transformer-based LLMs. In Table 4, we show the perplexity results of two relatively new LLMs, Llama-2 (Touvron et al., 2023b) and Mixtral (Jiang et al., 2024). To generalize on MoE models, Atom only needs to adapt to using different reorder indices for different experts' FFN[4]. As Table 4 shows, Atom still outperforms baselines and maintains high accuracy.

Generality on data formats. With the support for emerging data formats such as FP4 and MX (Liu et al., 2023b; Rouhani et al., 2023) on new hardware, we also evaluate the effectiveness of Atom in FP4. As shown in Table 4, Atom maintains a similar accuracy to INT4 when quantizing both weights and activations into FP4. We conclude that the representation capability between INT4 and FP4 is similar. Additionally, group quantization with the MX format is supported by NVIDIA Blackwell GPUs. We expect this hardware feature can mitigate the group quantization overhead of Atom as described in § 5.4.2.

7 Related Work

LLM serving. Various works have been explored to improve LLM serving throughput. (Pope et al., 2022) investigated the batching effect when scaling up LLMs. Orca (Yu et al., 2022) proposed *continuous batching* to improve GPU utilization by refilling the on-the-fly batch. vLLM (Kwon et al., 2023) utilized page tables to manage KV-cache, which significantly increases GPU memory utilization. FlexGen (Sheng et al., 2023) proposed an offload mechanism to support larger batches for high serving throughput. However, unlike prior works, in this paper, we delve deep into the intersection between quantization and LLM serving.

Weight-only quantization. For LLMs, weight matrices lead to large memory movement, limiting decode efficiency. Weight-only quantization uses low-bit precision to approximate weight matrices. For instance, GPTQ (Frantar et al., 2023) used 4-bit to quantize the weight based on the approximate second-order information. AWQ (Lin et al., 2023) further advanced accuracy by preserving salient weights. SqueezeLLM (Kim et al., 2023) handled outliers through non-uniform quantization and used a sparse format to keep outliers and sensitive weights at high precision. QuiP (Chee et al., 2023) successfully represented weights using 2-bit by an adaptive rounding method. Nonetheless, in the LLM serving scenario, the overhead of loading the weight matrix is amortized due to batching. Thus, the dense layer becomes compute-bound, while weight-only quantization fails to use efficient low-bit hardware to deliver ideal throughput.

Weight-activation quantization. Weight-activation quantization quantizes both the weight and activation matrices, which is considered more challenging due to the outlier phenomenon of the activation. LLM.INT8 (Dettmers et al., 2022) proposed mixed precision to preserve outlier values in activation matrices. (Xiao et al., 2023; Shao et al., 2023; Yao et al., 2022; Wei et al., 2023) used mathematical equivalent transformations to manage activation outliers. RPTQ (Yuan et al., 2023) rearranges the channels to reduce the variance within one quantization group, further enhancing the accuracy. Some works (Liu et al., 2023a; Wu et al., 2023) used low-rank matrices to compensate for quantization error. Others (Guo et al., 2023; Zhou et al., 2023) used algorithm and architecture co-design to accommodate outliers. However, these approaches either suffer significant accuracy loss at extremely low-bit precision or lack practical hardware support. In this work, our method achieves notable accuracy with low-bit representation and ensures practical speedup.

8 Conclusion

We presented Atom, a low-bit quantization method that leverages the underlying hardware efficiently to achieve both high accuracy and high throughput for LLM serving. We use mixed-precision quantization with reordering, fine-grained group quantization, dynamic quantization, and KV-cache quantization to preserve accuracy while fully exploiting emerging low-bit hardware support. We integrate Atom into an end-to-end serving framework, achieving up to 7.73× throughput enhancement compared to the FP16 baseline as well as maintaining less than 1.4% zero-shot accuracy loss.

Acknowledgments

We thank Jiaming Tang and Yixin Dong for their discussion and insightful feedback. This work was supported in part by ACE and PRISM, two of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; by the National Science Foundation (NSF) under grant CCF-1518703 and award CNS-2211882; and by DARPA under the RTML program. The work was also supported by gifts from Qualcomm and Intel (TSA center).

[4]In practice, we find that accuracy is similar when Atom share reorder indices across all experts in an MoE layer. Therefore, we use shared indices for efficiency consideration.

REFERENCES

Abdelkhalik, H., Arafa, Y., Santhi, N., and Badawy, A.-H. Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis, 2022.

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language, 2019.

Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. Quip: 2-bit quantization of large language models with guarantees, 2023.

Chen, L. Dissecting batching effects in gpt inference, May 2023. URL https://le.qun.ch/en/blog/2023/05/13/transformer-batching/.

Chen, L., Ye, Z., Wu, Y., Zhuo, D., Ceze, L., and Krishnamurthy, A. Punica: Multi-tenant lora serving, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., and Liang, W. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

Duarte, F. Number of chatgpt users, Jul 2023. URL https://explodingtopics.com/blog/chatgpt-users.

Elimian, G. Chatgpt costs 700,000 to run daily, openai may go bankrupt in 2024, Aug 2023. URL https://technext24.com/2023/08/14/chatgpt-costs-700000-daily-openai.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022.

Guo, C., Tang, J., Hu, W., Leng, J., Zhang, C., Yang, F., Liu, Y., Guo, M., and Zhu, Y. OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ACM, jun 2023. doi: 10.1145/3579371.3589038. URL https://doi.org/10.1145%2F3579371.3589038.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

HuggingFace. Sharegpt vicuna unfiltered, May 2023. URL https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts, 2024.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. Squeezellm: Dense-and-sparse quantization, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient

memory management for large language model serving with pagedattention, 2023.

Lee, C., Jin, J., Kim, T., Kim, H., and Park, E. Owq: Lessons learned from activation outliers for weight quantization in large language models. *ArXiv*, abs/2306.02272, 2023. URL https://api.semanticscholar.org/CorpusID:259076427.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration, 2023.

Liu, J., Gong, R., Wei, X., Dong, Z., Cai, J., and Zhuang, B. Qllm: Accurate and efficient low-bitwidth quantization for large language models, 2023a.

Liu, S., Liu, Z., Huang, X., Dong, P., and Cheng, K. Llm-fp4: 4-bit floating-point quantized transformers, 2023b.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pp. 114–119, USA, 1994. Association for Computational Linguistics. ISBN 1558603573. doi: 10.3115/1075812.1075835. URL https://doi.org/10.3115/1075812.1075835.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellempudi, N., Oberman, S., Shoeybi, M., Siu, M., and Wu, H. Fp8 formats for deep learning, 2022.

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., and Blankevoort, T. A white paper on neural network quantization, 2021.

NVIDIA. Nvidia a100 specifications, a. URL https://www.nvidia.com/en-us/data-center/a100/.

NVIDIA. Nvidia tensor core, b. URL https://www.nvidia.com/en-us/data-center/tensor-cores/.

NVIDIA. Nvidia blackwell platform arrives to power a new era of computing, March 2024a. URL https://nvidianews.nvidia.com/news/nvidia-blackwell-platform-arrives-to-power-a-new-era-of-computing.

NVIDIA. Nvbench: Nvidia's benchmarking tool for gpus, 2024b. Available online: https://github.com/NVIDIA/nvbench.

Patel, P., Choukse, E., Zhang, C., Íñigo Goiri, Shah, A., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting, 2023.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *ArXiv*, abs/2211.05102, 2022. URL https://api.semanticscholar.org/CorpusID:253420623.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

Rouhani, B. D., Zhao, R., More, A., Hall, M., Khodamoradi, A., Deng, S., Choudhary, D., Cornea, M., Dellinger, E., Denolf, K., Dusan, S., Elango, V., Golub, M., Heinecke, A., James-Roxby, P., Jani, D., Kolhe, G., Langhammer, M., Li, A., Melnick, L., Mesmakhosroshahi, M., Rodriguez, A., Schulte, M., Shafipour, R., Shao, L., Siu, M., Dubey, P., Micikevicius, P., Naumov, M., Verrilli, C., Wittig, R., Burger, D., and Chung, E. Microscaling data formats for deep learning, 2023.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale, 2019.

Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., Li, Z., Zhang, K., Gao, P., Qiao, Y., and Luo, P. Omniquant: Omnidirectionally calibrated quantization for large language models, 2023.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., Xie, Z., Chen, B., Barrett, C. W., Gonzalez, J., Liang, P., Ré, C., Stoica, I., and Zhang, C. High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, 2023. URL https://api.semanticscholar.org/CorpusID:257495837.

Thakkar, V., Ramani, P., Cecka, C., Shivam, A., Lu, H., Yan, E., Kosaian, J., Hoemmen, M., Wu, H., Kerr, A., Nicely, M., Merrill, D., Blasig, D., Qiao, F., Majcher, P., Springer, P., Hohnerbach, M., Wang, J., and Gupta, M. CUTLASS, January 2023. URL https://github.com/NVIDIA/cutlass.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Trauzettel-Klosinski, S., Dietz, K., and the IReST Study Group. Standardized Assessment of Reading Performance: The New International Reading Speed Texts IReST. *Investigative Ophthalmology & Visual Science*, 53(9):5452–5461, 08 2012. ISSN 1552-5783. doi: 10.1167/iovs.11-8284. URL https://doi.org/10.1167/iovs.11-8284.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

Wei, X., Zhang, Y., Li, Y., Zhang, X., Gong, R., Guo, J., and Liu, X. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling, 2023.

Wikipedia contributors. List of qualcomm snapdragon systems on chips — Wikipedia, the free encyclopedia, 2023. URL https://en.wikipedia.org/w/index.php?title=List_of_Qualcomm_Snapdragon_systems_on_chips&oldid=1182026635. [Online; accessed 26-October-2023].

Williams, S., Waterman, A., and Patterson, D. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Wu, X., Yao, Z., and He, Y. Zeroquant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models, 2023.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers, 2022.

Ye, Z., Chen, L., Lai, R., Zhao, Y., Zheng, S., Shao, J., Hou, B., Jin, H., Zuo, Y., Yin, L., Chen, T., and Ceze, L. Accelerating self-attentions for llm serving with flashinfer, February 2024. URL https://flashinfer.ai/2024/02/02/introduce-flashinfer.html.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Yuan, Z., Niu, L., Liu, J., Liu, W., Wang, X., Shang, Y., Sun, G., Wu, Q., Wu, J., and Wu, B. Rptq: Reorder-based post-training quantization for large language models, 2023.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence?, 2019.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024.

Zhou, C., Richard, V., Savarese, P., Hassman, Z., Maire, M., DiBrino, M., and Li, Y. Sysmol: A hardware-software co-design framework for ultra-low and fine-grained mixed-precision neural networks, 2023.