ABQ-LLM: Arbitrary-Bit Quantized Inference Acceleration for Large Language Models

Chao Zeng [*†], **Songwei Liu**[*], **Yusheng Xie**[*], **Hong Liu, Xiaojian Wang, Miao Wei, Shu Yang, Fangmin Chen, Xing Mei** [‡]

ByteDance Inc, Shenzhen, China

{zengchaocs, cfangmin}@gmail.com, {liusongwei.zju, xieyusheng.12, xing.mei}@bytedance.com

Abstract

Large Language Models (LLMs) have revolutionized natural language processing tasks. However, their practical application is constrained by substantial memory and computational demands. Post-training quantization (PTQ) is considered an effective method to accelerate LLMs inference. Despite its growing popularity in LLMs model compression, PTQ deployment faces two major challenges. First, low-bit quantization leads to performance degradation. Second, restricted by the limited integer computing unit type on GPUs, quantized matrix operations with different precisions cannot be effectively accelerated. To address these issues, we introduce a novel arbitrary-bit quantization algorithm and inference framework, ABQ-LLM. It achieves superior performance across various quantization settings and enables efficient arbitrary-precision quantized inference on the GPU. ABQ-LLM introduces several key innovations: (1) a distribution correction method for transformer blocks to mitigate distribution differences caused by full quantization of weights and activations, improving performance at low bit-widths. (2) the bit balance strategy to counteract performance degradation from asymmetric distribution issues at very low bit-widths (e.g., 2-bit). (3) an innovative quantization acceleration framework that reconstructs the quantization matrix multiplication of arbitrary precision combinations based on BTC (Binary TensorCore) equivalents, gets rid of the limitations of INT4/INT8 computing units. ABQ-LLM can convert each component bit width gain into actual acceleration gain, maximizing performance under mixed precision(e.g., W6A6, W2A8). Based on W2*A8 quantization configuration on LLaMA-7B model, it achieved a WikiText2 perplexity of 7.59 (2.17↓ vs 9.76 in AffineQuant). Compared to SmoothQuant, we realized 1.6× acceleration improvement and 2.7× memory compression gain.

Introduction

Recent advancements in large language models (LLMs) (Bubeck et al. 2023; Touvron et al. 2023a,b) have demonstrated impressive capabilities across various natural language benchmark, including reasoning (Clark et al. 2019, 2018), cognitive processing (Xu et al. 2023a; Hardy et al. 2023), and dialogue generation(Hu et al. 2023). However, these models are characterized by a substantial number of parameters, posing significant challenges in terms of memory consumption and bandwidth (Zheng et al. 2024; Kim et al. 2023).

Post-training quantization (PTQ) effectively reduces both computational and storage requirements. This technique significantly accelerates model inference by converting the weights and activation values of large language models (LLMs) from high-precision floating-point numbers to low-precision integer values for storage, and using efficient integer matrix multiplication operators to handle the bulk of matrix multiplication computations during inference. Currently, 80% of the computation and parameter access in LLMs is concentrated on general matrix multiplication (GEMM) and vector multiplication (GEMV) operations, especially during autoregressive decoding, where all GEMM operations degrade into GEMV operations due to single-token generation. Consequently, the efficiency of GEMV computation and memory access directly determines the efficiency and power consumption of LLM inference.

To improve GEMM/GEMV memory access efficiency, LLMs inference typically employs a quantized inference strategy. The current mainstream approach is weight-only quantization, where the kernel performs actual computation based on dequantized FP16 values. However, this approach offers limited performance improvement in highly parallel scenarios. To further enhance quantized inference performance, the industry is pursuing full quantization of both weights and activation values to reduce activation memory access and leverage higher computational power using quantized kernels, such as those from NVIDIA. However, current industry practices in weight and activation full quantization (WA full quantization) face several limitations. NVIDIA provides only a limited set of hardware-accelerated instructions(Lin et al. 2024a; Ashkboos et al. 2024; Zhao et al. 2024), which constrains the design space for quantization algorithms. Other quantization combinations (e.g., W4A8 or W2A4) require type conversion to W8A8 or W4A4 during computation, leading to inefficiency(Lin et al. 2024b). Furthermore, due to GEMV, additional padding calculations are required in scenarios with a batch size less than 8, resulting in inefficient matrix multiplication for W4A4 and W8A8. Fi-

[*]These authors contributed equally.

[†]Work was done when Chao Zeng was intern at ByteDance Inc.

[‡]Corresponding author

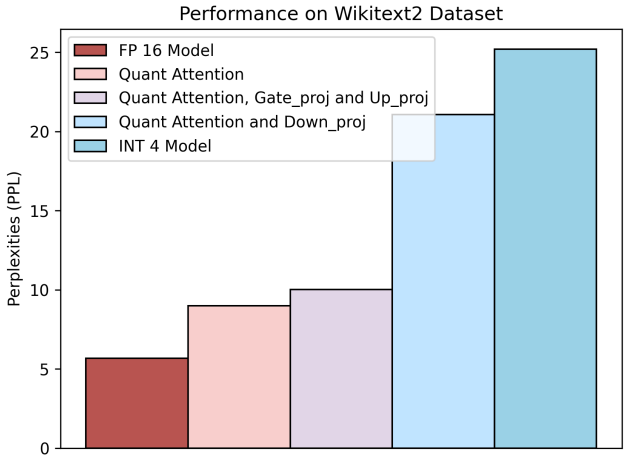

Figure 1: Perplexities analysis (lower is better) on the Wikitext2 dataset of LLaMA-7B with different quantized modules.

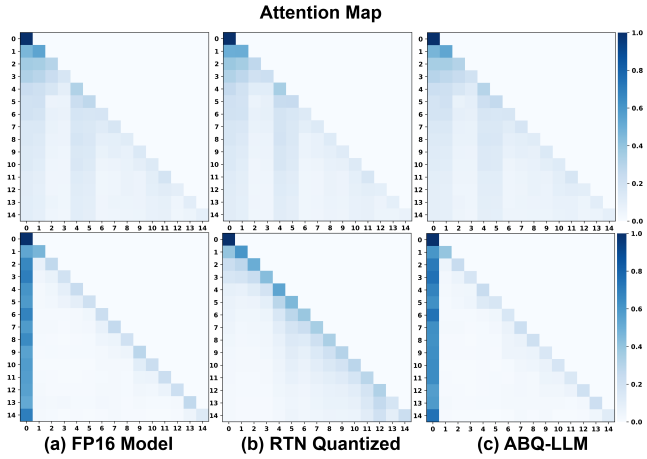

Figure 2: Attention maps for the first block (above) and the final block (below) are shown.

nally, WA fully quantized models encounter significant challenges in low-bit quantization (e.g., W2A8, W2A6).

In this paper, we introduce a novel quantization framework for PTQ, called ABQ-LLM. By examining the quantization sensitivity of components within the transformer block (Figure 1) and the attention map before and after quantization (Figure 2), we find the down_proj linear layer and the attention map particularly sensitive to quantization. To address this, we propose a double cosine similarity distribution correction and an attention map distribution bootstrap for the output of down_proj. This method calibrates the quantization constants and restores the model's performance at low bit-widths such as W6A6, W4A4 and W2A8. Additionally, we analyze performance degradation in low-bit quantization and address the asymmetric loss issue in low-bit representations like INT2 using the bit balance strategy. Finally, we implement customized software engine to support fully quantized inference of various precision combinations based on BTC equivalents, fully exploiting the advantages of quantized models under mixed precision. Our contributions are summarized as follows:

- We propose a novel block-wise distribution correction and compensation scheme in the PTQ domain to mitigate the distribution discrepancy caused by full quantization of weights and activations, thereby improving model performance at low bit-widths.
- We address the problem of asymmetric loss at low bit-widths, such as INT2, and significantly improve INT2 quantization performance using the bit balance strategy, enhancing model performance under the INT2 quantization configuration.
- We propose a software engine which achieves quantization freedom for the first time in the LLM field. It eliminates the limitations of INT4/INT8 computational units, and effectively avoids the GEMV problem. Under the LLaMA-7B W2A8 configuration, it has 1.6× ultimate acceleration compared to SmoothQuant, achieving SOTA performance.

Related Work

LLM quantization can be broadly divided into weight-only quantization and weight-activation quantization.

Weight-only quantization. To alleviate computational burdens, some studies focus on weight-only quantization. GPTQ (Frantar et al. 2022) uses Hessian-based error compensation to reduce quantization errors in LLMs, enabling 3-bit quantization. AWQ (Lin et al. 2024a) and OWQ (Lee et al. 2024) significantly enhance quantized model performance by considering the impact of activation outliers on weight quantization. Methods like QuIP(Chee et al. 2024), QuIP# (Tseng et al. 2024), and AQLM(Egiazarian et al. 2024) facilitate 2-bit quantization through learnable codebooks or additional fine-tuning. Approaches such as (Dettmers et al. 2023; Shang et al. 2023; Huang et al. 2024) improve PTQ performance through unstructured mixed-precision fine-grained weight grouping. Additionally, research such as (Dettmers et al. 2024; Xu et al. 2023b; Arshia et al. 2022; Bondarenko, Del Chiaro, and Nagel 2024) employs efficient parameter fine-tuning (PEFT) techniques to compress weights through fine-tuning.

Weight-activation quantization. Weight-activation quantization differs from weight-only quantization by quantizing both weights and activation (including KV caches) to accelerate LLM inference. The main challenge in quantizing activation is handling outliers, which can cause significant quantization errors. To address this issue, ZeroQuant(Yao et al. 2022) proposes a fine-grained, hardware-friendly quantization scheme for weights and activation. SmoothQuant(Xiao et al. 2023) shifts the quantization difficulty from activation to weights through mathematically equivalent transformations, achieving W8A8 quantization. (Shao et al. 2023; Ma et al. 2024b; Hu et al. 2024a) enhances performance by training quantization parameters. Limited by GPU platform instruction limitations, these jobs can only use W8A8 to per-

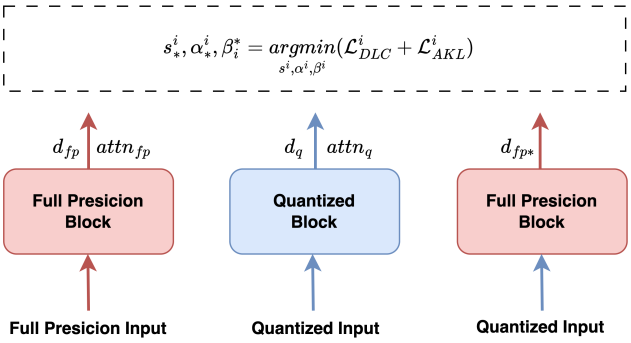

Figure 3: An overview of our ABQ-LLM. ABQ-LLM use our DLC loss and AKL loss to update learnable parameters.

form actual inference, even if they achieve lower quantization bit-widths (e.g., W6A6).

Method

In this section, we provide a detailed introduction to our ABQ-LLM. We first describe the distribution correction and bit balance strategy and then introduce our arbitrary-bit inference framework.

Preliminary

(Xiao et al. 2023) achieves WA full quantization by scaling activation outliers, but this increases the range variability of weights, making weight quantization more sensitive. Conversely, (Lin et al. 2024a) optimizes weight quantization by scaling weights, which significantly increases the diversity of activation, complicating activation quantization. These approaches highlight the drawbacks of manually setting scaling balance factors between activation and weight, making it challenging to achieve a perfect balance. To address this issue, we introduce a distribution correction-guided scaling method. Following (Shao et al. 2023) approach, we set the balance vectors between weights and activation as learnable parameters and add a learnable clipping parameter for weight. By employing our distribution correction and bit balance strategy to optimize model performance, our objectives are as follows:

$$\underset{s,\alpha,\beta}{\operatorname{argmin}} \left\| WX - Q(\text{clip}(W) \cdot \text{diag}(s)) Q(\text{diag}(s)^{-1} \cdot X) \right\|, \quad (1)$$

where W and X are full-precision weight and activation, $Q(\cdot)$ denotes the quantizer of weight and activation, $clip(\cdot)$ denotes the clipping operation, s is the scale factor, and letting $W_{max} = \alpha \max(W)$ and $W_{min} = \beta \min(W)$ to control the clipping range of the weight.

Improving Quantization by Distribution Correction

We observed significant variations in sensitivity across different layers of LLM models during quantization, with some layers having a critical impact on quantization performance. To validate this, as shown in Figure 1, we quantified various components of the LLaMA-7B model under weight-activation full quantization. While quantizing the gate_proj and up_proj layers in mlp and attention resulted in only minor performance degradation, quantizing the down_proj linear layer caused a substantial performance drop. This indicates that addressing down_proj quantization is crucial for performance recovery. Further analysis revealed that the primary cause of performance degradation due to down_proj quantization is the quantization of down_proj activation. At low bit-widths such as INT4, INT3, and INT2, the limited representation range causes a significant shift in the model distribution compared to full precision. As illustrated in Figure 3, during the block-wise quantization calibration process, we apply a double logarithm of cosine similarity loss on the output of down_proj to correct the distribution of the quantized model. The loss function called DLC loss $\mathcal{L}^i_{DLC}$:

$$\mathcal{L}^i_{DLC} = -\log\left(\frac{d^i_q \cdot d^i_{fp}}{\left\|d^i_q\right\| \left\|d^i_{fp}\right\|}\right) - \log\left(\frac{d^i_q \cdot d^i_{fp^*}}{\left\|d^i_q\right\| \left\|d^i_{fp^*}\right\|}\right), \quad (2)$$

where d^i_q represent the quantized output of the i-th transformer block, d^i_{fp} represent the full-precision output of the i-th transformer block, and $d^i_{fp^*}$ represent the full-precision output of the i-th transformer block, with its input originating from the quantized output of the $(i-1)$-th transformer block.

Additionally, we conducted an analysis of the cosine similarity between activation at the input and output of decoder blocks in the LLaMA-7B model. The results revealed significant differences in similarity for the initial and final blocks, indicating their considerable impact on model inference performance. In response, we applied distribution compensation vector to the down_proj layers of these blocks to address and correct the distribution discrepancies using Eq. (3).

$$W_q = \text{clamp}(\lceil \frac{W + \gamma a b^\top}{\triangle} \rfloor + z, 0, 2^n - 1), \quad (3)$$

where $\lceil \cdot \rfloor$ denotes round operation, n represents the target bit-width, $\triangle$ denotes the step-size, and z is the zero-point. W_q and W denote the quantized and full-presicion weight, respectively. The vectors a and b are distribution compensation vectors, where $\gamma = 1$ indicates compensation is performed, and $\gamma = 0$ indicates no compensation.

To enhance the performance of the quantized model, we analyzed the changes in Attention Map distribution before and after quantization, as shown in Figure 2. In the full-precision model, attention is heavily focused on the first token, highlighting its key role in guiding text generation, consistent with LLM-QAT(Liu et al. 2023) findings. However, quantization disrupts this attention pattern, diminishing focus on the first token. To address this and restore the model's attention during quantization, we introduced attention-aware KL divergence to reconstruct the attention map.

$$\mathcal{L}^i_{AKL} = D_{KL}(attn^i_q \parallel attn^i_{fp}) + D_{KL}(attn^i_{fp} \parallel attn^i_q), \quad (4)$$

where $attn^i_q$ denotes the quantized attention map output of the i-th transformer block, while $attn^i_{fp}$ refers to the full-precision attention map output of the same block.

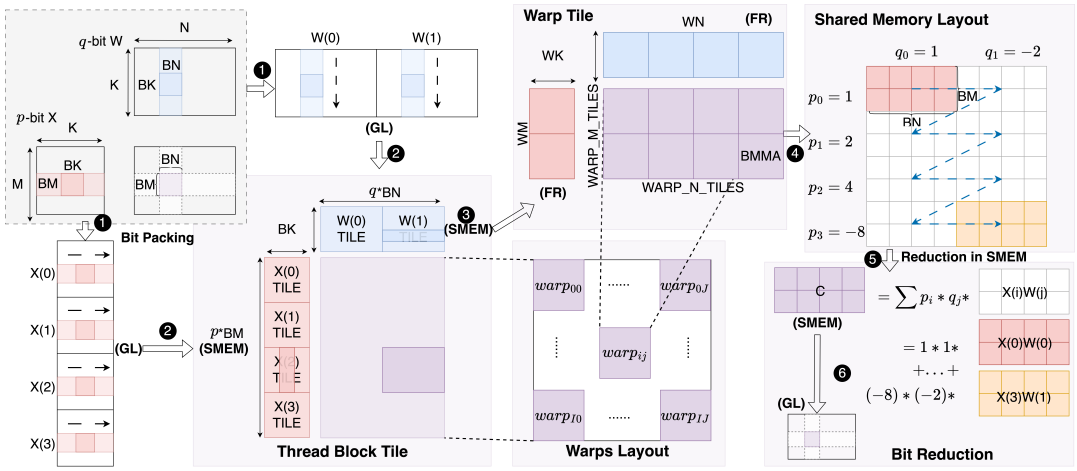

Figure 4: System Overview of Custom Software Engine. **p** and **q** represent the quantization bit width of input X and weight W. Data flows are carefully designed to support efficient computation: global memory(**GL**) → shared memory(**SMEM**) → fragment(**FR**) → shared memory(**SMEM**) → global memory(**GL**).

In the end, we combined DLC loss and AKL loss, and our final optimization goal is:

$$s_*^i, \alpha_*^i, \beta_i^* = \underset{s^i, \alpha^i, \beta^i}{\operatorname{argmin}} (\mathcal{L}_{DLC}^i + \mathcal{L}_{AKL}^i), \tag{5}$$

where s_*^i, α_*^i and β_i^* are the parameters of the i-th transformer block after calibration. When the distributions of the quantized output and the full-precision output match, we have a loss close to 0, which effectively guides the quantization process.

Bit Balance Strategy

Typically, pre-trained LLM model weight exhibit near-normal distribution, characterized by symmetry. Using Q-Q plots (Quantile-Quantile Plots), we confirmed the strong symmetry in the weight distribution of pre-trained models. However, in standard INT2 quantization, the numerical representation is limited to four values, with symmetric quantization ranges of {-2, -1, 0, 1} or {-1, 0, 1, 2}, disrupting the original symmetric weight distribution. This asymmetry leads to significant performance degradation, as shown in Table 1, where performance drops by 0.46 from W4A16 to W3A16 and by 5.19 from W3A16 to W2A16, indicating a sharp decline. To address this asymmetry impact on LLMs quantization, we adopted the bit balance strategy like (Li et al. 2016; Ma et al. 2024a), extending the INT2 symmetric quantization space to {-2, -1, 0, 1, 2}. This modification restored model performance to 7.50, which is within a reasonable range compared to W3A16.

Custom Software Engine

Reconstructing Arbitrary Bit Computation. To support W1A1 quantization, NVIDIA introduced INT1 TensorCore in Turing and later architectures to provide hardware support. However, W1A1 quantization has not been widely applied due to significant performance degradation. Through mathematical analysis of quantized matrix multiplication, we find that any combination of quantization can be decomposed into a superposition of 1-bit matrix multiplications. Assuming the weight W of a particular neural network layer are quantized to q bits and the input activation value X is quantized to p bits, the matrix multiplication of W and X results in a 32-bit output $Y = WX$. The key is to observe that the scalar values at any position in W and X can be decomposed into a series of 1-bit scalar numbers. Scalar operations with any combination of precision can be decomposed into 1-bit operations and shift operations. For example, a 2-bit x can be expressed as:

$$x = x^1 x^0, \text{where} \quad x^i \in \text{INT1}, \tag{6}$$

where $x^1 = (x \gg 1)\&1$, $x^0 = (x \gg 0)\&1$. We use $OP(a, b)$ to denote a computational operation where the input is 1-bit data and the output is 32-bit. Thus, the original scalar-level arbitrary precision computation wx can be represented as:

$$wx = w * (x^1 x^0) = OP(w, x^1) * 2 + OP(w, x^0). \tag{7}$$

The above procedure can be generalized to any combination of matrix multiplications with bit-widths p and q. The detailed formulas are shown in the Appendix. Using these transformations, we decompose the operation of arbitrary quantized combinations into a superposition of 1-bit matrix multiplications, enabling the underlying layer to invoke high-computing instruction implementations.

Engine Implementation. NVIDIA GPU has many processing elements called Streaming Multiprocessors (SMs) and uses a large number of threads to perform computing tasks in parallel. Threads are structured into thread blocks, which become the smallest scheduling execution unit on SMs. Therefore, the computation target is decomposed and mapped to each thread block, called **Thread Block Tile**, to achieve parallel computing. As shown in Figure 4, for a GEMM task of shape M×N×K, each thread block is responsible for computing a BM×BN output block, which is decomposed into $\frac{K}{BK}$

Model	Bits	WikiText2	C4
LLaMA-7B	W4A16	5.83	7.29
	W3A16	6.29	8.01
	W2A16	11.48	15.74
	W2*A16	7.50	9.86

Table 1: Performance comparison of LLaMA-7B under W4, W3, and W2 quantization configurations. * denotes the use of the bit balance strategy.

sub-GEMM tasks of shape BM×BN×BK. Our engine converts quantized matrix multiplications with bit widths configured as {p, q} into special accumulations of p*q binarized matrix multiplications, so the true calculation task of thread block tile is p*BM × q*BN. ❶ First, in order to improve memory access continuity, we propose **BitPacking** strategy to decompose the quantized tensor into n binary matrices, where n is the quantization bit width. Taking input X as an example, this means that its bit perspective layout changes from [M, K, p] to [p, M, K]. All threads within a thread block share the same shared memory space. Within each thread block, threads are further organized into a set of warps, with each warp consisting of 32 consecutive threads. ❷ Next, warps collaborates to load the A matrix (p*BM × BK) and B matrix (BK × q*BN) data required for thread block tile calculation from GL and caches them in SMEM. Thanks to BitPacking, the process of reading p BM*BK single-bit row-major tiles and writing p*BM*BK bits SMEM is efficient and continuous. ❸ Subsequently, thread block contains multiple warps, so thread block tile can be further decomposed into **Warp Tile** to achieve warp-level parallelism, and the computing tasks of each warp are WM× WN. In the calculation preparation stage, the A matrix (WM×WK, row-major) and B matrix (WK×WN, col-major) are independently loaded from SMEM to FR. Then, the calculation is decomposed into WM_TILES*WARP_N_TILES TensorCore MMA(matrix-multiply-accumulate). Since A and B are binarized matrices, we actually use Binary TensorCore MMA (BMMA), which has a computing power 8 times and 4 times higher than INT8 and INT4 TensorCore respectively. ❹ All warps collaboratively complete the Thread Block Tile calculation, and the results are stored in the c fragment of each warp. Therefore, each warp needs to independently write the calculation results back to SMEM. ❺ Output tile(p*BM × q*BN) is globally reduced to obtain a final result(BM × BN), where each BM × BN sub-Tile needs to be multiplied by a certain scaling factor. We call this process **Bit Reduction**. ❻ As the final step, warps collaboratively load the final result from SMEM and write back to the target location in GL.

We implement the above calculation process as a GPU Kernel, called ABQKernel. ABQKernel is used to replace all gemm operations in the decoder layer, and assists with necessary BitPacking, quantization, and dequantization operations to achieve arbitrary quantization inference of the LLaMA model. We carefully manages the overhead of quantization operators by fusing them into existing operators and weight BitPacking is implemented offline for increased efficiency.

GPU Kernel Optimization. When M=1, the GEMM problem of shape MxNxK transforms into a GEMV problem, shifting from computation-intensive to memory-intensive, which becomes a performance bottleneck for model inference. When using ordinary TensorCore for accelerated computation, the dimensions of M are usually chunked in groups of 8, requiring padding if M<8, leading to 87.5% redundant computation. Thanks to the revolutionary reconstruction of computing and BitPacking strategy, for the W_qA_p configuration, the actual computing task undertaken by ABQKernel is p*M × q*N × K. The expansion of the M dimension can effectively reduce the redundant calculations when calling TensorCore, and even when p*M >= 8 and p*M % 8 = 0, padding can be completely avoided. We call the above optimization strategy *GEMV Elimination*. In addition, *Computational and Pipeline Optimization*, *Auto Kernel Search*, and *Bank Conflicts Elimination* are also applied.

Experiments

Experimental Setup

Baseline. For weight-only quantization, we compare our approach with GPTQ(Frantar et al. 2022), AWQ(Lin et al. 2024a), OmniQuant(Shao et al. 2023), and AffineQuant(Ma et al. 2024b). For weight-activation quantization, we benchmark our method against SmoothQuant(Xiao et al. 2023), OmniQuant(Shao et al. 2023), and I-LLM(Hu et al. 2024b).

Models and Datasets. We primarily evaluate our method using LLaMA (7B-13B) (Touvron et al. 2023a) and LLaMA-2 (7B-13B) (Touvron et al. 2023b)in this paper. Following previous work(Shao et al. 2023; Ma et al. 2024b), we evaluate the quantized models by reporting the perplexity of language generation experiments on WikiText2(Merity et al. 2016) and C4(Raffel et al. 2020). To assess performance on zero-shot tasks, we select several popular benchmarks including PIQA(Bisk et al. 2020), ARC(Clark et al. 2018), BoolQ(Clark et al. 2019), HellaSwag(Zellers et al. 2019), and Winogrande(Sakaguchi et al. 2021) using the lm-evaluation-harness(Gao et al. 2021).

Calibration. We initialize the balance vectors for weights and activations following (Xiao et al. 2023), with the learnable clipping parameter for weights set to 1. For distribution correction compensation vectors, we set a as an all-ones vector and b as an all-zeros vector, ensuring $ab^\top$ starts at 0. Using the AdamW optimizer (Loshchilov and Hutter 2017) with no weight decay, we set learning rates of 5e-3 for balance vectors and 1e-2 for the clipping parameter and vector compensation vector. Calibration data includes 128 randomly selected 2048-token segments from WikiText2. The calibration process, conducted on an NVIDIA A800-40G GPU, utilized a batch size of 1 and spanned 20 epochs. For activation and KV Cache we perform per-token quantization, and for weight we perform per-channel quantization. By default, activation and KV cache use the same quantization bit.

Bits	Method	LLaMA-7B		LLaMA-13B		LLaMA-2-7B		LLaMA-2-13B	
		WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4
W6A6	SmoothQuant	6.03	7.47	5.42	6.97	6.20	7.76	5.18	7.67
	OmniQuant	5.96	7.43	5.28	6.84	5.87	7.48	5.14	6.74
	I-LLM	5.84	7.32	5.23	6.79	5.68	7.27	5.10	6.74
	ABQ-LLM	**5.81**	**7.27**	**5.21**	**6.77**	**5.63**	**7.21**	**5.00**	**6.64**
W4A4	SmoothQuant	22.25	32.22	40.05	47.18	83.12	77.27	35.88	43.19
	OmniQuant	11.26	14.51	10.87	13.78	14.26	18.02	12.30	14.55
	AffineQuant	10.28	13.64	10.32	13.44	12.69	15.76	11.45	13.97
	I-LLM	9.10	12.33	7.99	10.96	10.55	12.92	9.76	12.57
	ABQ-LLM	**8.63**	**12.10**	**7.69**	**10.90**	**9.31**	**12.85**	**8.62**	**11.47**
W2A8	OmniQuant	15.70	26.44	13.50	19.01	37.95	103.39	21.74	31.72
	AffineQuant	**9.76**	15.52	9.21	12.55	1483	4688	**12.30**	29.32
	I-LLM	14.08	18.89	11.80	16.19	123.93	200.54	25.74	40.59
	ABQ-LLM	11.35	**15.41**	**9.20**	**12.48**	**13.47**	**17.82**	13.24	**18.07**
W2*A8	**ABQ-LLM**	**7.59**	**10.00**	**6.49**	**8.53**	**7.85**	**10.33**	**6.65**	**10.01**

Table 2: Weight-activation quantization perplexities (lower is better) comparison of quantized LLaMA and LLaMA-2 models. * denotes the use of the bit balance strategy. More results can be found in at Appendix.

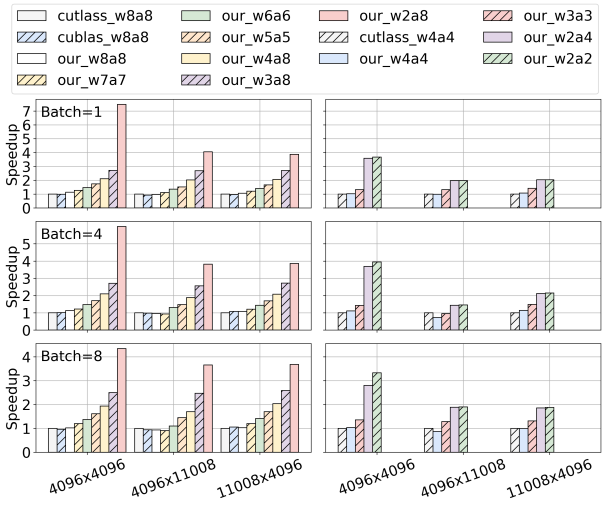

Figure 5: The GEMV speedup comparison of our ABQKernel, CUTLASS (W4A4/W8A8), and cuBLAS (W8A8) in RTX 3070. The left side is compared against W8A8 and the right side against W4A4. More result in RTX 4080 can be found in Appendix.

Experiments on Language Generation Tasks

Language generation capability is central to large language models (LLMs). To validate our extraordinary performance in the challenging quantization task, we first compare perplexity, a crucial metric for language generation, with the baseline. As shown in Table 2, ABQ-LLM demonstrates outstanding performance across various quantization configurations, surpassing state-of-the-art methods such as AffineQuant and I-LLM. Notably, in the INT2 setting, the application of bit balance strategy yields significant improvements at minimal cost. The W2*A8 configurations substantially outperform the W2A8 configurations. Specifically, perplexity on WikiText2 and C4 datasets decreases by an average of 1.42 and 2.11 points, respectively, for W2*A8 compared to the W4A4. These findings validate the effectiveness of our distribution correction and bit balance strategy. The results of weight-only quantization are presented at Appendix. Additional results for full WA quantization are provided at Appendix.

Experiments on Zero-Shot Tasks

To further validate our model, we compare zero-shot accuracy with the baseline method, as shown in Table 3. Our ABQ-LLM method outperforms the previous method in most cases. Notably, after applying the bit balance strategy, the performance of W2*A8 improves significantly by 7.50% on average. Combining the performance in both language generation and zero-shot tasks, we conclude that ABQ-LLM achieves state-of-the-art results in handling challenging quantization tasks. See the Appendix for more results and analysis of quantized configurations.

Inference Engine Evaluation

Kernel Benchmark. We evaluated the GEMV speedup of our ABQKernel across three matrix dimensions in LLaMA-7B and compared it with the quantization kernel provided by cuBLAS and CUTLASS. It is important to note that CUTLASS only supports W4A4 and W8A8, while cuBLAS supports only W8A8 for quantization operations. Our experiments were conducted on two different GPUs: the RTX 4080 and the RTX 3070. Figure 5 presents the results, showing that our ABQKernel achieve superior acceleration across all matrix sizes. Specifically, for special bit combinations such as W2A8 and W2A4, our ABQKernel significantly outperforms the baseline approaches, as cuBLAS and CUTLASS require conversion to W8A8 and W4A4 for computation. In the W2A8 configuration, our throughput is improved by 7.47× compared to W8A8 with CUTLASS and cuBLAS on dimensions (1, 4096) × (4096, 4096).

Model	Bits	Method	PiQA	ARC-e	ARC-c	BoolQ	HellaSwag	Winogrande	Avg.
LLaMA-13B	W4A4	OmniQuant	69.69	47.30	33.10	62.84	58.96	55.80	54.37
		AffineQuant	66.32	43.90	29.61	**64.10**	56.88	54.70	52.58
		I-LLM	67.95	**48.15**	34.47	62.29	63.13	**59.98**	55.99
		ABQ-LLM	**71.82**	47.60	**35.67**	63.52	**64.31**	57.54	**56.74**
	W2A8	OmniQuant	66.76	45.62	30.20	61.13	52.93	55.72	52.06
		AffineQuant	71.00	46.70	**32.33**	62.23	58.62	**63.53**	55.73
		I-LLM	67.46	43.73	29.69	62.41	53.37	55.09	51.95
		ABQ-LLM	**72.03**	**46.72**	31.74	**65.17**	**58.71**	62.50	**56.15**
	W2*A8	**ABQ-LLM**	**74.91**	**54.92**	**38.65**	**68.53**	**68.21**	**66.54**	**61.96**
LLaMA-2-13B	W4A4	OmniQuant	67.08	45.66	32.25	63.73	58.39	54.61	53.62
		AffineQuant	67.68	46.63	32.85	**65.90**	60.62	54.14	54.63
		I-LLM	68.00	45.74	30.97	64.55	60.62	54.22	54.01
		ABQ-LLM	**69.04**	**47.01**	**33.53**	64.74	**62.70**	**54.38**	**55.23**
	W2A8	OmniQuant	62.67	38.80	28.41	62.11	49.04	51.69	48.78
		AffineQuant	61.31	38.51	26.96	62.04	41.92	50.74	46.91
		I-LLM	61.86	38.67	26.45	62.17	43.30	51.85	47.38
		ABQ-LLM	**64.30**	**40.19**	**29.78**	**63.18**	**49.58**	**52.17**	**49.87**
	W2*A8	**ABQ-LLM**	**73.50**	**49.79**	**35.15**	**70.15**	**67.45**	**58.87**	**59.15**

Table 3: Zero-shot accuracies (higher is better) comparison of quantized LLaMA and LLaMA-2 models. * denotes the use of the bit balance strategy. More results can be found in at Appendix.

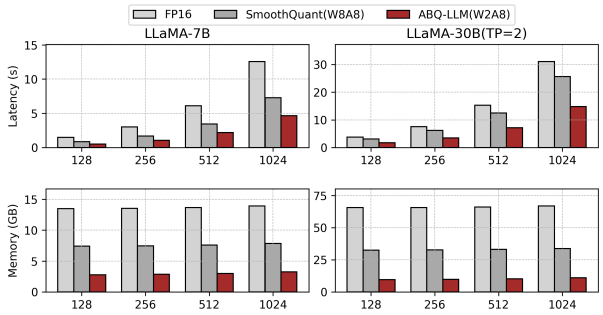

Figure 6: Inference latency (top) and memory usage (bottom) of the FastTransformer implementation on NVIDIA A800-40GB GPU with a fixed input length of 15. More results can be found in at Appendix.

End-to-end throughput. As shown in Figure 6, we integrate our ABQKernel into FastTransformer and compare it with the FP16 implementation of FastTransformer and the INT8 implementation of SmoothQuant. Compared to FP16, our scheme achieves 2.95× inference acceleration and 4.8× memory compression gain, while requiring only 10GB of memory for inference on the LLaMA-30B model, which is less than the memory required for LLaMA-7B with FP16. Additionally, our scheme achieves 1.6× speedup and 2.7× memory compression gain over SmoothQuant, significantly outperforming current mainstream inference methods. This substantial improvement reduces the cost of LLM services and facilitates their practical deployment.

Kernel Optimization Ablation. Table 4 presents the impact of various optimization techniques on the inference latency of the kernel when performing the GEMV operation with dimensions (1, 4096) × (4096, 4096). Our ABQKernel significantly outperforms the CUTLASS W8A8 kernel when unoptimized. Additionally, by employing pipeline optimization, GEMV elimination and auto kernel search, we achieve a latency reduction by 7.47× and a corresponding increase in throughput by 7.47×. These results significantly outperform CUTLASS.

Method	Latency(us)↓	TOPS ↑
CUTLASS	49.96	0.67
Native_kernel	20.05	1.67
+ Pipeline Optimization	14.66	2.28
+ Eliminiate GEMV	10.92	3.07
+ Auto Kernel Search	**6.68**	**5.01**

Table 4: An ablation study of the impact of optimization techniques used in the inference engine on Kernel latency and throughput.

Conclusion

We present an arbitrary bit quantization inference framework called ABQ-LLM. Through an in-depth analysis of LLM quantization, we introduce distribution correction and bit balance strategy to enhance model performance. We then design a novel arbitrary bit inference engine to fully leverage the advantages of LLM quantization. Extensive experimental results demonstrate that ABQ-LLM achieves outstanding performance across various quantization configurations, including W6A6, W4A4, and W2A8. Moreover, ABQKernel consistently outperformed both CUTLASS and cuBLAS in all configurations. Our end-to-end inference speed is 1.6 × faster than the industry SOTA, SmoothQuant, and achieves 2.7 × memory compression gain.

References

Arshia, F. Z.; Keyvanrad, M. A.; Sadidpour, S. S.; and Mohammadi, S. M. R. 2022. PeQA: A Massive Persian Question-Answering and Chatbot Dataset. In *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*, 392–397. IEEE.

Ashkboos, S.; Mohtashami, A.; Croci, M. L.; Li, B.; Jaggi, M.; Alistarh, D.; Hoefler, T.; and Hensman, J. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*.

Bisk, Y.; Zellers, R.; Gao, J.; Choi, Y.; et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 7432–7439.

Bondarenko, Y.; Del Chiaro, R.; and Nagel, M. 2024. Low-Rank Quantization-Aware Training for LLMs. *arXiv preprint arXiv:2406.06385*.

Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Chee, J.; Cai, Y.; Kuleshov, V.; and De Sa, C. M. 2024. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36.

Clark, C.; Lee, K.; Chang, M.-W.; Kwiatkowski, T.; Collins, M.; and Toutanova, K. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Dettmers, T.; Pagnoni, A.; Holtzman, A.; and Zettlemoyer, L. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Dettmers, T.; Svirschevski, R.; Egiazarian, V.; Kuznedelev, D.; Frantar, E.; Ashkboos, S.; Borzunov, A.; Hoefler, T.; and Alistarh, D. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Egiazarian, V.; Panferov, A.; Kuznedelev, D.; Frantar, E.; Babenko, A.; and Alistarh, D. 2024. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Gao, L.; Tow, J.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; McDonell, K.; Muennighoff, N.; et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10: 8–9.

Hardy, M.; Sucholutsky, I.; Thompson, B.; and Griffiths, T. 2023. Large language models meet cognitive science: LLMs as tools, models, and participants. In *Proceedings of the annual meeting of the cognitive science society*, volume 45.

Hu, X.; Chen, Y.; Yang, D.; Zhou, S.; Yuan, Z.; Yu, J.; and Xu, C. 2024a. I-LLM: Efficient Integer-Only Inference for Fully-Quantized Low-Bit Large Language Models. *arXiv preprint arXiv:2405.17849*.

Hu, X.; Chen, Y.; Yang, D.; Zhou, S.; Yuan, Z.; Yu, J.; and Xu, C. 2024b. I-LLM: Efficient Integer-Only Inference for Fully-Quantized Low-Bit Large Language Models. *arXiv preprint arXiv:2405.17849*.

Hu, Z.; Feng, Y.; Luu, A. T.; Hooi, B.; and Lipani, A. 2023. Unlocking the potential of user feedback: Leveraging large language model as user simulators to enhance dialogue system. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 3953–3957.

Huang, W.; Liu, Y.; Qin, H.; Li, Y.; Zhang, S.; Liu, X.; Magno, M.; and Qi, X. 2024. Billm: Pushing the limit of post-training quantization for llms. *arXiv preprint arXiv:2402.04291*.

Kim, S.; Hooper, C.; Gholami, A.; Dong, Z.; Li, X.; Shen, S.; Mahoney, M. W.; and Keutzer, K. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*.

Lee, C.; Jin, J.; Kim, T.; Kim, H.; and Park, E. 2024. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 13355–13364.

Li, F.; Liu, B.; Wang, X.; Zhang, B.; and Yan, J. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711*.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024a. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100.

Lin, Y.; Tang, H.; Yang, S.; Zhang, Z.; Xiao, G.; Gan, C.; and Han, S. 2024b. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*.

Liu, Z.; Oguz, B.; Zhao, C.; Chang, E.; Stock, P.; Mehdad, Y.; Shi, Y.; Krishnamoorthi, R.; and Chandra, V. 2023. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Ma, S.; Wang, H.; Ma, L.; Wang, L.; Wang, W.; Huang, S.; Dong, L.; Wang, R.; Xue, J.; and Wei, F. 2024a. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*.

Ma, Y.; Li, H.; Zheng, X.; Ling, F.; Xiao, X.; Wang, R.; Wen, S.; Chao, F.; and Ji, R. 2024b. Affinequant: Affine transformation quantization for large language models. *arXiv preprint arXiv:2403.12544*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9): 99–106.

Shang, Y.; Yuan, Z.; Wu, Q.; and Dong, Z. 2023. Pb-llm: Partially binarized large language models. *arXiv preprint arXiv:2310.00034*.

Shao, W.; Chen, M.; Zhang, Z.; Xu, P.; Zhao, L.; Li, Z.; Zhang, K.; Gao, P.; Qiao, Y.; and Luo, P. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Tseng, A.; Chee, J.; Sun, Q.; Kuleshov, V.; and De Sa, C. 2024. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, 38087–38099. PMLR.

Xu, P.; Shao, W.; Zhang, K.; Gao, P.; Liu, S.; Lei, M.; Meng, F.; Huang, S.; Qiao, Y.; and Luo, P. 2023a. Lvlm-ehub: A comprehensive evaluation benchmark for large vision-language models. *arXiv preprint arXiv:2306.09265*.

Xu, Y.; Xie, L.; Gu, X.; Chen, X.; Chang, H.; Zhang, H.; Chen, Z.; Zhang, X.; and Tian, Q. 2023b. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

Yao, Z.; Yazdani Aminabadi, R.; Zhang, M.; Wu, X.; Li, C.; and He, Y. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35: 27168–27183.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhao, Y.; Lin, C.-Y.; Zhu, K.; Ye, Z.; Chen, L.; Zheng, S.; Ceze, L.; Krishnamurthy, A.; Chen, T.; and Kasikci, B. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6: 196–209.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.