BTC-LLM: Efficient Sub-1-Bit LLM Quantization via Learnable Transformation and Binary Codebook

Hao Gu[1*] **Lujun Li**[1*] **Hao Wang**[2] **LEI WANG**[1] **Zheyu Wang**[3]
Bei Liu[1] **Jiacheng Liu**[1] **Qiyuan Zhu**[1] **Sirui Han**[1†] **Yike Guo**[1†]
[1]The Hong Kong University of Science and Technology [2]City University of Hong Kong
[3]University of Electronic Science and Technology of China
`marcusguhao@gmail.com` `siruihan@ust.hk` `yikeguo@ust.hk`

Abstract

Binary quantization represents the most extreme form of compression, reducing weights to ±1 for maximal memory and computational efficiency. While recent sparsity-aware binarization achieves sub-1-bit compression via weight pruning, it faces critical challenger: performance degradation, mask-management overhead, and limited hardware compatibility. In this paper, we present BTC-LLM, a novel sub-1-bit LLM quantization framework that leverages binary pattern clustering and weight transformation to overcome these limitations. Our approach incorporates two key innovations: (1) a Binary Codebook that clusters recurring vectors into compact indices using custom distance metrics and sign-based updates; (2) a Learnable Transformation that reduces outliers and promotes shared sign patterns among binary weights. This eliminates sparse masks, enabling efficient inference on standard hardware. Extensive evaluations across LLaMA, Qwen, and FBI-LLM families demonstrate that BTC-LLM achieves state-of-the-art results in extreme compression (1.11–0.7 bits). Notably, BTC-LLM compressed to 0.8 bits on LLaMA-2-13B maintains high performance—with only a 3.1% accuracy drop in zero-shot benchmarks—while delivering a 1.6× speedup over FP16.

1 Introduction

Recent Large Language Models (LLMs) such as GPT-4o (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025) have revolutionized natural language processing (NLP), achieving state-of-the-art performance across diverse tasks (Wei et al., 2022). However, the massive scale of models like DeepSeek-R1 (671B parameters) creates unsustainable memory and storage requirements, preventing practical deployment in constrained environments. Model quantization (Ma et al., 2024b) addresses this by reducing numerical precision, slashing memory usage by 4∼8× with minimal accuracy drop. Recent advances, such as Omniquant (Shao et al., 2023) and DuQuant (Lin et al., 2024a), demonstrate that even sub-4-bit methods can maintain > 90% of original model performance.

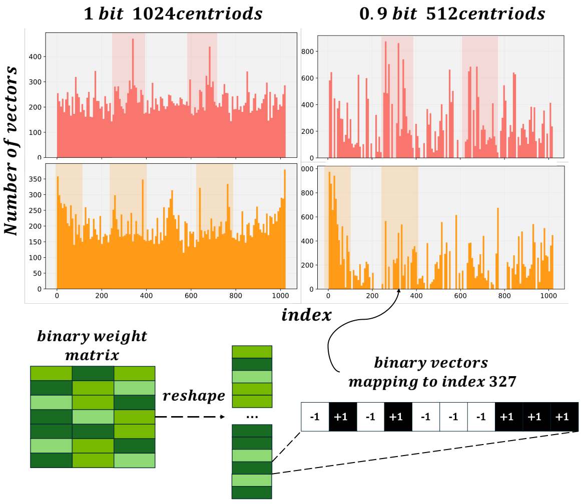

Figure 1: Binary vector distribution (length 10). **Left**: Standard mapping to 1024 indices. **Right**: 512 codebook centroids.

Binary quantization (Rastegari et al., 2016b) represents the most aggressive quantization approach, converting floating-point weights to binary values (±1) to reduce memory requirements by over 32× (Liu et al., 2018). For instance, BitNet (Wang et al., 2023) pioneered QAT for 1-bit LLMs, achieving low memory consumption (0.4GB) and fast inference (29ms). PTQ methods like BiLLM (Huang et al., 2024a) and ARB-LLM (Li et al., 2025) employ advanced binarization strategies (*e.g.*, residual approximation, alternating refinement) to enhance 1-bit LLM performance without retraining. STBLLM (Dong et al., 2025) removes redundant binary parameters to achieve sub-1-bit compression with semi-structured N:M sparsity. However, such sparsity-based binarization faces critical chal-

---

[*] Equal contribution.
[†] Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 23274–23294
July 2-7, 2026

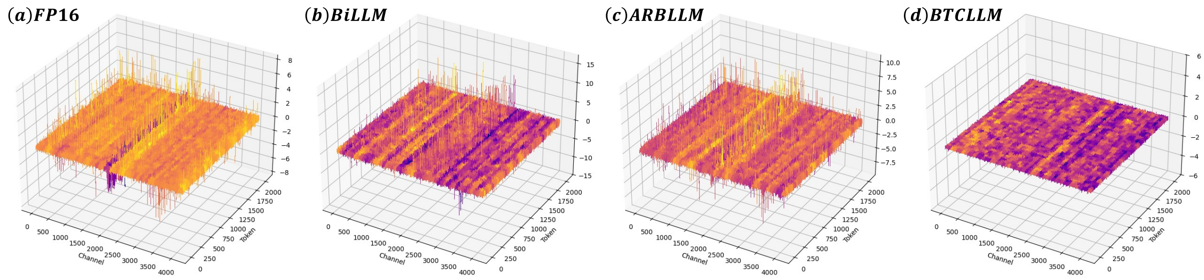

Figure 2: Activation distributions for the `self_attn.k_proj` layer in the LLaMA-2-7B model: (a) Original FP16 (max abs:8), (b) BiLLM (max abs:15), (c) ARB-LLM (max abs:10), and (d) our proposed BTC-LLM (max abs:0.4).

lenges: **(1) Performance Collapse:** STBLLM relies on detecting which elements to prune, yet suffers from accuracy degradation, retaining only 51∼65% of full-precision performance on challenging benchmarks (*e.g.*, ARC-c and HellaSwag). **(2) Hardware Incompatibility:** Structured sparsity such as 2:4 is not a free lunch. In a 4-value tuple, the 2:4 pattern admits $\binom{4}{2} = 6$ possible mask configurations, requiring $\lceil \log_2 6 \rceil = 3$ bits to encode. Consequently, the effective storage cost per weight is

$$\frac{\text{sign bits}(2) + \text{mask bits}(3)}{\text{number of weights}(4)} = 1.25 \text{ bits/weight.}$$

These naturally yield a question: **(RQ) How can we design a hardware-friendly algorithm to further compress binary weights for sub-1-bit LLMs while maintaining performance?** To address this, we first analyze the weight distribution patterns of binarized LLMs to explore potential for more compact compression. As shown in Figure 1, we adopt product quantization by splitting the binary weight matrix into sub-vectors, each mapped to an index (e.g., index 327 corresponds to the binary pattern [-1, +1, -1, +1, ...]). Interestingly, these locally continuous blocks exhibit clear clustering patterns, which motivates us to further compress the model by representing redundant ±1 weights with a compact set of centroid vectors.

We further examine the activation distribution of binarized LLMs and empirically observe the presence of prominent outliers. Such large activations amplify the quantization error, since the forward error term can be expressed as $XW - X\widehat{W} = X(W - \widehat{W})$, where outlier entries in X magnify the impact of binarized weight noise. As shown in Figure 2(b-c), BiLLM shows a wide dynamic range (with absolute values up to 15) with prominent outliers, while ARB-LLM still exhibits noticeable noise and instability. This motivates the need for outlier mitigation, even in binarization methods.

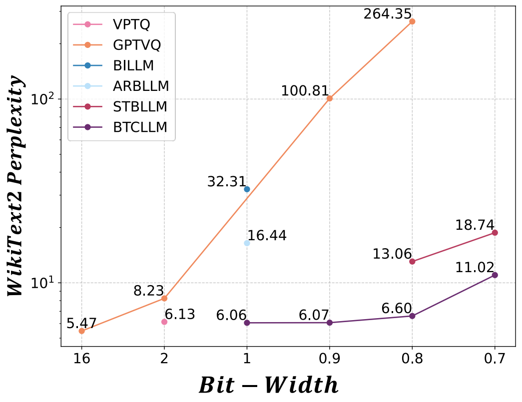

Figure 3: Perplexity of LLaMA-2-7B on WikiText2. Our BTC-LLM outperforms 2-bit methods at 0.9-bit.

Building on these insights, we propose BTC-LLM, a novel framework that enables extreme compression of LLMs to below 1 bit per parameter. Our approach adopts a two-pronged strategy to tackle key challenges. **First, we propose the Flash and Accurate Binary Codebook to leverage binary weight redundancy.** This hardware-friendly approach achieves sub-1-bit compression without the overhead of sparse masks, maintaining model performance by preserving key distributional characteristics. **Second, to mitigate activation outliers and encourage cluster in binary weights, we introduce a Learnable Transformation** consisting of an invertible parameter $D_{\pm}$ and P. Fig. 2(d) shows that this approach suppresses activation outliers, capping the maximum magnitude at 0.4.

As shown in Figure 3, our comprehensive evaluations of the LLaMA family of models (7B to 65B parameters) demonstrate the performance of BTC-LLM in multiple bit width settings. Our binary baseline achieves 6.06 PPL, outperforming even 2-bit quantization methods. BTC-LLM remains robust at 0.9–0.8 bits, matching its 1.11-bit performance. Even at 0.7 bits, it achieves 11.02 PPL with 22× memory savings. It significantly surpasses STBLLM in zero-shot tasks (e.g., +5.0% on LLaMA-2-13B at 0.8 bits), proving its superior efficiency under extreme quantization.

2 Related Work

LLM Quantization reduces memory and computation by representing parameters with fewer bits. The pioneering Quantization-aware Training (QAT) methods (Xu et al., 2026; Chen et al., 2025) like LLM-QAT (Liu et al., 2024b) can achieve excellent results but require extensive retraining that is expensive for billion-parameter LLMs, QaRL (Gu et al., 2026) that enable QAT in RL training to accelerate rollout. Existing PTQ methods fall into two main categories: (1) scaling-based approaches, such as AWQ (Lin et al., 2024b) and SmoothQuant (Xiao et al., 2023), which identify and rescale influential weights to control activation outliers; and (2) rotation-based approaches, such as QuIP# (Tseng et al., 2024) and QuaRot (Ashkboos et al., 2024), which redistribute outliers more evenly across dimensions with transformations.

Binarization represents the most extreme form of quantization, constraining parameters to a single bit (± 1). It was first explored in CNNs with XNOR-Net (Rastegari et al., 2016a) and Bi-Real Net (Liu et al., 2018), and later extended to LLMs by BitNet (Wang et al., 2023), which showed the feasibility of training 1-bit models from scratch. Recent PTQ methods for LLMs include BiLLM (Huang et al., 2024b), which preserves salient weights, and ARB-LLM (Li et al., 2025), which iteratively refines bias and scaling factors. To push beyond 1 bit, STBLLM (Dong et al., 2025) introduced sparsity on binary weights for sub-1-bit compression.

3 Preliminary

Binarization. Binarization represents an extreme form of weight compression in LLMs. For a full-precision weight $\mathbf{W} \in \mathbb{R}^{n \times m}$, we define the objective of binarization as

$$\arg\min_{\alpha, \mathbf{B}} ||\widetilde{\mathbf{W}} - \alpha \mathbf{B}||_F^2, \tag{1}$$

$$\widetilde{\mathbf{W}} = \mathbf{W} - \mu,\ \mu = \frac{1}{m} \sum_{j=1}^{m} \mathbf{W}_{\cdot j}\,, \tag{2}$$

where $\alpha \in \mathbb{R}^n$ denotes the row-wise scaling factor, and $\mathbf{B} \in \{+1, -1\}^{n \times m}$ is a binary matrix.

It is a common practice to apply a row-wise redistribution before binarization first to achieve a zero-mean distribution in a row. Under the objective of binarization (Equation 2), the optimal solutions for α and $\mathbf{B}$ can be solved with $\alpha = \frac{1}{m}\sum_{j=1}^{m} |\widetilde{\mathbf{W}}_{\cdot j}|$ and $\mathbf{B} = \text{sign}(\widetilde{\mathbf{W}})$ respectively. However, simply applying this strategy can incur substantial L_1 binarization error for LLMs, formulated as:

$$L_1 = ||R||_F^2, \quad \text{where } R = W - \alpha_1 B_1 - \mu, \tag{3}$$

To mitigate this error, different approaches have been proposed. BiLLM (Huang et al., 2024b) considers salient weights and approximates the residual with a secondary binarization $R \approx \alpha_2 B_2$. In contrast, ARB-LLM (Li et al., 2025) addresses the distribution shift between the means of binarized and full-precision weights by iteratively refining the bias $\mu_{\text{refine}} = \mu + \frac{1}{m}\sum_{j=1}^{m} R_{\cdot j}$, the row scaling factor $\alpha_{\text{refine}} = \frac{1}{m}\text{diag}(B^\top (W - \mu_{\text{refine}}))$, and the binarized matrix $B_{\text{refine}} = \text{sign}(W - \mu_{\text{refine}})$.

Codebook Compression. Pruning is appealing in principle, but it often leads to accuracy degradation and non-trivial mask-index overhead. As noted in introduction, semi-structured pruning requires 0.25 mask bits per weight. Besides scalar quantization, vector quantize (Liu et al., 2024a; Van Baalen et al., 2024) employs a codebook to represent weights. To be specific, a weight $\mathbf{W}_{n \times m}$ is mapped in a codebook $\mathbf{C}_{c \times v}$ where v is the sub-vector length and c is the number of centroids. Now we need to store the codebook $\mathbf{C}_{c \times v}$ as well as the index assignments instead of the original weights. Since the codebook overhead can be ignored and the compression ratio can be calculated as $\lceil \log_2 c \rceil / (16 \cdot v)$ bits of weights index storage.

4 Methodology

As shown in Figure 4, we introduce BTC-LLM, a novel sub-1-bit LLM quantization method combines a Flash and Accurate Binary Codebook to capture repeated ± 1 patterns with a learnable incoherence-processing transform that reduce outilers and align weights to the codebook.

4.1 Flash and Accurate Binary Codebook

Binary Codebook. Existing vector quantization methods (Liu et al., 2024a; Van Baalen et al., 2024) are tailored for full-precision weights and are misaligned with the nature of binary weights, directly apply a sign function to full-precision codebooks resulting in significant errors. To address this mismatch, we introduce a binary-specific codebook tailored for compressing binarized weights.

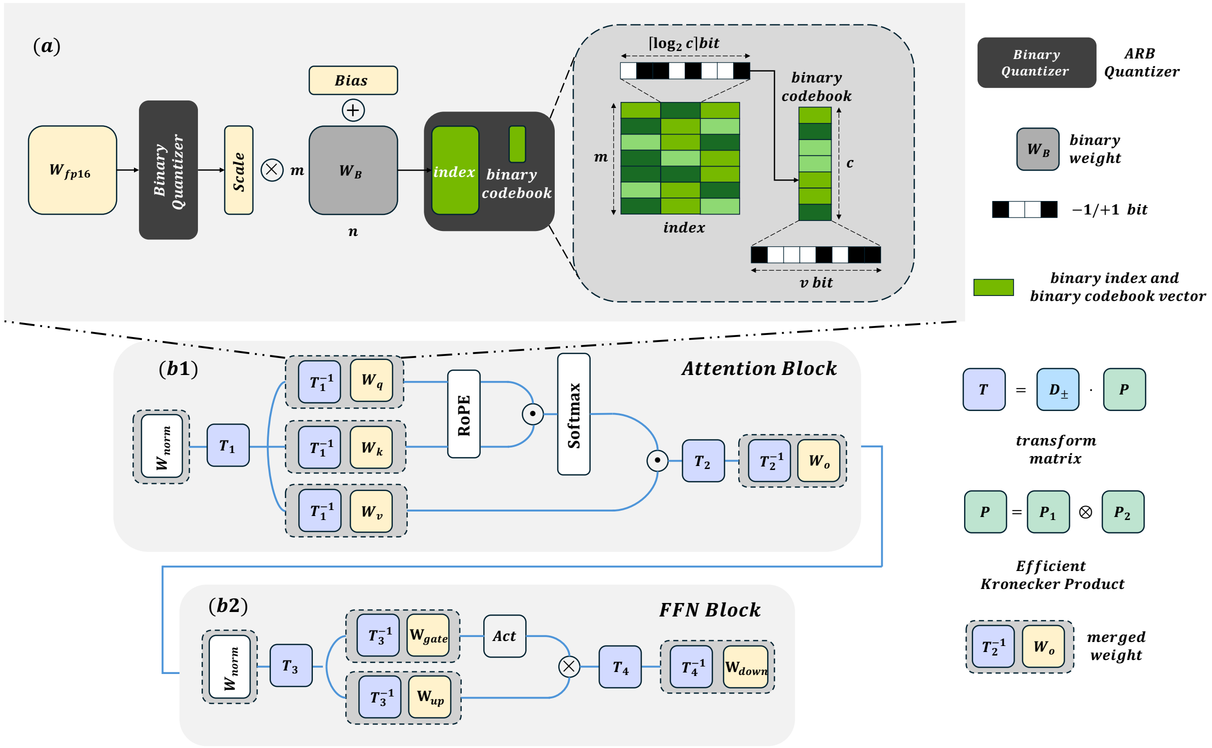

Figure 4: Overall architecture of BTC-LLM. **(a)** Sub-bit pipeline: the ARB quantizer transforms full-precision weights into binary form with associated scale and bias, followed by binary codebook representation and index assignment. **(b)** Structure of transformed attention (b1) and FFN (b2) blocks. The transform matrix is merged into the weights to ensure computational equivalence and efficiency.

Although both the codebook entries and weights are constrained to -1 and $+1$, finding the optimal codebook remains an NP-hard problem, detail refer to Appendix F. To solve this, we propose an efficient approximate optimization method inspired by the floating-point KMeans algorithm (Ikotun et al., 2023), combined with the feature of binary vectors distribution. The process consists of three main stages:

(1) Initialization: Given binary vectors $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N\}$ where $\mathbf{b}_i \in \{-1, +1\}^v$, we extract the set of unique vectors $\mathcal{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_M\}$ from $\mathbf{B}$. If $M \geq K$ (codebook size), we select the top-K most frequent vectors in $\mathcal{U}$ as the initial centroids $\mathcal{C}^{(0)} = \{\mathbf{c}_1^{(0)}, \ldots, \mathbf{c}_K^{(0)}\}$. Otherwise $M < K$, we set $\mathcal{C}^{(0)} = \mathcal{U}$ and let $K = M$.

(2) E-step Assignment: For each vector $\mathbf{b}_i$, we first test whether it is identical to any centroid $\mathbf{c}_k$; if so, we simply set $z_i = k$. Otherwise, we choose the nearest centroid via $z_i = \arg\min_k \left\|\mathbf{b}_i - \mathbf{c}_k\right\|_2^2$. Because every element is binary (± 1), the squared Euclidean distance reduces to a Hamming distance:

$$\left\|\mathbf{b} - \mathbf{c}\right\|_2^2 = \sum_j (b_j - c_j)^2 = \qquad (4)$$

$$4 \sum_j \left[b_j \neq c_j\right] = 4\, d_H(\mathbf{b}, \mathbf{c}), \qquad (5)$$

where $d_H(\mathbf{b}, \mathbf{c})$ counts the number of different elements. By packing the ± 1 entries into `int64`, the Hamming distance can be computed with one `XOR` $\rightarrow$ `POPCNT` instruction: $d_H(\mathbf{b}, \mathbf{c}) =$ `POPCNT` $(\mathbf{b} \oplus \mathbf{c})$ (Piao, 2022; Pham et al., 2025). Unlike reconstruction error-based metrics such as $\|X\mathbf{B} - X\hat{\mathbf{B}}\|_2^2$, this approach directly leverages the binary structure, avoiding costly matrix multiplications.

(3) M-step Centroid Update: For cluster k with assignment set $\mathcal{B}_k \subset \{\pm 1\}^L$, we update the binary centroid $\mathbf{c}_k \in \{\pm 1\}^L$ by solving:

$$\mathbf{c}_k = \operatorname{sign}\Big(\frac{1}{|\mathcal{B}_k|} \sum_{\mathbf{b}_i \in \mathcal{B}_k} \mathbf{b}_i\Big), \quad \operatorname{sign}(0) = +1.$$

This keeps the centroid still in binary.

After initialization, we alternate **E–step** and **M–step**: the E-step assigns each binary vector to its nearest codeword, yielding index $\mathbf{z}$; the M-step updates the codebook $\mathbf{C}$. Both steps are implemented with bit-packing and XNOR/POPCNT, rather than costly floating-point reductions. This full binarization approach bypasses the computationally expensive distance metrics inherent in full-precision K-means, and eliminates the need for complex variants such as Hessian-weighted K-means.

To fully exploit the efficiency of binary arithmetic, we propose **Binary Codebook LUT-GEMM**, following the lookup-table GEMM design in (Guo et al., 2024; Wei et al., 2024) to accelerate inference with binary weights. The key idea is to replace most multiply–accumulate operations with table lookups. Since our weights are stored as a binary codebook, only a finite set of binary patterns can appear, and these patterns are repeatedly reused across the matrix. Given an input activation, we first compute the dot products between the activation blocks and all codebook patterns once, and store these results in a small lookup table. Consequently, the GEMM operation is simplified to **a lookup-and-accumulate process**: we simply fetch the pre-computed results using the weight indices and sum them up. This design avoids any runtime dequant and significantly reduces arithmetic cost by maximally reusing the activation–pattern partial sums. More details refer to appendix G.

4.2 Learnable Transformation

Transformation Pair. To address the outlier issue in binarized LLM and align weights to the codebook, in this section, we propose learnable transform scheme to reduce quantization error. Specifically, we introduce two learnable parameters, $D_{\pm}$ and P, combining them into a transformation pair $T := D_{\pm}P$, where $D_{\pm} = \mathrm{diag}(\sigma)$ denote a diagonal sign matrix with $\sigma_i \in \{\pm 1\}$, being invertible, it performs channel-wise sign flips without changing magnitudes and P is a learnable invertible affine matrix. Following FlatQuant (Sun et al., 2024), we parameterize P as a Kronecker product $P = P_1 \otimes P_2$ and directly update the lightweight factors (P_1, P_2). Its inverse is computed via $P^{-1} = P_1^{-1} \otimes P_2^{-1}$, enable efficient online transformation.

Optimization detail. We learn the discrete sign flips $D_{\pm}$ using a straight-through estimator (STE), and apply a larger learning rate to $D_{\pm}$ for stable updates. The affine transform P is optimized with standard gradient descent. We optimize the transform parameters in a block-wise manner. For the l-th Transformer block, we solve

$$\min_{\mathbf{T}_l} \Big(\big|\big|\mathcal{F}_l(X) - \hat{\mathcal{F}}_l(X; T_l)\big|\big|_F^2 + \mathcal{L}_{\text{aux}} \Big), \qquad (6)$$

where $\mathcal{F}_l(\cdot)$ and $\hat{\mathcal{F}}_l(\cdot)$ denote the original and quantized block, and T_l collects the transformation parameters for that block.

To facilitate efficient representation via a compact codebook, we introduce $\mathcal{L}_{\text{aux}}$. By regularizing the Gram matrix, this term encourages binary weights to reuse the same sign patterns (see Figure 1). Specifically, stack the B binary vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_B\}$ into a row matrix $M \in \{\pm 1\}^{B \times v}$ and define the vector-similarity Gram matrix $G = \frac{1}{v} M M^{\top} \in \mathbb{R}^{B \times B}$. When vectors share similar sign patterns, the energy of G concentrates on its top-K eigenvalues $\{\lambda_i(G)\}_{i=1}^{K}$. We encourage this by minimizing $\mathcal{L}_{\text{sim}} = \mathrm{Tr}(G) - \sum_{i=1}^{K} \lambda_i(G)$. To avoid the collapse where all entries are $+1$ or -1, we add a global balance term that keeps the overall sign mean near zero, $\mathcal{L}_{\text{bal}} = \left(\frac{1}{Bv} \sum_{b=1}^{B} \sum_{\ell=1}^{v} M_{b,\ell} \right)^2$, and define the auxiliary objective $\mathcal{L}_{\text{aux}} = \lambda_1 \mathcal{L}_{\text{sim}} + \lambda_2 \mathcal{L}_{\text{bal}}$.

Integration into Transformer Architecture. For each linear layer $Y = XW^{\top}$, we apply the invertible transform T and reparameterize the weight as

$$Y = XW^{\top} \equiv (XT)\,(T^{-1}W^{\top}), \qquad (7)$$

which leaves the output unchanged in full precision and T^{-1} can be computed efficiently with kernel. We then binarize and compress only the reparameterized weights,

$$\hat{W} \;=\; \mathbf{Codebook}\big(\mathbf{B}(T^{-1}W^{\top})\big), \qquad (8)$$

where $\mathbf{B}(\cdot)$ is the binary quantizer and $\mathbf{Codebook}(\cdot)$ denotes codebook compression. During inference, we apply the transform T to activations on-the-fly (X ¸XT) and compute the output via LUT-based GEMM, leveraging the index from our binary codebook.

For the binary quantizer, we follow the binarization procedure described in ARB-LLM. Since the incoherence-processing transformation inherently incorporates activation information, we specifically adopt the naive ARB method rather than the ARB-RC or ARB-X variants for weight binarization which is faster and simpler.

4.3 Compression Analysis

As illustrated in Figure 4 (a), binary weights are compressed into a binary codebook and index mappings. Given an original weight matrix of shape $n \times m$, with a codebook of size c and vector length of v, the index requires $\lceil \log_2 c \rceil$ bits per vector, and each centroid occupies v bits. In Figure 4 (b), the transformation matrix can be fused into the model weights, incurring no additional storage overhead. Thus, the total storage cost is $vc + \lceil \log_2 c \rceil \cdot mn/v$. Since vc is relatively small and can be amortized, the effective compression ratio is approximately $16 \cdot v / \lceil \log_2 c \rceil$.

5 Experiments

5.1 Settings

Models, Datasets, and Baselines. We evaluate BTC-LLM on LLaMA-1/2/3 (AI@Meta, 2024) models ranging from 7B to 65B parameters. Performance is measured by WikiText2 perplexity and zero-shot accuracy on seven QA benchmarks: ARC-c/e (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OBQA (Mihaylov et al., 2018), RTE (Chakrabarty et al., 2021), and Winogrande (Sakaguchi et al., 2020). We compare strong PTQ baselines spanning vector and binary quantization, including VPTQ (Liu et al., 2024a), GPTVQ (Van Baalen et al., 2024), QuIP# (Tseng et al., 2024), BiLLM (Huang et al., 2024b), ARB-LLM (Li et al., 2025), and STBLLM (Dong et al., 2025).

5.2 Main Results on LLaMA Family

We observe in Table 1 that BTC-LLM consistently achieves the best perplexity on Wikitext2 across diverse quantization settings and model sizes. At 1.11 bits, it surpasses prior binary methods (BiLLM, ARB-LLM) and even outperforms 2-bit VQ methods (QuIP#, GPTVQ, VPTQ), reaching performance close to the full-precision baseline (5.47 $\rightarrow$ 6.06). Under aggressive settings (0.9–0.7 bits), BTC-LLM remains robust—matching 1.11-bit accuracy at 0.9 bits and still outperforming STBLLM by large margins (e.g., 6.60 vs. 13.06 at 0.8 bits), while VPTQ collapses.

Zero-Shot Results. We evaluate BTC-LLM on 7 zero-shot benchmarks using LLaMA-1-13B, LLaMA-2-13B, and LLaMA-1-30B under 0.80-bit settings. As shown in Table 2, BTC-LLM consistently outperforms STBLLM in all models, with gains of +4.7% and +5.0% on LLaMA-1-13B and LLaMA-2-13B, respectively. Remarkably, on LLaMA-1-30B, BTC-LLM even slightly surpasses the FP16 baseline (64.48 vs. 64.40), demonstrating strong robustness under aggressive compression. For more comprehensive results, please refer to Appendix Table 6.

5.3 Ablation Study

Extending to Pretrained Binary LLMs. Recent works such as BitNet (Wang et al., 2023) demonstrate the promise of training LLMs with binarized weights from scratch. Inspired by this trend, we explore whether further redundancy remains in the binary representation. Specifically, we extend our binary codebook compression to FBI-LLM (Ma et al., 2024a), a distilled, fully binarized LLM.

As shown in Table 4, compared to the original 1-bit FBI-LLM baseline, our codebook-based compression (FBI-LLM$_{\text{BC}}$) achieves comparable or even superior performance under more aggressive bit reductions. For example, at 0.80 bits, FBI-LLM$_{\text{BC}}$ improves the 1.3B model's mean accuracy from 43.02 to 43.49 with only a slight perplexity increase (14.41 $\rightarrow$ 18.23). Even at 0.50 bits, it maintains 39.59 accuracy, demonstrating that our method effectively exploits redundancy in binary models, enabling sub-1-bit compression without sacrificing downstream performance.

Effectiveness on Qwen Family Models. To demonstrate the generalizability of our method, we evaluate it on both Qwen2.5 and Qwen3 model families (Yang et al., 2024) across various model sizes. As shown in the Table 5, our sub-bit quantization consistently maintains strong performance across different bit-widths. Even at 1.11-bit and 0.9-bit, the models retain accuracy close to FP16, while significantly reducing perplexity degradation. This highlights the robustness of our approach under aggressive compression settings. Additional results on the Qwen family are provided in Appendix Table 7.

Memory, Latency. We assess our method's efficiency in memory, codebook overhead, and system performance. As shown in Table 3c, memory usage drops from 13.48 (FP16) to 0.65 at 0.7-bit, achieving an 20.7× compression. The codebook overhead is negligible (e.g., 1.2% at 0.7-bit), confirming its scalability. As shown in Figure 5, we evaluate kernel latency on an H800 GPU for an MLP layer of size 8,192×28,672. Here M = batch size × sequence length. For standard 1-bit weights, packing enables

Table 1: Perplexity results comparison on the LLaMA family.

Settings		**LLaMA-1**				**LLaMA-2**		**LLaMA-3**
Method	W-Bits	7B	13B	30B	65B	7B	13B	8B
FP16	16	5.68	5.09	4.1	3.53	5.47	4.88	6.14
QuIP#	2	6.86	5.97	5.02	4.36	6.66	5.74	-
GPTVQ	2.15	9.64	6.58	5.63	4.91	8.23	6.50	12.05
VPTQ	2	9.90	8.77	7.13	4.01	6.13	5.32	9.19
BiLLM	1.11	49.79	14.58	9.90	8.37	32.31	21.35	55.80
ARB-LLM	1.11	14.03	10.18	7.75	6.56	16.44	11.85	27.42
BTC-LLM	1.11	**6.23**	**5.53**	**4.59**	**3.94**	**6.06**	**5.29**	**7.70**
GPTVQ	0.90	206.19	47.08	26.12	12.33	100.81	82.34	1.3e3
VPTQ	0.90	2e4	8.8e3	2.3e3	1.1e3	2.3e4	5.0e3	9.5e5
BTC-LLM	0.90	**6.24**	**5.56**	**4.63**	**4.03**	**6.07**	**5.32**	**7.84**
GPTVQ	0.80	667.55	131.72	68.85	32.56	264.35	201.67	1e5
VPTQ	0.80	2.4e3	9.2e3	3.2e3	1.2e3	2.2e5	6.3e3	1.6e5
STBLLM	0.80	15.03	9.66	7.56	6.43	13.06	11.67	33.44
BTC-LLM	0.80	**6.72**	**6.01**	**5.29**	**4.74**	**6.60**	**5.83**	**9.49**
GPTVQ	0.70	1.4e3	933.55	261.77	61.52	803.44	640.95	1.8e5
VPTQ	0.70	2.9e5	1.4e5	4.8e3	1.4e3	1.9e5	9.4e3	2.7e5
STBLLM	0.70	19.48	11.33	9.19	7.91	18.74	13.26	49.12
BTC-LLM	0.70	**10.72**	**9.01**	**7.80**	**6.61**	**11.02**	**8.76**	**18.54**

Table 2: Accuracies (%) for 7 zero-shot tasks from sub-bit binarized LLaMA family with STBLLM and BTC-LLM.

Models	**Method**	**W-Bits**	**Winogrande**	**OBQA**	**Hellaswag**	**Boolq**	**ARC-e**	**ARC-c**	**RTE**	**Average**
LLaMA-1-13B	FP16	16	72.69	33.20	59.91	77.89	77.40	46.42	70.40	63.80
	STBLLM	0.80	65.98	36.20	63.67	65.38	68.86	34.04	56.68	55.83
	BTC-LLM	0.80	**70.8**	**41.6**	**72.48**	**74.86**	**67.8**	**42.24**	**55.96**	**60.82**
LLaMA-1-30B	FP16	16	75.77	36.00	63.37	82.69	80.30	52.90	67.15	67.40
	STBLLM	0.80	71.59	41.00	69.85	77.37	71.55	41.3	48.01	60.10
	BTC-LLM	0.80	**76.07**	**45.0**	**76.07**	**71.71**	**73.99**	**45.39**	**66.06**	**64.48**
LLaMA-2-13B	FP16	16	72.22	35.20	60.03	80.55	79.42	48.38	65.34	65.00
	STBLLM	0.80	63.93	37.00	57.76	71.53	60.56	31.99	54.15	53.85
	BTC-LLM	0.80	**69.46**	**71.53**	**72.63**	**71.53**	**70.75**	**42.75**	**64.62**	**61.91**

efficient loading and reuse in shared memory. Since $\pm 1 \times a$ operation is implemented as simple addition or subtraction, the kernel shifts from being bandwidth-bound to compute-bound, allowing our custom W1A16 GEMM to outperform the native PyTorch baseline. In the sub-1-bit regime, we evaluate the proposed Binary Codebook LUT-GEMM. By completely bypassing the dequantization step, this method achieves a 1.6× speedup. We note that these results are based on a preliminary kernel implementation, suggesting significant potential for further performance optimization.

Activation Quantization on sub-bit LLMs. We introduce a transformation that suppresses outliers and improves activation quantization efficiency, thereby accelerating inference (Microsoft, 2023). As shown in Table 3d, the W0.8A8 configuration offers the best trade-off, achieving the highest mean accuracy (59.6%) with low perplexity, compared to W0.8A16 (58.46%) and W0.8A4 (55.74%). More results are provided in Appendix Table 6.

Codebook Vector Length. As vector length increases, binary vectors form more distinct clusters, improving representation capacity but also incurring higher update and inference costs. Table 3a shows that a vector length of 20 already matches the performance of the 1.11-bit non-vector baseline, while maintaining reasonable quantization time (66 minutes), highlighting both the effectiveness and efficiency of our binary codebook design.

Ablation for Transformation Components. We ablate the learned transform by progressively

Table 3: Ablation study on LLaMA-2-7B across WikiText2 and 7 zero-shot tasks.

(a) Study of Codebook Vector Length (vector length / centroids) under 0.8bit

Vector length	1.11bit	v4c9	v8c85	v10c256	v12c777	v14c2353	v16c7132	v18c21619	v20c65536
WikiText2 ↓	6.06	39.97	17.58	14.00	11.68	8.75	6.60	6.12	6.06
mean acc ↑	61.84	36.52	41.15	42.77	45.62	49.84	58.46	60.79	61.84
quant time(min)	36	43	44	46	46	52	56	61	66

(b) Study of Learned Transform

Method	WikiText2 ↓	mean acc ↑
no	9.23	49.54
P	6.95	55.64
$P + D_{\pm}$	6.60	58.46

(c) Study of Memory and codebook overhead

Method	Model Mem	Codebook Mem(overhead)
FP16	13.48GB	-
0.9bit	0.84GB	77.47MB(9.2%)
0.8bit	0.74GB	25.56MB(3.4%)
0.7bit	0.65GB	8.43MB(1.2%)

(d) Study of Activation Quantization

Method	WikiText2 ↓	mean acc ↑
LLaMA-2-7b W0.8A16	6.60	58.46
LLaMA-2-7b W0.8A8	6.61	59.60
LLaMA-2-7b W0.8A4	7.20	55.74

(e) Study of Number of Split Points

Method	WikiText2 ↓	mean acc ↑
LLaMA-2-7b 0.8bit 1 Split Point	10.12	49.18
LLaMA-2-7b 0.8bit 2 Split Point	6.60	58.46
LLaMA-2-7b 0.8bit 3 Split Point	6.13	61.11

Figure 5: Latency, memory usage, and accuracy under sub-1-bit quantization on LLaMA-2-7B.

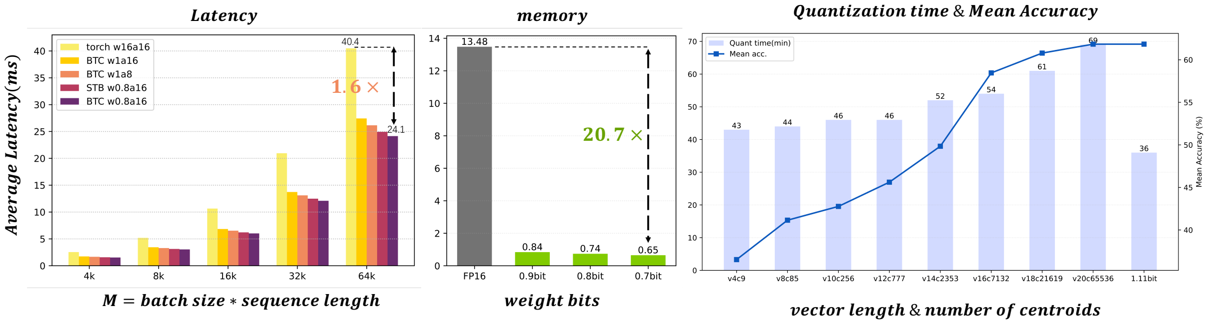

Table 4: Results of FBI-LLM with our binary codebook (FBI-LLM$_{BC}$).

Settings		130M		1.3B	
Method	Bits	WikiText2 PPL	Mean Acc	WikiText2 PPL	Mean Acc
Original	1.00	31.56	39.42	14.41	43.49
FBI-LLM$_{BC}$	0.80	34.99	39.30	18.23	43.02
FBI-LLM$_{BC}$	0.70	38.29	39.19	19.02	41.48
FBI-LLM$_{BC}$	0.50	48.13	39.07	20.91	39.59

Table 5: Implementation on Qwen Family Models (WikiText2 ppl / mean accuracy)

Model	Qwen2.5-3b	Qwen2.5-14b	Qwen3-8b	Qwen3-14b
FP16	8.03/65.24	5.29/72.25	9.72/69.47	8.64/72.71
1.11bit	9.75/62.77	6.49/72.79	11.60/65.45	12.05/66.53
0.9bit	9.85/59.8	6.58/71.5	11.70/65.53	12.93/62.65
0.8bit	11.26/55.88	7.42/67.73	13.12/62.11	14.05/60.71
0.7bit	18.71/46.48	12.28/56.98	15.87/59.00	16.11/58.23

adding components. As shown in Table 3b, using only the P component alleviates outliers and already outperforms the naive baseline. By incorporating $D_{\pm}$, $P + D_{\pm}$ variant induces more pronounced clustering among binary vectors. For a fixed bit-rate, this enhanced pattern redundancy allows the codebook to capture weight distributions more accurately, reaching 6.60 perplexity and 58.46% accuracy.

Ablation for Number of Split Point. We adopt a grouping strategy to quantize non-salient weights using a split point p (Li et al., 2025; Huang et al., 2024b), which controls their partitioning. Varying the number of split points affects model performance. As shown in Table 3e, using two split points (as in STBLLM) improves mean accuracy from 49.18% to 58.46%, while three split points further boost it to 61.11%, confirming the effectiveness of this approach.

6 Conclusions

We present BTC-LLM, a sub-1-bit compression framework for LLMs. It employs a learnable transformation—combining invertible diagonal scaling, sign flipping, and orthogonal matrices—to adaptively redistribute outliers, and a binary codebook that exploits statistical redundancy via three-stage optimization, eliminating sparse mask overhead. Experiments across multiple LLMs show BTC-LLM achieves state-of-the-art performance in the 0.7–1.11 bit range. While activations can be quantized, our treatment of ultra-low-bit KV cache remains preliminary (see Appendix E). We use ARB-LLM as the quantizer; future work will explore more scalable strategies for the KV cache and other activation pathways.

References

Unsloth AI. 2026. unsloth/kimi-k2.5 - hugging face model. https://huggingface.co/unsloth/Kimi-K2.5-GGUF. Accessed: 2026-04-15.

AI@Meta. 2024. Llama 3 model card.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240.

Tuhin Chakrabarty, Debanjan Ghosh, Adam Poliak, and Smaranda Muresan. 2021. Figurative language in recognizing textual entailment. *arXiv preprint arXiv:2106.01195*.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2025. Efficientqat: Efficient quantization-aware training for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10081–10100.

Pengyu Cheng, Jiacheng Wang, Tianle Chen, Bei Liu, Xiaofeng Hou, and Jiacheng Liu. 2026. Desirekv: Decoupling sensitivity and importance for reasoning-aware kv cache compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 20518–20526.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *ACL*, pages 2924–2936.

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

Peijie Dong, Lujun Li, Yuedong Zhong, Dayou Du, Ruibo Fan, Yuhan Chen, Zhenheng Tang, Qiang Wang, Wei Xue, Yike Guo, and 1 others. 2025. Stbllm: Breaking the 1-bit barrier with structured binary llms. In *ICLR*.

Hao Gu, Wei Li, Lujun Li, Qiyuan Zhu, Mark Lee, Shengjie Sun, Wei Xue, and Yike Guo. 2025. Delta decompression for moe-based llms compression. *arXiv preprint arXiv:2502.17298*.

Hao Gu, Hao Wang, Jiacheng Liu, Lujun Li, Qiyuan Zhu, Bei Liu, Binxing Xu, Lei Wang, Xintong Yang, Sida Lin, and 1 others. 2026. Qarl: Rollout-aligned quantization-aware rl for fast and stable training under training–inference mismatch. *arXiv preprint arXiv:2604.07853*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric Xing, and Yoon Kim. 2024. Fast matrix multiplications for lookup table-quantized llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12419–12433.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024a. Billm: Pushing the limit of post-training quantization for llms. *ICML*.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024b. Billm: Pushing the limit of post-training quantization for llms. *arXiv preprint arXiv:2402.04291*.

Abiodun M. Ikotun, Absalom E. Ezugwu, Laith Abualigah, Belal Abuhaija, and Jia Heming. 2023. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210.

Lujun Li, Qiyuan Zhu, Jiacheng Wang, Xiaoyu Qin, Wei Li, Hao Gu, Sirui Han, and Yike Guo. 2026. Sub-moe: Efficient mixture-of-expert llms compression via subspace expert merging. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 22994–23002.

Wei Li, Lujun Li, You-Liang Huang, Mark G Lee, Shengjie Sun, Wei Xue, and Yike Guo. Structured mixture-of-experts llms compression via singular value decomposition.

Zhiteng Li, Xianglong Yan, Tianao Zhang, Haotong Qin, Dong Xie, Jiang Tian, Linghe Kong, Yulun Zhang, Xiaokang Yang, and 1 others. 2025. Arb-llm: Alternating refined binarizations for large language models. In *ICLR*.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. 2024a. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. In *NeurIPS*.

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

Jiacheng Liu, Peng Tang, Wenfeng Wang, Yuhang Ren, Xiaofeng Hou, Pheng Ann Heng, Minyi Guo, and Chao Li. 2026. A survey on inference optimization techniques for mixture of experts models. *ACM Computing Surveys*, 58(10):1–37.

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. 2024a. Vptq: Extreme low-bit vector post-training quantization for large language models. *arXiv preprint arXiv:2409.17066*.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024b. Llm-qat: Data-free quantization aware training for large language models. In *ACL*.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. 2018. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*.

Liqun Ma, Mingjie Sun, and Zhiqiang Shen. 2024a. Fbi-llm: Scaling up fully binarized llms from scratch via autoregressive distillation. *arXiv preprint arXiv:2407.07093*.

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. 2024b. Affinequant: Affine transformation quantization for large language models. *arXiv preprint arXiv:2403.12544*.

Microsoft. 2023. Bitblas: A high-performance BLAS library for quantized matrix multiplication. `https://github.com/microsoft/BitBLAS`. Accessed: 2024-03-01.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

OpenAI. 2024. Hello GPT-4o.

Van-Khoa Pham, Lai Le, Thanh-Kieu Tran Thi, and 1 others. 2025. Xnor-popcount, an alternative solution to the accumulation multiplication method for approximate computations, to improve latency and power efficiency. *Journal of Technical Education Science*, 20(01):12–20.

Tairen Piao. 2022. XNOR-popcount-GEMM-PyTorch-CPU-CUDA: A PyTorch implementation of real XNOR-popcount (1-bit op) GEMM Linear PyTorch extension. `https://github.com/tairenpiao/XNOR-popcount-GEMM-PyTorch-CPU-CUDA`. Accessed: 2025-05-15.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016a. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Joseph. 2016b. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. In *ICLR2024 Spotlight*.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, and 1 others. 2024. Flatquant: Flatness matters for llm quantization. *arXiv preprint arXiv:2410.09426*.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*.

Mart Van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. Bitnet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, and 1 others. 2022. Emergent abilities of large language models. *Transactions on Machine Learning Research*.

Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. 2024. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. *Preprint*, arXiv:2407.00088.

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

Binxing Xu, Hao Gu, Lujun Li, Hao Wang, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Xintong Yang, Chao Li, Sirui Han, and 1 others. 2026. Bit-by-bit: Progressive qat strategy with outlier channel splitting for stable low-bit llms. *arXiv preprint arXiv:2604.07888.*

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115.*

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *ACL.*

Appendix

In the appendix, we include further discussions on the broader implications of our work, additional experimental results, implementation details, and pseudocode to facilitate reproducibility.

A Ethics statement

We acknowledge and adhere to the ICLR Code of Ethics. We have carefully considered the ethical implications of our research and paper submission. Our work does not involve human subjects, and it does not make use of data sets that could raise privacy or security concerns. We have ensured that our methodology and applications do not introduce or perpetuate harmful biases, and we have taken care to document our data sources and experimental procedures to promote transparency and reproducibility. We have no known conflicts of interest or sponsorship to disclose.

B Reproducibility statement

All experiments follow standard setups with results reported from three repetitions. Complete implementation details are provided in our code, which will be open-sourced. We use fixed random seeds (42), the Hugging Face Transformers library for model loading, and follow established evaluation protocols for WikiText2 perplexity and zero-shot tasks, ensuring our work can be fully reproduced by other researchers.

C Extended Discussion

C.1 The Use of Large Language Models (LLMs)

A large language model was utilized for grammatical and stylistic refinement of the manuscript. Its role was strictly limited to text editing and polishing to enhance clarity. All research ideas, experimental design, and analytical content are the original work of the authors.

C.2 Broader impacts

Our work on BTC-LLM is primarily a technical approach applied to publicly available models and is not designed to have specific ethical or moral implications. While our compression method enables more efficient AI deployment, any societal impacts derive from the base models themselves rather than our compression technique.

C.3 Limitations

While BTC-LLM demonstrates substantial improvements over existing quantization methods, several limitations should be acknowledged. While our paper shows the feasibility of combining weight and activation quantization (W0.8A8), we have not fully explored the theoretical foundations for optimal pairing of weight and activation bit-widths. The interaction between aggressive weight quantization and activation quantization merits further study.

Our current approach does not address the compression of the KV cache (Cheng et al., 2026; Liu et al., 2026), which can dominate memory usage during long-context inference. Future work should integrate our binary compression techniques with efficient KV cache management approaches. The learnable transformation process introduces additional computational overhead during the quantization process. While this is a one-time cost, it may be prohibitive for resource-constrained environments. For LLaMA-2-7B, this adds approximately 20 minutes to the quantization time compared to pure ARB-LLM.

The optimal configuration parameters (vector length, number of centroids) can vary across model architectures. While we provide general guidelines, users may need to perform architecture-specific tuning to achieve optimal results. Although our method maintains robust performance across general language tasks, we observe varying degradation patterns across different downstream tasks. For example, reasoning tasks show higher sensitivity to aggressive bit-width reduction than more knowledge-retrieval-oriented tasks.

Furthermore, our primary motivation for exploring sub-1-bit quantization stems from the deployment challenges of trillion-parameter Mixture-of-Experts (MoE) models, such as Kimi-K2. Compressing MoEs (Gu et al., 2025; Li et al., 2026) at this massive scale is highly crucial, as empirical evidence suggests that larger models exhibit a surprising resilience to aggressive quantization. While recent efforts (e.g., Unsloth (AI, 2026)) have successfully pushed the boundary to approximately 1.6-bit quantization, evaluating sub-1-bit techniques at the trillion-parameter scale remains unexplored in this paper. Therefore, scaling and validating our quantization approach on these ultra-large MoE models will be the primary focus of our future work.

Algorithm 1 Binary Vector Processing

```
function WEIGHT_TO_VECTOR($\mathbf{B}$, $v$)
    Extract non-zero elements from $\mathbf{B}$
    Pad with alternating $+1/-1$ to ensure divisibility by $v$
    Reshape to form vectors of length $v$
    return Vectors of shape $[N, v]$
end function
function VECTOR_TO_WEIGHT($\mathbf{V}$, $\mathbf{B}$)
    Create mask of non-zero positions in $\mathbf{B}$
    Flatten vectors and remove padding
    Place vector elements back into original positions
    return Reconstructed binary matrix
end function
```

Algorithm 2 Efficient Binary Vector Packing and Unpacking

```
function WEIGHT_TO_VECTOR($\mathbf{B}$, $v$)
    Extract indices of non-zero entries: idx $\leftarrow \{(i,j) \mid \mathbf{B}_{i,j} \neq 0\}$
    Extract binary values and map: $b \leftarrow (\mathbf{B}_{\text{idx}} + 1)/2 \in \{0,1\}$
    Pad $b$ with 0/1 alternately to make length divisible by $v$
    Reshape $b$ to bit-vectors: $\mathbf{V}_{\text{bit}} \in \{0,1\}^{N \times v}$
    return $(\mathbf{V}_{\text{bit}}, \text{idx})$
end function
function VECTOR_TO_WEIGHT($\mathbf{V}_{\text{bit}}$, idx)
    Flatten bits: $b \leftarrow \text{reshape}(\mathbf{V}_{\text{bit}}, [-1])$
    Remove padding to match len(idx)
    Map bits back: $\mathbf{B}_{\text{idx}} \leftarrow 2 \cdot b - 1$
    Fill remaining entries in $\mathbf{B}$ with zeros
    return $\mathbf{B}$
end function
```

D Implementation Details and Pseudocode

D.1 Binary Vector Processing

Our binary codebook compression approach is implemented through an efficient algorithm that leverages the unique characteristics of binary weights. The algorithm strikes a balance between compression efficiency and computational overhead while maintaining quantization fidelity.

The first step in our approach involves processing the binary weight matrix for efficient codebook generation refer in pseudocode 2.

D.2 Binary Codebook Optimization

The core of our approach is an EM-based algorithm optimized specifically for binary weights. Hamming distance can be calculated by $d_H(\mathbf{b}, \mathbf{c}) = \texttt{POPCNT}(\mathbf{b} \oplus \mathbf{c})$, and sign base centroid update can be accelerated by `POPCNT`, `PCMPGTB`.

D.3 Efficient Implementation Details

Our implementation incorporates several optimizations specifically for binary weights:

1. **Early termination**: For cases where the number of unique vectors is less than or equal to the codebook size, we achieve perfect reconstruction with exact vector matching in a single iteration.

2. **Efficient centroid updates**: Unlike traditional k-means requiring reconstruction for each update, our method directly computes means and applies the sign function to maintain binary constraints.

3. **Vectorized operations**: We leverage PyTorch's efficient tensor operations like `scatter_add_` and `bincount` to accelerate cluster assignment and centroid updates.

4. **Binary-specific distance metric**: Distance calculations between binary vectors utilize squared Euclidean distance, which is more efficient than computing full reconstruction error.

D.4 Complete Binary Transformation and Compression

Our complete binary transformation and compression (BTC) approach combines learned transformations with binary codebook compression refer to pseudocode 3.

Algorithm 3 Binary Transformation and Compression

```
function BTC($\mathbf{W}$, $[\mathbf{R}, \mathbf{s}, \mathbf{d}]$)
    Apply transformation: $\mathbf{W} \leftarrow \mathrm{diag}(\mathbf{s} \odot \mathbf{d})^{-1} \cdot \mathbf{R}^\top \cdot \mathbf{W}$
    Binarize weights: $\alpha, \mathbf{B}, \mu \leftarrow \mathrm{ARB}(\mathbf{W})$
    Generate codebook: $\mathrm{idx}, \mathbf{C} \leftarrow$ BINARYCODEBOOK($\mathbf{B}$)
    Reconstruct binary: $\hat{\mathbf{B}} \leftarrow \mathbf{C}[\mathrm{idx}]$
    Dequantize: $\hat{\mathbf{W}} \leftarrow \alpha \cdot \hat{\mathbf{B}} + \mu$
    return $\hat{\mathbf{W}}$
end function
```

This approach achieves a compression ratio of approximately $16 \cdot v / \lceil \log_2 c \rceil$, providing significant memory savings while maintaining model quality through tailored binary-specific optimization methods.

E Future Work on Activation and KV cache Quantization

Activation quantization reduces memory transfer overhead and leverages efficient low-precision compute units. Moreover, we observe substantial redundancy in the KV cache, enabling aggressive low-bit quantization. We further implement KV cache quantization to exploit this potential. First, we redesign the saliency metric for the binary quantizer. Since the KV cache exhibits a shift in window importance, we assign higher salient weights to local windows. To avoid dequantization overhead from extreme codebook compression, we preserve local windows binary representation without sub-bit quantization.

Given the need for on-the-fly quantization and dequantization in KV cache compression, developing simpler and more computationally efficient quantizers remains an important direction for future research. Inspired by Binarized Neural Networks (BNNs) in convolutional architectures, where activations are also quantized to binary, we aim to further explore fully binarized LLMs with binary activations.

F Binary codebook Analysis

Finding the optimal binary codebook is **NP-hard**, as it reduces to a special case of the well-known **k-means clustering** problem, which is NP-hard when the number of clusters $K \geq 2$ and vector dimension $D \geq 2$.

In our setting, each codebook vector is constrained to binary values $\{-1,+1\}^D$, and the goal is to choose K such vectors to minimize the total reconstruction error. This requires searching over all possible combinations of K vectors from a space of 2^D candidates, yielding:

Search space size= $\binom{2^D}{K}$, and total complexity: $O\left(\binom{2^D}{K}\cdot N\cdot K\cdot D\right)$,

where N is the number of weight vectors being quantized. This combinatorial explosion makes the global optimum intractable even for moderate D, a hallmark of NP-hard problems.

G Binary Codebook LUT-GEMM.

To enable fast multiplication with binary compressed weights, we represent each 1-bit weight matrix $W \in \pm 1^{m\times n}$ using a binary codebook and an index matrix. Concretely, we partition the input dimension into subvectors of length v (with $v \mid n$), and store (i) a binary codebook $C \in \pm 1^{c\times v}$ and (ii) an index matrix $I \in 0,\dots,c-1^{m\times(n/v)}$ such that each block of weights is selected by an index:

$$W_{r,;jv:(j+1)v};=;C_{,I_{r,j}}, \qquad j=0,\dots,\frac{n}{v}-1.$$

Given an activation vector $x \in \mathbb{R}^n$, the output can be written as a sum over blocks

$$y_r;=;\sum_{j=0}^{n/v-1}\left\langle x_j,;C_{I_{r,j}}\right\rangle, \quad x_j := x_{jv:(j+1)v}.$$

A naive implementation would dequantize each selected codebook entry and perform v multiply-adds per block. Instead, we propose a two-stage lookup scheme that converts the computation into **lookup + accumulation**, amortizing the cost of codebook evaluation across many output rows.

Stage-I (activation LUT). We further split each block $x_j \in \mathbb{R}^v$ into $P = v/\mu$ segments of length μ (e.g., $\mu \in 4,8$). For each segment $x_{j,p} \in \mathbb{R}^\mu$, we build a small signed-sum table

$$\mathrm{LUT}_{j,p}[s] = \sum_{t=1}^{\mu}\sigma_t(s), x_{j,p}[t] \qquad (9)$$

$$s \in 0,\dots,2^\mu-1, \sigma_t(s) \in \pm 1, \qquad (10)$$

where the table index s encodes a μ-bit ± 1 pattern. This table is activation-dependent but **shared** for all output rows.

Stage-II (codebook LUT). We precompute, **offline**, the μ-bit pattern key of each codebook entry in each segment. Specifically, for every $k \in [0,c)$ and segment $p \in [0,P)$, we store a compact key $\mathrm{key}[k,p] \in [0,2^\mu)$ that encodes $C_k[p\mu : (p+1)\mu] \in \pm 1^\mu$. At runtime, the dot-product between activation block x_j and a codebook entry C_k is obtained without multiplications:

$$\mathrm{CBLUT}_j[k] = \langle x_j, C_k\rangle = \sum_{p=0}^{P-1}\mathrm{LUT}_{j,p}\big[\mathrm{key}[k,p]\big].$$

Finally, the GEMM reduces to an index-gather accumulation:

$$y_r = \sum_{j=0}^{n/v-1}\mathrm{CBLUT}_j\left[I_{r,j}\right],$$

i.e., each block contributes one table lookup and one addition per output row.

Complexity and implementation. Per block j, the proposed kernel costs $O(P\cdot 2^\mu)$ to build $\mathrm{LUT}_{j,p}$, $O(c\cdot P)$ to build CBLUT_j, and $O(m)$ for index-gather accumulation—replacing the naive $O(m\cdot v)$ multiply-adds. In practice, CBLUT_j is computed once and reused by a large tile of output rows (large m, making the amortized overhead small. We place $\mathrm{LUT}_{j,p}$ and CBLUT_j in shared memory (or registers when c is small), and optionally replicate CBLUT_j across warps to mitigate shared-memory bank conflicts. For throughput, we also pack two FP16 entries (e.g., `half2`) to halve the number of table loads when accumulating consecutive indices.

H Full Results

H.1 Quantitative Results

In this section, we provide a comprehensive presentation of our results across various datasets to complement the main paper. Specifically, the results include: Complete comparison of the perplexity score on WikiText2 and averaged accuracy on zero-shot common sense reasoning tasks on LLaMA Model Family in Table 6 and Qwen Model Family in Table 7. And validate the effectiveness the activation quantization and KV cache quantization of BTC-LLM.

H.2 Visualization Results

Figure 6 and Figure 7 illustrate the relative quantization error between quantized and full-precision weights for BTC-LLM, ARB-LLM, and BiLLM, highlighting the improved accuracy of BTC-LLM. In contrast, Figure 8 and Figure 9 visualize the activation distributions across different layers of LLaMA-2-7B before and after applying BTC-LLM, showing how our method suppresses outliers and promotes a more compact activation range.

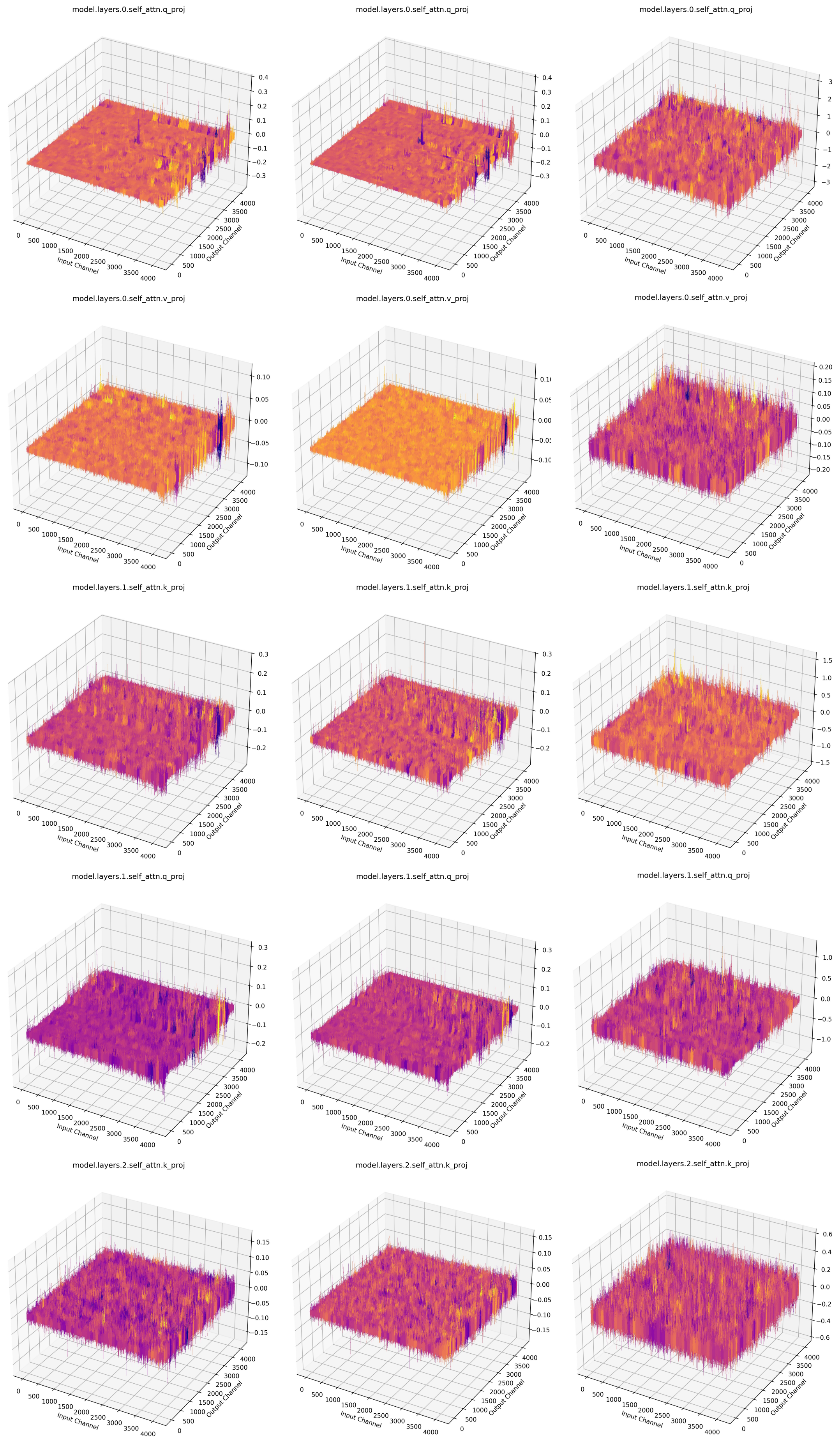

Figure 6: Visualizations comparing of the weight relative quantize error of LLaMA-2-7B with BTC-LLM (1st column), ARB-LLM(2nd column), and BiLLM (3rd column), respectively.

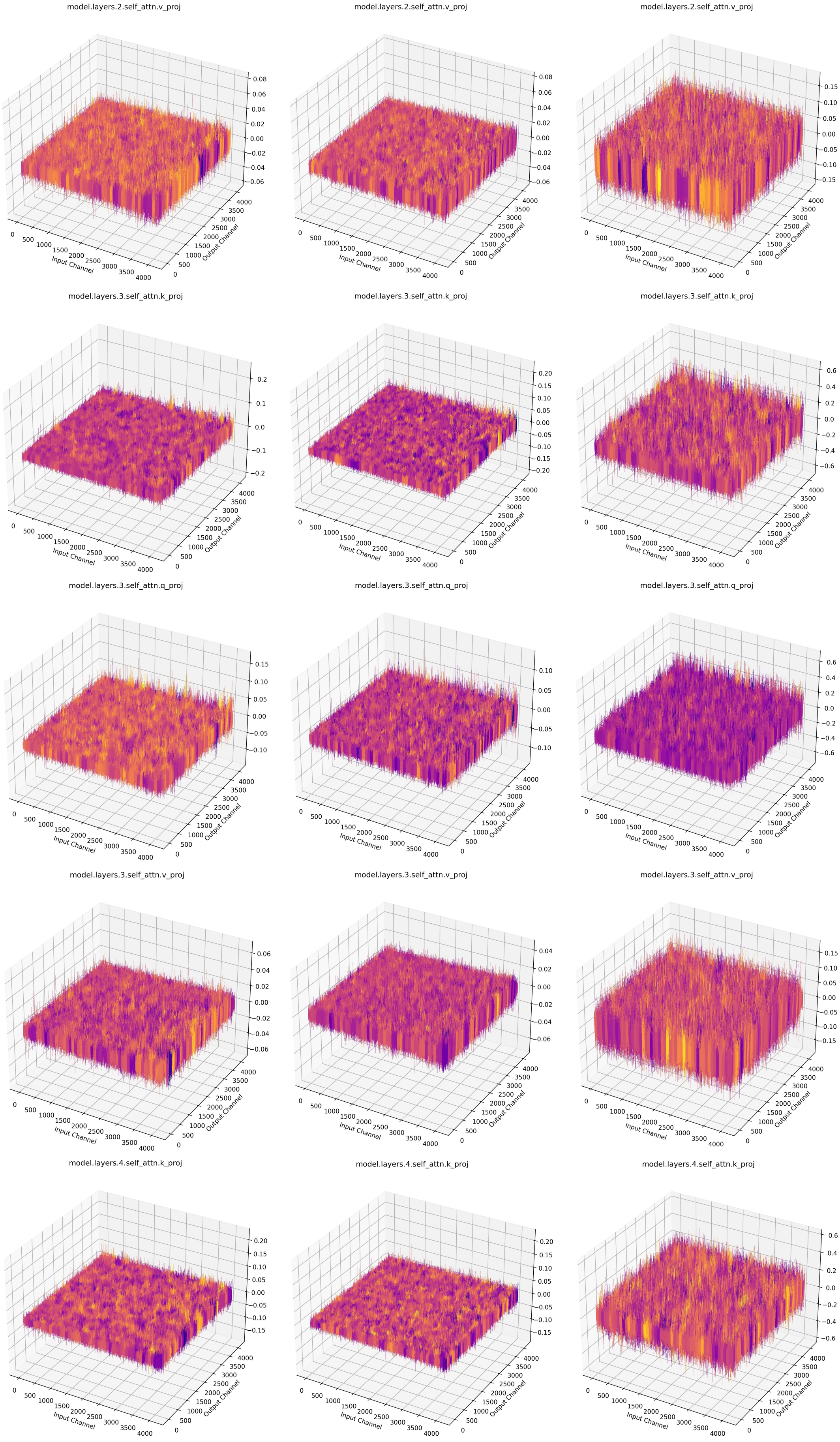

Figure 7: Visualizations comparing of the weight relative quantize error of LLaMA-2-7B with BTC-LLM (1st column), ARB-LLM(2nd column), and BiLLM (3rd column), respectively.

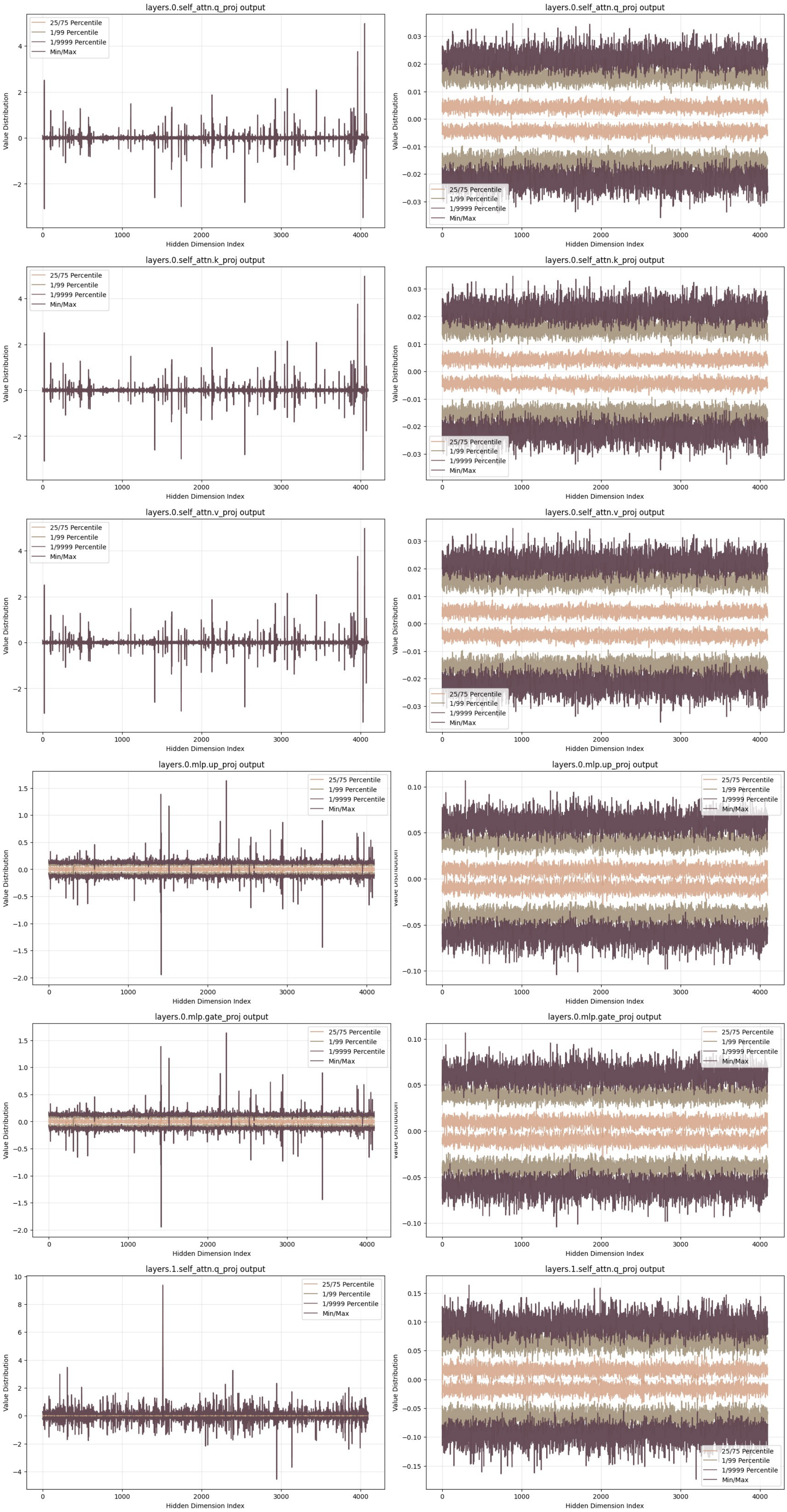

Figure 8: Visualizations of the activation distribution of different layers in LLaMA-2-7B before and after BTC-LLM. Left original activation, Right BTC-LLM activation.

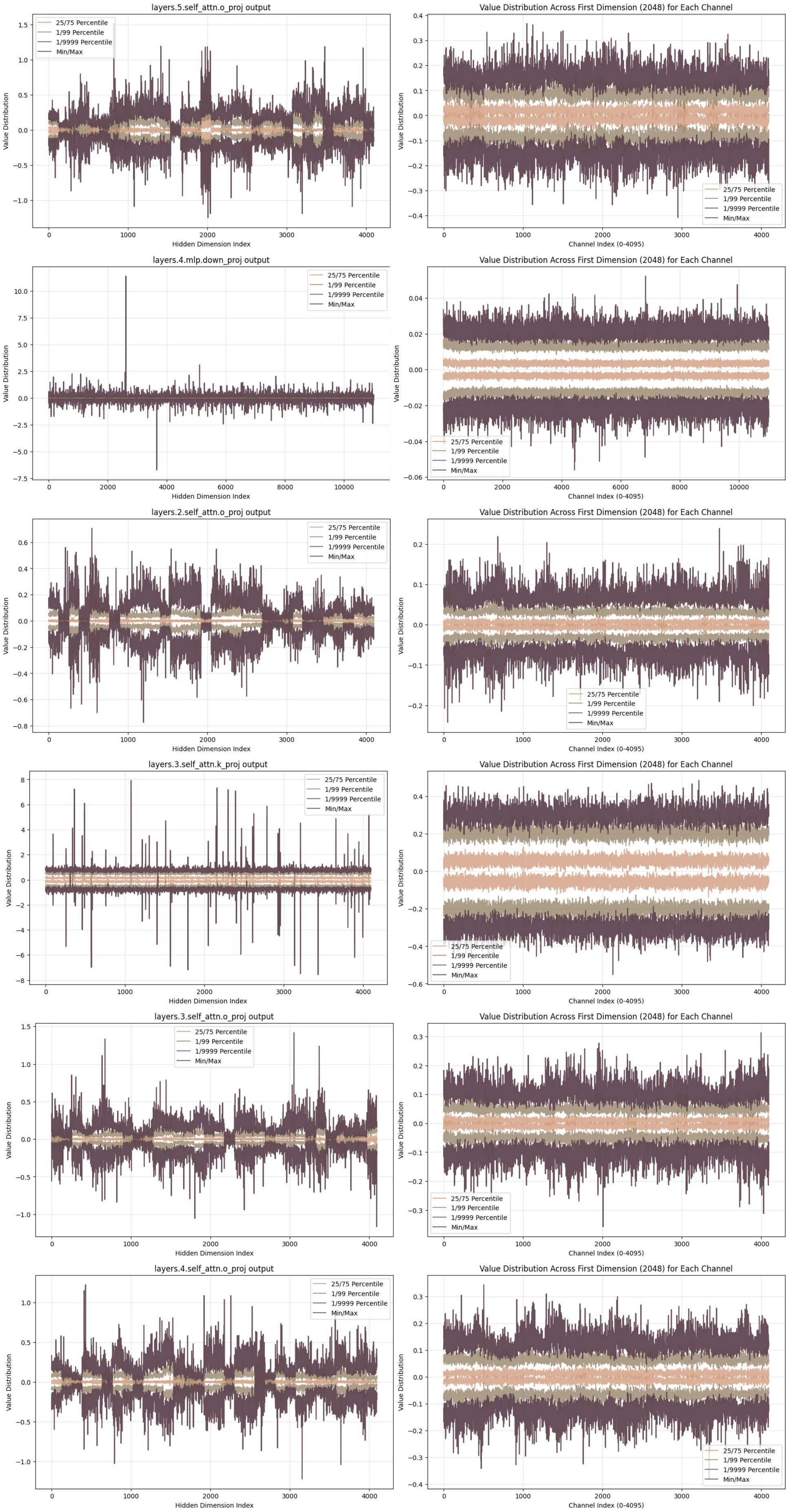

Figure 9: Visualizations of the activation distribution of different layers in LLaMA-2-7B before and after BTC-LLM. Left original activation, Right BTC-LLM activation.

Table 6: Complete comparison of the perplexity score on WikiText2 and averaged accuracy on Zero-shot Common Sense Reasoning tasks for LLaMA Model Family

Models	Method	#Bits W-A-KV	Winogrande	OBQA	Hellaswag	Boolq	ARC-e	ARC-c	RTE	Average	WikiText2
LLaMA-1-7B	FP16	16-16-16	69.93	43.80	76.20	74.98	72.90	44.71	67.15	64.37	5.68
	BTC-LLM	1.11-16-16	68.98	40.6	71.49	73.79	68.6	40.87	63.9	61.18	6.23
	BTC-LLM	0.90-16-16	68.9	74.4	71.44	74.4	69.65	40.53	60.29	60.86	6.24
	BTC-LLM	0.80-16-16	67.4	69.02	68.79	69.02	64.6	37.97	50.18	56.71	6.72
	BTC-LLM	0.70-16-16	56.99	31.2	49.64	63.49	46.46	27.22	53.43	46.92	10.72
LLaMA-1-13B	FP16	16-16-16	72.69	33.20	59.91	77.89	77.40	46.42	70.40	63.80	5.09
	BTC-LLM	1.11-16-16	72.77	43.2	75.57	75.5	72.94	44.8	67.87	64.66	5.53
	BTC-LLM	0.90-16-16	71.43	44.4	75.12	77.34	72.94	43.94	69.31	64.93	5.56
	BTC-LLM	0.80-16-16	67.4	69.02	68.79	69.02	64.6	37.97	50.18	60.82	6.01
	BTC-LLM	0.70-16-16	63.54	33.6	54.75	66.51	54.17	30.8	52.71	50.87	9.01
LLaMA-1-30B	FP16	16-16-16	75.69	48.8	82.59	82.66	78.83	52.73	67.15	69.78	4.10
	BTC-LLM	1.11-16-16	74.74	47.6	79.94	81.83	78.07	50.94	66.79	68.56	4.59
	BTC-LLM	0.90-16-16	74.82	46.8	79.82	78.13	77.78	51.19	63.18	67.39	4.63
	BTC-LLM	0.80-16-16	73.16	45.0	76.07	71.71	73.99	45.39	66.06	64.48	5.29
	BTC-LLM	0.70-16-16	67.8	36.0	58.93	65.87	62.84	37.12	54.51	54.72	7.80
LLaMA-1-65B	FP16	16-16-16	77.11	47.2	84.15	84.86	79.84	55.55	69.68	71.2	3.53
	BTC-LLM	1.11-16-16	76.56	45.8	82.09	84.37	79.25	53.84	69.31	70.17	3.94
	BTC-LLM	0.90-16-16	76.01	46.6	81.79	82.94	79.17	54.1	71.84	70.35	4.03
	BTC-LLM	0.80-16-16	74.98	45.4	78.77	76.76	77.31	50.94	66.79	67.28	4.74
	BTC-LLM	0.70-16-16	70.01	40.4	65.66	71.04	66.75	40.02	60.65	59.22	6.61
LLaMA-2-7B	FP16	16-16-16	68.67	44.2	75.93	77.86	74.62	46.25	63.54	64.44	5.47
	BTC-LLM	1.11-16-16	67.09	41.4	71.36	74.71	71.17	41.47	65.7	61.84	6.06
	BTC-LLM	0.90-16-16	67.64	41.0	71.35	74.16	68.9	39.51	63.18	60.82	6.07
	BTC-LLM	0.80-16-16	74.98	45.4	78.77	76.76	77.31	50.94	66.79	67.28	6.60
	BTC-LLM	0.80-8-16	65.75	39.2	67.94	73.09	69.82	39.33	62.09	59.6	6.61
	BTC-LLM	0.80-8-8	65.98	27.00	50.06	71.77	71.13	36.38	62.09	59.8	6.52
	BTC-LLM	0.80-4-16	63.3	38.0	65.42	68.53	61.32	36.6	57.04	55.74	7.20
	BTC-LLM	0.80-4-4	58.17	22.40	44.90	67.58	63.05	30.55	57.04	53.44	7.94
	BTC-LLM	0.70-16-16	58.88	33.6	48.84	62.45	47.14	28.07	51.26	47.18	11.02
LLaMA-2-13B	FP16	16-16-16	72.22	45.4	79.39	80.58	77.48	49.32	64.98	67.05	4.88
	BTC-LLM	1.11-16-16	71.11	44.8	75.24	76.79	74.66	45.31	62.82	64.39	5.29
	BTC-LLM	0.90-16-16	71.9	45.0	75.4	76.21	74.79	46.33	62.45	64.58	5.32
	BTC-LLM	0.80-16-16	69.46	41.6	72.63	71.53	70.75	42.75	64.62	61.91	5.83
	BTC-LLM	0.70-16-16	62.83	32.8	52.07	63.18	54.12	30.89	51.99	49.7	8.76
LLaMA-3-8B	FP16	16-16-16	73.01	44.6	79.06	81.16	77.82	53.41	68.23	68.18	6.13
	BTC-LLM	1.11-16-16	72.77	42.8	73.53	76.94	73.02	47.01	59.93	63.71	7.70
	BTC-LLM	0.90-16-16	72.69	43.0	73.53	77.4	73.27	45.82	58.12	63.4	7.84
	BTC-LLM	0.80-16-16	67.96	41.6	66.76	75.32	65.32	41.13	57.04	59.3	9.49
	BTC-LLM	0.70-16-16	55.17	29.4	43.47	61.8	43.43	26.19	53.07	44.65	18.54

Table 7: Complete comparison of the perplexity score on WikiText2 and averaged accuracy on Zero-shot Common Sense Reasoning tasks for Qwen Model Family

Models	**Method**	**#Bits W-A-KV**	**Winogrande**	**OBQA**	**Hellaswag**	**Boolq**	**ARC-e**	**ARC-c**	**RTE**	**Average**	**WikiText2**
Qwen-2.5-3B	FP16	16-16-16	68.59	42.4	73.55	76.88	73.27	46.93	75.09	65.24	8.03
	BTC-LLM	1.11-16-16	66.69	39.4	66.32	75.14	70.75	42.75	78.34	62.77	9.70
	BTC-LLM	0.90-16-16	67.96	39.4	65.9	73.27	66.29	41.89	63.9	59.8	9.85
	BTC-LLM	0.80-16-16	64.88	37.0	61.54	64.92	67.21	39.68	55.96	55.88	11.26
	BTC-LLM	0.70-16-16	56.27	34.0	46.98	60.12	46.68	28.58	52.71	46.48	18.71
Qwen-2.5-14B	FP16	16-16-16	75.22	45.0	82.96	85.23	79.21	58.7	79.42	72.25	5.29
	BTC-LLM	1.11-16-16	76.01	46.0	79.37	86.3	82.83	57.76	81.23	72.79	6.49
	BTC-LLM	0.90-16-16	75.53	43.8	79.12	87.28	80.47	55.97	78.34	71.5	6.58
	BTC-LLM	0.80-16-16	74.43	41.2	75.42	86.02	76.64	50.0	70.4	67.73	7.42
	BTC-LLM	0.70-16-16	62.98	35.0	60.11	69.05	68.56	37.12	66.06	56.98	12.28
Qwen-3-0.6B	FP16	16-16-16	56.43	31.4	47.3	63.82	55.93	33.7	53.79	48.91	20.95
	BTC-LLM	0.8-16-16	50.2	26.6	32.61	61.16	33.42	24.66	53.07	40.25	120.08
Qwen-3-1.7B	FP16	16-16-16	61.17	36.6	60.46	77.68	69.95	42.75	70.04	59.81	16.71
	BTC-LLM	1.11-16-16	55.41	30.6	46.02	62.17	45.96	27.3	53.43	45.84	32.56
Qwen-3-8B	FP16	16-16-16	67.72	41.8	75.02	86.64	80.93	56.23	77.98	69.47	9.72
	BTC-LLM	1.11-16-16	65.67	39.6	67.02	81.38	76.68	50.17	77.62	65.45	11.60
	BTC-LLM	0.90-16-16	67.8	38.2	66.29	84.01	75.63	49.15	77.62	65.53	11.70
Qwen-3-14B	FP16	16-16-16	73.16	46.4	78.97	89.45	82.91	60.49	77.62	72.71	8.64
	BTC-LLM	1.11-16-16	67.64	40.0	66.92	85.72	71.72	48.38	78.34	65.53	12.05
	BTC-LLM	0.90-16-16	66.38	38.0	65.82	83.82	67.85	43.77	72.92	62.65	12.93