ANYBCQ: HARDWARE EFFICIENT FLEXIBLE BINARY-CODED QUANTIZATION FOR MULTI-PRECISION LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The deployment of large language models (LLMs) is increasingly constrained by memory and latency bottlenecks, motivating the need for quantization techniques that flexibly balance accuracy and efficiency. Recent work has introduced multi-precision models, which enable inference at multiple precisions within a single model depending on runtime constraints. To support such flexibility, quantized weights are often stored as bit-planes, where hardware efficiency improves when the compute operates directly at the bit-plane level and activates only the precision required by each request. In this work, we present AnyBCQ, a hardware-friendly multi-precision extension of Binary-Coded Quantization (BCQ) that supports direct bit-plane operations. By representing weights as binary bit-planes with corresponding scale factors, AnyBCQ enables bit-plane–level computation and maps naturally to accelerator-friendly, bit-parallel arithmetic. Our progressive precision expansion mechanism incrementally refines scaling factors while reusing previously assigned binary codes, yielding monotonic improvements in accuracy as additional bits are enabled. We further co-design a specialized kernel that exploits the BCQ structure to support dynamic per-request precision selection with negligible overhead. Experiments on recent LLMs demonstrate that AnyBCQ significantly narrows the accuracy drop in the low-bit regime (e.g. 2-bit), remains competitive at higher precision, and achieves throughput gains of up to $3.0\times$ over half precision and $1.2\times$ over state-of-the-art multi-precision methods. By aligning algorithmic flexibility with hardware efficiency, AnyBCQ provides a practical foundation for multi-precision LLM deployment across diverse service-level objectives.

1 INTRODUCTION

The rapid scaling of large language models (LLMs) has brought remarkable improvements in reasoning, generation, and downstream task performance (Kaplan et al., 2020; Hoffmann et al., 2022; Wei et al., 2022). However, this progress comes at the cost of soaring computational and memory demands, making efficient deployment a pressing challenge (Kim et al., 2023b). To address these constraints, a wide range of model compression techniques has been explored, including knowledge distillation (Sreenivas et al., 2024; Xu et al., 2024), pruning (Frantar & Alistarh, 2023; Sun et al., 2023; Lee et al., 2025), and quantization (Xiao et al., 2023; Ashkboos et al., 2024; Kim et al., 2025). Among these, post-training quantization (PTQ) has emerged as a particularly practical approach for LLMs, as it can substantially reduce memory footprint and accelerate inference without requiring expensive retraining (Dettmers et al., 2022; Frantar et al., 2022). Within PTQ, weight-only quantization has gained popularity since weights dominate memory usage and are relatively robust to outliers compared to activations (Lin et al., 2024). Recent state-of-the-art methods further demonstrate that 4-bit quantization can achieve accuracy comparable to full-precision models (Xiao et al., 2023).

While uniform quantization has been the most widely adopted strategy, recent works have introduced more expressive schemes, such as binary-coded quantization (BCQ) (You et al., 2024) and clustering-based non-uniform quantization (Kim et al., 2023a), to better capture the distribution of weight values and preserve accuracy after quantization. Despite their effectiveness, these methods are typically bound to a fixed precision configuration, which limits their ability to satisfy diverse service-level objectives (SLOs) in real-world deployments.

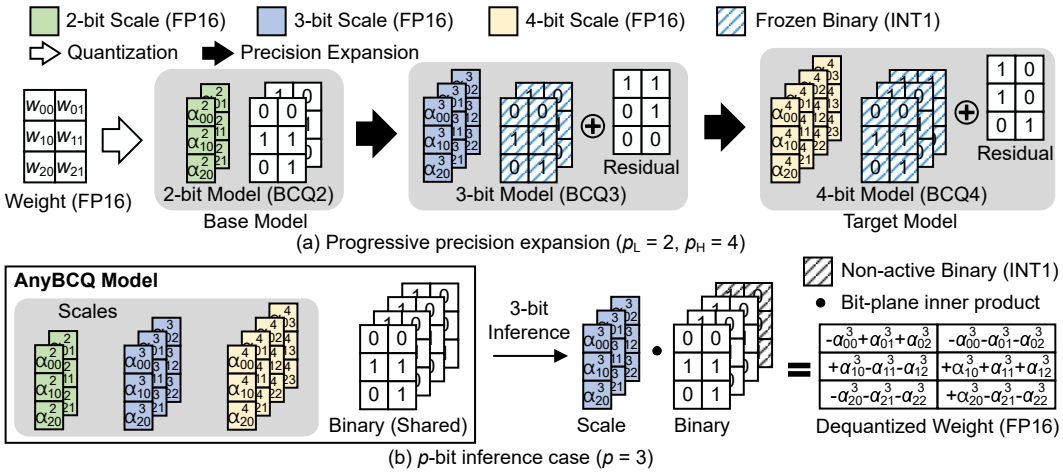

Figure 1: Overview of AnyBCQ: (a) weights are first quantized to a base precision and progressively expanded to higher precisions by reusing the existing binary codes while adding residual bit-planes; (b) p-bit inference reconstructs weights by combining the corresponding scaling factors with the first p binary bit-planes. In the binary representation, elements denoted as 0 are mapped as -1.

To address this limitation, the concept of multi-precision models has recently been proposed, allowing a single model to flexibly operate under multi-precisions and thereby adapt accuracy–latency trade-offs with dynamic system requirements (Yu et al., 2021; Park et al., 2024; Nair et al., 2025). This flexibility has, in turn, spurred research on mixed-precision inference, including methods that dynamically assign precision across decoding steps (Chen et al., 2024) or adaptively assign precisions on a per-layer basis (Kwon et al., 2025). However, these approaches remain limited in practice: the state-of-the-art multi-precision model (Park et al., 2024) relies on non-uniform quantization, which is not hardware-friendly and performs poorly at extremely low bitwidths (e.g., 2-bit).

Multi-precision inference often stores weights as bit-planes so that the system can fetch only the data required by the target precision. In this organization, the most efficient execution strategy is to compute directly on bit-planes and to activate only the planes needed for each request. Any-Precision LLM (Park et al., 2024) demonstrated strong accuracy across multiple precisions within a single unified model, yet its reliance on non-uniform quantization prevents true bit-plane computation. Even with carefully optimized kernels that accelerate centroid indexing through bit-transpose operations and table lookups, additional overheads and irregular memory access remain. As a result, dependence on centroid lookups continues to be a key bottleneck for hardware-efficient inference. Furthermore, extremely low-bit quantization (e.g., 2-bit) often induces severe accuracy degradation, while 4-bit quantization achieves accuracy close to full-precision models. As a result, the effective operating range of current methods is largely restricted to 3–4 bits, limiting the practical benefits of multi-precision quantization.

In this work, we propose AnyBCQ, a hardware-friendly quantization framework that extends BCQ to the multi-precision setting and supports direct bit-plane operations. Unlike non-uniform quantization, which relies on centroid lookups and bit transpositions to construct centroid indices and is therefore difficult to optimize on hardware, BCQ represents weights as binary bit-planes with associated scale factors. This structure is inherently accelerator-friendly, enabling efficient mapping to binary operations and simplifying kernel design (You et al., 2024; Park et al., 2022; 2025; Kim et al., 2023c).

Figure 1(a) presents the overall procedure of AnyBCQ. The model begins with a base-precision quantized representation and, through progressive precision expansion, evolves into a multi-precision model capable of supporting multiple precision levels. Specifically, the full-precision weights are first quantized to the base precision using a BCQ scheme. At each subsequent stage, the binary codes from earlier levels are frozen, while new scaling factors α are initialized and refined with an additional residual-derived bit-plane. This procedure is applied iteratively until the desired target precision is reached, yielding a model that can be used for inference at multiple precision levels. Figure 1(b) illustrates an example with inference precision $p = 3$. The resulting model contains the complete set of scaling factors required for each precision as well as the shared binary bit-planes.

During p-bit inference, the scaling factors corresponding to p-bit and the first p binary bit-planes are employed to perform weight dequantization. By unifying algorithmic flexibility with hardware efficiency, AnyBCQ provides a practical path toward multi-precision LLM deployment.

Our major contributions in this work include the following:

- We introduce AnyBCQ, a BCQ-based multi-precision framework that achieves strong low-bit accuracy and smooth, monotone improvements as additional bits are enabled.
- We co-design a hardware-friendly CUDA kernel that leverages a binary basis representation, supports direct bit-plane–level operations, and enables per-request precision selection with negligible overhead.
- We demonstrate state-of-the-art accuracy–latency trade-offs across LLM benchmarks, showing that AnyBCQ more effectively supports diverse SLOs with a single deployable model.

2 Background

2.1 Multi-Precision LLM

The multi-precision paradigm emerged from a practical need to serve heterogeneous SLOs in latency, throughput, and accuracy with a single deployable model. Early work in computer vision, mainly with CNNs, demonstrated that one network can operate at multiple precisions by training with quantization-aware training (QAT) across those settings (Yu et al., 2021). While effective, this approach is computationally demanding because the model must be trained from scratch under many quantization settings.

As Transformer-based LLMs scaled up, multi-configuration QAT became impractical due to high training cost and resource demands. Research therefore shifted toward post-training, often in a weight-only form. A prominent direction employs clustering-based non-uniform quantization with learned centroid tables (Kim et al., 2023a). To support multi-precision behavior, Any-Precision LLM (Park et al., 2024) introduces incremental upscaling, progressively splitting clusters and storing the centroid table so that a single model covers multiple precisions. Such approaches preserve accuracy well at medium and high precisions, often matching fixed-precision baselines, but performance drops sharply in extremely low-bit regimes (e.g., 2 bits). Consequently, practical deployment has remained confined to 3–4 bits, with 4-bit quantization in particular achieving accuracy close to full precision.

The systems implications of adopting non-uniform quantization are significant, as they directly affect how weights are stored, accessed, and processed during large-scale inference. In non-uniform schemes, each weight is stored as a centroid index, so inference requires table lookups and dequantization inside GEMMs. Any-Precision LLM mitigates this by storing weights as binary bit-planes and, at runtime, reading multiple bit-planes, transposing or packing them, and gathering the corresponding centroids before computation. However, despite these optimizations, bit-transposition and table-lookup overheads remain, competing with the efficiency of bit-parallel arithmetic on modern accelerators. These limitations motivate our AnyBCQ framework, which builds on BCQ to enable direct bit-plane operations and thereby supports dynamic-precision computation with low overhead. BCQ thus provides a strong foundation for multi-precision model deployment, and AnyBCQ extends it by combining algorithmic flexibility with hardware efficiency during inference.

2.2 Binary-coded Quantization

BCQ quantizes a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ by expressing it as a linear combination of q binary bases and real-valued scales: $\hat{\mathbf{W}} = \sum_{i=1}^{q} \alpha_i \mathbf{B}_i$, where each $\mathbf{B}_i \in \{-1, 1\}^{m\times n}$ and $\alpha_i \in \mathbb{R}$. Here, q denotes the quantization bitwidth. The parameters are obtained by minimizing the Frobenius reconstruction error $e = \|\mathbf{W} - \hat{\mathbf{W}}\|_F^2$. When $q = 1$, the solution reduces to standard binary quantization with $\mathbf{B}_1^* = \mathrm{sign}(\mathbf{W})$ and $\alpha_1^* = \langle \mathbf{W}, \mathbf{B}^* \rangle / \|\mathbf{B}^*\|_F^2$. For multi-bit quantization ($q > 1$), we adopt a greedy initialization followed by alternating refinement (Xu et al., 2018). Specifically, the residual after i-bit quantization is defined as $\mathbf{R}_i = \mathbf{W} - \sum_{j=1}^{i} \alpha_j \mathbf{B}_j$. The next binary and scale are initialized as $\mathbf{B}_{i+1} = \mathrm{sign}(\mathbf{R}_i)$ and $\alpha_{i+1} = \langle \mathbf{R}_i, \mathbf{B}_{i+1} \rangle / \|\mathbf{B}_{i+1}\|_F^2$. These initial values are then refined by alternating updates of scales and binary codes. Concatenating binary bases as

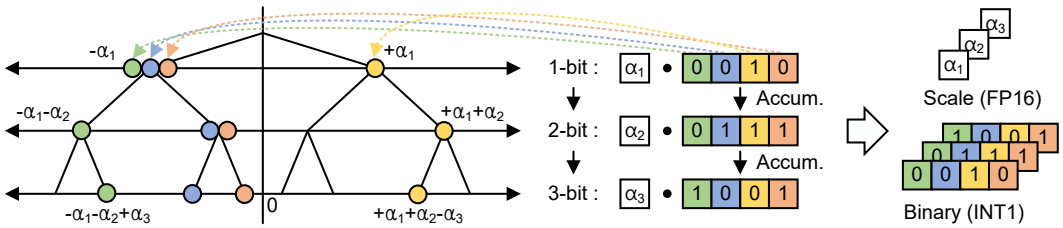

Figure 2: Illustration of the binary-coding quantization scheme. Weights are quantized hierarchically, with each bit level determining its corresponding binary values. At each level, the scaling factor and binary assignment are computed and accumulated with the value obtained from the previous bit level to approximate the original weight. The resulting representation comprises bit-planes for each precision level, each paired with its associated scaling factors.

$\mathbf{B} = [\mathbf{B}_1 \cdots \mathbf{B}_q]$, the scale vector is updated by ordinary least squares: $\boldsymbol{\alpha} = (\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{W}$, after which each $\mathbf{B}_i$ is recalibrated using the binary search given the refined $\boldsymbol{\alpha}$. This greedy-plus-alternating procedure yields an efficient q-bit BCQ approximation in binary bit-planes that we later exploit for multi-precision execution without centroid lookups or bit-transpose passes.

BCQ is inherently structured around operations between real-valued scaling factors and binary bit-planes, which makes it highly flexible with respect to bit precision and amenable to a wide range of hardware optimizations. For example, iFPU (Kim et al., 2023c) demonstrates that arithmetic within each binary bit-plane can be simplified by exploiting exponent pre-alignment: floating-point additions and subtractions are reduced to integer-level operations on mantissas, thereby lowering the complexity of floating-point computations to that of integer arithmetic. Similarly, LUT-GEMM (Park et al., 2022) and FIGLUT (Park et al., 2025) both exploit the fact that the output of each binary bit-plane is ultimately a simple combination $(+/-)$ of floating-point input activations. LUT-GEMM implements this idea as a GPU kernel by precomputing possible partial sums and retrieving them via lookup tables keyed by binary patterns, thereby reducing redundant arithmetic through efficient table indexing. FIGLUT applies the same principle at the architectural level in a custom accelerator design, where partial sums are stored and reused in hardware to further boost efficiency. Building multi-precision models on top of BCQ further enhances deployability, as they can seamlessly run on accelerators that already support BCQ-based formats, ensuring compatibility while retaining efficiency across diverse bitwidth configurations.

3 Methodology

AnyBCQ is a multi-precision LLM framework built on Binary-Coded Quantization. It encodes each weight as a sum of binary components with associated scale factors, which enables direct bit-plane operations and permits dynamic precision selection at inference time with negligible overhead. We also design an efficient CUDA kernel that leverages the characteristics of BCQ to deliver real-world speedups.

3.1 Motivation

Efforts have been made to design multi-precision models for diverse SLOs, but existing methods are not hardware-friendly since they cannot operate directly at the binary bit-plane level. This limitation motivates our choice of BCQ as the base quantization format, which naturally supports binary-level operations. Figure 2 illustrates the hierarchical process of BCQ. Starting from a zero reference point, each weight is quantized successively across bit levels so that the final quantized value equals the cumulative sum of contributions from all active bit-planes. For example, the leftmost weight in Figure 2 (green) is first encoded as 0 at the 1-bit level, representing $-\alpha_1$. At the 2-bit level, it is again assigned 0, giving $-\alpha_1 - \alpha_2$. Finally, at the 3-bit level, it is assigned 1, yielding the final value $-\alpha_1 - \alpha_2 + \alpha_3$. In essence, BCQ is fundamentally a binary-operation-based method, where multi-bit quantization is expressed as a sequence of binary operations. Thus, p-bit inference naturally corresponds to p binary bit-plane computations, a property that makes BCQ particularly well-suited for multi-precision LLMs in which bit precision is determined dynamically at runtime.

Algorithm 1 AnyBCQ initialization and progressive precision expansion

Input: Full-precision weight $\mathbf{W} \in \mathbb{R}^{m \times n}$, base precision p_L, target precision p_H, alternating cycles T

Output: $\{\alpha_i^{p_L}\}_{i=1}^{p_L}, \ldots, \{\alpha_i^{p_H}\}_{i=1}^{p_H}, \{\mathbf{B}_i\}_{i=1}^{p_H}$ with $\mathbf{B}_i \in \{-1, 1\}^{m \times n}$

1: **Function** ANYBCQ($\mathbf{W}, p_L, p_H, T$)
2: **for** $p \leftarrow p_L$ **to** p_H **do**
3: **if** $p = p_L$ **then** ▷ Initialization at base precision
4: $\{\alpha_i^p, \mathbf{B}_i\}_{i=1}^{p} \leftarrow$ GREEDY($\mathbf{W}$)
5: **for** $t \leftarrow 1$ **to** T **do**
6: $\mathbf{B} \leftarrow [\mathbf{B}_1, \ldots, \mathbf{B}_p]$
7: $\{\alpha_i^p\}_{i=1}^{p} \leftarrow \mathrm{LS}(\mathbf{B}, \mathbf{W})$
8: $\{\mathbf{B}_i\}_{i=1}^{p} \leftarrow \mathrm{BS}(\alpha_1^p, \ldots, \alpha_p^p, \mathbf{W})$
9: **else** ▷ Initialization at progressive expansion step
10: $\{\alpha_i^p\}_{i=1}^{p} \leftarrow [\alpha_1^{p-1}, \ldots, \alpha_{p-1}^{p-1}, 0]$
11: $\mathbf{B}_p \leftarrow 0$
12: **for** $t \leftarrow 1$ **to** T **do**
13: $\mathbf{R} \leftarrow \mathbf{W} - \text{DEQUANT}(\{\alpha_i^p, \mathbf{B}_i\}_{i=1}^{p})$
14: $\mathbf{B}_p \leftarrow \text{SIGN}(\mathbf{R})$ ▷ Initialize new bit-plane
15: $\mathbf{B} \leftarrow [\mathbf{B}_1, \ldots, \mathbf{B}_p]$
16: $\{\alpha_i^p\}_{i=1}^{p} \leftarrow \mathrm{LS}(\mathbf{B}, \mathbf{W})$

3.2 ANYBCQ FRAMEWORK

In line with the principles of multi-precision LLMs, AnyBCQ introduces a progressive precision expansion mechanism that enables seamless transitions across different bit-width representations. Let the candidate precision set be $\mathcal{P} = \{p \mid p_L \leq p \leq p_H\}$, where p_L and p_H denote the lowest base precision and the highest target precision, respectively (indices L and H). As illustrated in Figure 1(a) for $p_L = 2$ and $p_H = 4$, the procedure starts from a model quantized at p_L bits and then increases the precision one bit at a time until reaching the target model at p_H bits.

To support multiple precision levels while ensuring efficient memory utilization within a single model, AnyBCQ employs shared binary representations and scaling factors tailored to each target precision. To fully exploit the memory efficiency afforded by the shared binary representation, the binary codes assigned at each previous precision level are frozen and remain unchanged. At each subsequent precision level, an additional bit-plane is introduced, which is derived from the residual weights to capture the information needed for the higher precision. The scaling factors required for the current precision are then initialized accordingly, ensuring accurate representation as the model scales to higher bit-widths. Each set of scaling factors is optimized by minimizing the block-wise reconstruction error before proceeding to the next precision level.

The optimization procedure for each precision level in AnyBCQ follows a two-stage framework, consisting of an initialization phase followed by a subsequent error-minimization phase. Algorithm 1 outlines the initialization procedure of the BCQ framework before the error-minimization phase. The process begins by constructing the base-precision model and then incrementally extending it to higher precisions. When $p = p_L$, the model is initialized as described in Algorithm 1 (lines 3–8). In this stage, the scaling factors α and the binary codes B are first determined in a greedy way from the original weights. Subsequently, during the T optimization cycles, the scaling factors are refined by solving a least-squares (LS) problem between the binary representation and the original weights, after which the binary codes are reassigned via a binary search (BS) between the optimized scaling factors and the original weights. In contrast, for higher target precisions when the current precision exceeds the base precision, the model initializes both the newly introduced scaling factor α and its associated bit-plane to zero as described in Algorithm 1 (lines 10–11). During subsequent T optimization cycles, the binary codes of this additional bit-plane are reassigned by taking the sign of the residual weights with the optimized scaling factors, while all scaling factors are updated via least-squares refinement as in the base precision initialization step. Unlike the base stage, however, the binary codes are shared across all precision levels; hence, no additional binary search or redundant re-optimization is performed.

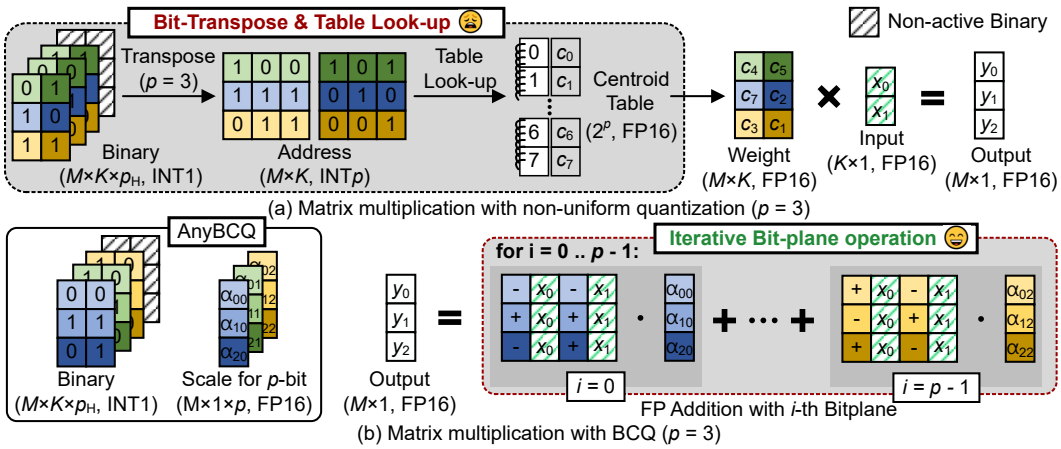

Figure 3: Matrix multiplication with (a) clustering-based quantization, which requires bit-plane transposition and centroid lookups, and (b) the proposed AnyBCQ kernel, which directly operates on binary bit-planes with scaling factors for hardware-efficient, dynamic-precision inference.

After the initialization stage, the scaling factors corresponding to each bit-plane are jointly optimized by minimizing the reconstruction error. Because a distinct set of scaling factors is maintained for each target precision, the reconstruction error minimization (Li et al., 2021) is applied only to the scaling factors associated with the current bit precision, while the corresponding bit-plane remains fixed. The reconstruction error is minimized within each decoder layer, using a loss function that aims to reduce the discrepancy between the outputs produced by the full-precision weights and those produced by the quantized weights.

To preserve accuracy on large LLM matrices, we adopt group-wise quantization with per-group scale factors (Yao et al., 2022; Park et al., 2022). Table 1 presents the memory footprint at each target precision with group size $g = 128$. *Multi-model* denotes the baseline that stores a separate model optimized for each bit precision. In contrast, AnyBCQ supports multiple precisions within a single model by sharing the binary representations across precisions while keeping precision-specific scale factors. Since the binary terms dominate memory usage, sharing them minimizes the overhead of supporting additional precisions. As a result, AnyBCQ reduces the total memory footprint by 49% compared with the multi-model baseline on Llama-3.1-8B.

Table 1: Memory footprint (GB) of quantized Llama-3.1-8B.

Bit	Scale	Binary	Total
BCQ2	0.24	1.95	2.19
BCQ3	0.36	2.92	3.28
BCQ4	0.49	3.89	4.38
Multi-model	1.09	8.76	9.85
Proposed	1.09	3.89	4.99

3.3 Kernel Design

To enable efficient inference of multi-precision LLMs, specialized kernel designs are required. Unlike conventional kernels, a multi-precision kernel is expected to support dynamically varying bit precision at runtime. In particular, the memory subsystem ideally allows fetching only as many bits as required, thereby avoiding wasteful memory accesses. For example, in a model with maximum 4-bit precision ($p_H = 4$), one could perform 3-bit inference ($p = 3$) by loading 4-bit weights and discarding the least significant bit. Although this is functionally correct, it eliminates the intended memory bandwidth savings of 3-bit inference, which is especially detrimental in the memory-bound regime typical of LLM inference.

To address this inefficiency, prior multi-precision kernels often store weights at the granularity of individual bit planes. Specifically, a quantized weight matrix of shape $M \times K$ with p_H bits is decomposed into a binary tensor of shape $M \times K \times p_H$. At inference time, only the first p bit planes are loaded to obtain the indices for centroid table lookup, thus preserving the bandwidth advantage of lower-bit inference. However, this representation introduces another overhead, namely bit transposition. The first p bit planes need to be rearranged to form the index values. Because

bitwise operations are not directly applicable in clustering-based non-uniform quantization, the fetched $M \times K \times p$ binary tensor is then transposed back into an $M \times K$ p-bit matrix to serve as indices for centroid table lookup (Figure 3(a)).

In contrast, BCQ-based multi-precision models offer a key advantage over prior approaches: they can directly operate on binary bit-planes at the required precision without incurring the overhead of bit transposition or centroid table lookup (Figure 3(b)). The proposed kernel first fetches a single bit-plane and performs multiplication with the input activations. Since each binary bit-plane contains values in {-1, +1}, the operation reduces to simple addition and subtraction of activation elements. To further improve efficiency, the kernel adopts a lookup-table–based GEMM scheme (Jeon et al., 2020; Park et al., 2022; 2025) in which frequently repeated computation results are cached in table form. Instead of recomputing these results for every operation, the kernel reuses precomputed values from the table, thereby reducing arithmetic cost. The output of each bit-plane computation is combined with its corresponding scaling factor α_i and accumulated as a partial sum. Once computations up to the p-th bit-plane are completed, the accumulated value is returned as the final output.

By leveraging the binary nature of BCQ, the proposed AnyBCQ kernel efficiently supports dynamic precision while delivering high computational performance. This advantage is particularly pronounced in the memory-bound regime of LLM inference. Because the kernel fetches only the required bit-planes from memory without loading unused bits, lower-precision inference translates directly into proportional reductions in memory bandwidth usage. Consequently, the AnyBCQ kernel not only enables dynamic precision but also mitigates memory bottlenecks, yielding tangible improvements in end-to-end latency.

4 Experimental Results

4.1 Experimental Settings

Models and Evaluations. We benchmarked our method on the LLaMA-3.1-8B (Grattafiori et al., 2024) model. To assess general knowledge, we evaluated the model on 5-shot MMLU (Hendrycks et al., 2020) and zero-shot common sense reasoning tasks: ARC-Challenge (Clark et al., 2018), ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), Phrase-Indexed Question Answering (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021), using the `lm-eval-harness` framework (v0.4.5) with HuggingFace implementations. Our baselines include state-of-the-art multi-precision methods together with other weight-only quantization techniques.

Implementation Details. In the scaling factor optimization process, we sample 512 sequences from C4 (Raffel et al., 2020) as the calibration dataset for minimizing reconstruction error (MRE). Unless otherwise noted, models are quantized and optimized for 10 epochs under the asymmetric BCQ (Park et al., 2022) scheme with group-wise quantization, using a fixed group size of $g = 128$. The learning rate is set to 1×10^{-4}, and the number of refinement cycles for both the base-precision stage and the incremental-precision initialization is $T = 20$. Kernel latency is measured on NVIDIA A100 GPUs with 80 GB HBM, running CUDA 12.6.

4.2 Accuracy Evaluation

Table 2 compares downstream accuracy on the LLaMA-3.1-8B model across different methods. Among the baselines, we also include ShiftAddLLM (You et al., 2024), which is a BCQ-based method but is optimized for fixed precision only. In addition, to isolate the effect of the progressive bit-width expansion mechanism, we report results for a variant of AnyBCQ optimized at a fixed bit-width (denoted as *Proposed (fixed-precision)*).

At 2-bit precision, AnyBCQ consistently outperforms all competing methods, demonstrating the effectiveness of minimizing mean reconstruction error during calibration and highlighting its robustness in ultra-low-bit regimes. Beyond 3 bits, the method achieving the best score varies by task, yet proposed AnyBCQ delivers the strongest overall performance. Unlike the 2-bit case, performance differences between *Proposed (multi-precision)* and *Proposed (fixed-precision)* become apparent; this gap arises from the shared-binary constraint in progressive precision expansion, which narrows the optimization space as the bit-width increases. At 4 bits, the gap to the FP16 baseline is largely diminished and

Any-Precision LLM attains the top accuracy on several tasks. Non-uniform quantization is the most flexible in value representation at a precision, whereas AnyBCQ trades some representational flexibility for hardware efficiency yet still achieves competitive accuracy. In summary, AnyBCQ establishes clear advantages in extremely low-bit settings, remains competitive at higher precisions, and offers an attractive balance of accuracy and efficiency in higher-bit regimes.

Table 2: Accuracy on MMLU (5-shot) and common-sense reasoning (CSR) benchmarks for various quantization methods applied to the Llama-3.1-8B model. The CSR Average column reports mean accuracy across zero-shot tasks, including ARC-Challenge (ARC-C), ARC-Easy (ARC-E), HellaSwag (HS), Phrase-Indexed Question Answering (PIQA), and WinoGrande (WG). "Fixed-precision" denotes a model optimized to operate at a single bit-width.

Method	Bit	MMLU	ARC-C	ARC-E	HS	PIQA	WG	CSR Avg.
FP16	16	65.02	53.41	77.69	79.15	80.74	72.61	72.72
AWQ	2	24.12	25.34	25.59	26.63	51.52	48.93	35.60
Any-Precision LLM	2	24.66	25.00	35.61	29.28	56.26	52.09	39.65
ShiftAddLLM	2	24.83	25.85	41.96	44.72	58.54	57.85	45.78
Proposed (fixed-precision)	2	**35.96**	36.60	**63.22**	62.56	73.45	**58.64**	**58.89**
Proposed (multi-precision)	2	35.32	**37.03**	62.50	**62.61**	**73.88**	57.54	58.71
AWQ	3	47.28	44.51	71.97	**75.53**	78.85	65.69	67.31
Any-Precision LLM	3	55.53	45.22	71.96	71.31	**79.43**	64.32	66.45
ShiftAddLLM	3	56.53	47.61	74.54	73.65	78.02	**72.85**	69.33
Proposed (fixed-precision)	3	**59.41**	**48.46**	**76.73**	75.29	79.22	71.98	**70.34**
Proposed (multi-precision)	3	58.28	46.76	76.05	74.39	79.38	69.93	69.30
AWQ	4	60.49	51.82	75.42	77.98	79.38	72.35	71.39
Any-Precision LLM	4	**64.04**	**53.32**	79.97	**78.11**	80.25	71.19	72.57
ShiftAddLLM	4	63.50	51.54	79.50	77.39	80.36	**74.03**	72.56
Proposed (fixed-precision)	4	63.90	52.65	**80.09**	77.74	**81.07**	72.69	**72.85**
Proposed (multi-precision)	4	63.15	51.96	78.79	77.28	80.58	73.24	72.37

4.3 KERNEL EVALUATION

Table 3 compares the latency of matrix–vector multiplication (GEMV) across three settings: cuBLAS with floating-point weights, the state-of-the-art multi-precision kernel (Park et al., 2024), and our proposed AnyBCQ at different precisions. To reflect a range of LLM model sizes, we instantiate layer shapes following the linear-layer configurations of Llama-3.1-8B, Phi-4-14B, and Llama-3.1-70B.

Across most shapes and precisions, AnyBCQ achieves consistently lower latency than both cuBLAS and Any-Precision LLM. We observe two general trends: (i) the performance gap widens as the model (matrix) size increases, and (ii) within a model, the gain grows with the input dimension K. These trends arise because AnyBCQ executes directly over binary bit-planes, which removes dequantization overheads that are intrinsic to non-uniform schemes, in particular bit transposition and centroid-table lookup. In addition, the BCQ representation allows AnyBCQ to exploit binary-matrix optimizations such as LUT-based computation (Park et al., 2022), which suppress redundant operations and further reduce runtime.

Table 3: Latency (μs) of GEMV kernels for representative linear-layer shapes from Llama-3-8B, Phi-4-14B, and Llama-3-70B. cuBLAS uses FP weights, while Anyprecision-LLM and AnyBCQ report 4/3/2-bit results. Lower is better.

N	K	cuBLAS	Any-Precision LLM			Proposed		
		16-bit	2-bit	3-bit	4-bit	2-bit	3-bit	4-bit
4096	4096	296 (×1.00)	230 (×1.29)	247 (×1.20)	266 (×1.11)	**223** (×1.33)	**246** (×1.20)	**263** (×1.12)
14336	4096	852 (×1.00)	353 (×2.41)	404 (×2.11)	476 (×1.79)	**319** (×2.67)	**384** (×2.22)	**456** (×1.87)
4096	14336	877 (×1.00)	356 (×2.47)	412 (×2.13)	502 (×1.75)	**315** (×2.78)	**373** (×2.35)	**462** (×1.90)
5120	5120	433 (×1.00)	**248** (×1.74)	272 (×1.59)	304 (×1.42)	253 (×1.71)	**270** (×1.60)	**298** (×1.45)
17920	5120	1230 (×1.00)	432 (×2.85)	546 (×2.25)	631 (×1.95)	**409** (×3.01)	**512** (×2.40)	**597** (×2.06)
5120	17920	1272 (×1.00)	445 (×2.86)	581 (×2.19)	687 (×1.85)	**406** (×3.14)	**521** (×2.44)	**593** (×2.14)
8192	8192	946 (×1.00)	378 (×2.50)	439 (×2.15)	544 (×1.74)	**336** (×2.82)	**428** (×2.21)	**499** (×1.90)
28672	8192	3040 (×1.00)	830 (×3.66)	1058 (×2.87)	1292 (×2.35)	**747** (×4.07)	**938** (×3.24)	**1133** (×2.68)
8192	28672	2968 (×1.00)	971 (×3.06)	1265 (×2.35)	1348 (×2.20)	**742** (×4.00)	**939** (×3.16)	**1142** (×2.60)

4.4 End-to-end Evaluation

We evaluate the accuracy–throughput trade-off of Any-Precision LLM (AP) and proposed AnyBCQ (AB) on Llama-3.1-8B, Gemma-2-9B, and Phi-4-14B. Table 4 reports Wiki perplexity (lower is better), MMLU accuracy, and decoding throughput in tokens per second.

At 2-bit precision, AnyBCQ consistently preserves accuracy substantially better than AP across all models, indicating that our MRE-based calibration is particularly effective under aggressive compression. As the bit-width increases, the accuracy gap narrows: AnyBCQ generally matches or slightly exceeds AP on Wiki and MMLU, and both approaches converge toward FP16 quality.

Throughput consistently favors AnyBCQ. Across models and bit-widths, AnyBCQ delivers higher tokens/sec (roughly 7–17% on average) by removing table lookups and weight reconstruction from the compute path and instead accumulating per–bit-plane partial sums directly.

Overall, AnyBCQ offers a stronger accuracy–throughput frontier. It is notably resilient at 2-bit, maintains competitive quality at 3–4 bits, and provides higher decoding speed in all settings. Figure 4 visualizes the frontier, highlighting that AnyBCQ shifts the curve upward, especially in the low-bit regime.

Table 4: End-to-end Evaluation of Any-Precision LLM (AP) and AnyBCQ (AB)

Model	bit	Wiki			MMLU			Token/sec		
		FP16	AP	AB	FP16	AP	AB	FP16	AP	AB
Llama-3.1-8B	2	6.24	1680.77	**19.01**	0.6535	0.2466	**0.3532**	105	228	**245**
	3	6.24	8.60	**8.08**	0.6535	0.5553	**0.5828**	105	196	**212**
	4	6.24	**6.70**	6.84	0.6535	**0.6404**	0.6315	105	169	**186**
Gemma-2-9b	2	6.84	19.62	**12.72**	0.7074	0.3309	**0.4336**	83	163	**185**
	3	6.84	**7.94**	7.95	0.7074	0.6530	**0.6538**	83	144	**164**
	4	6.84	**7.06**	7.23	0.7074	**0.6923**	0.6881	83	125	**146**
Phi-4-14b	2	6.46	13.37	**9.97**	0.8039	0.4952	**0.6638**	56	147	**171**
	3	6.46	**7.14**	7.44	0.8039	**0.7732**	0.7708	56	125	**144**
	4	6.46	**6.61**	7.16	0.8039	**0.7975**	0.7904	56	105	**123**

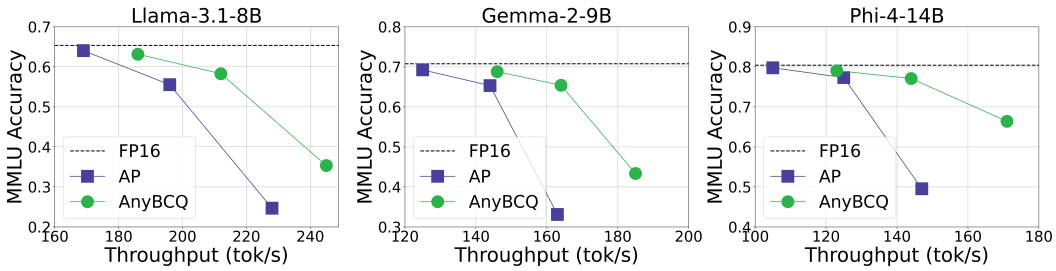

Figure 4: Accuracy–throughput trade-offs for 2-, 3-, and 4-bit configurations across models. The rightmost point denotes the 2-bit setting. For a given accuracy, AnyBCQ attains higher throughput (tokens/sec) than Any-Precision LLM (AP), with the largest gain at 2 bits.

5 Summary and Limitations

We present *AnyBCQ*, a BCQ-based framework for multi-precision LLMs co-designed with an efficient execution kernel. By sharing binary bit-planes across precisions while learning per-precision scales, AnyBCQ minimizes the memory overhead of multi-precision deployment, and its kernel executes directly on bit planes to improve hardware efficiency. Empirically, AnyBCQ substantially improves accuracy in the low-bit regime (for example, 2-bit), remains competitive at 3–4 bits, and offers a favorable accuracy–throughput trade-off. A limitation is that the inherent representational capacity of BCQ, together with the shared-binary constraint, can reduce peak accuracy at higher bit widths relative to non-uniform schemes. Nevertheless, recent advances in weight-only quantization yield 4-bit performance that is close to the full-precision baseline, so the absolute gap at higher precisions is modest in practice.

6 Reproducibility Statement

In the Supplementary Materials, we provide the necessary resources to reproduce all results reported in the *Experimental Results* section. Specifically, this entails:

- An implementation of our quantization method and evaluation pipelines for the supported tasks.
- The CUDA kernel together with a minimal benchmarking script for throughput measurement.
- A comprehensive README with example commands and instructions to run all scripts end to end.

References

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

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

Hao Mark Chen, Fuwen Tan, Alexandros Kouris, Royson Lee, Hongxiang Fan, and Stylianos I Venieris. Progressive mixed-precision decoding for efficient llm inference. *arXiv preprint arXiv:2410.13461*, 2024.

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

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

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

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Yongkweon Jeon, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Jeongin Yun, and Dongsoo Lee. Biqgemm: matrix multiplication with lookup table for binary-coding-based quantized dnns. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Jinuk Kim, Marwa El Halabi, Wonpyo Park, Clemens JS Schaefer, Deokjae Lee, Yeonhong Park, Jae W Lee, and Hyun Oh Song. Guidedquant: Large language model quantization via exploiting end loss guidance. *arXiv preprint arXiv:2505.07004*, 2025.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023a.

Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023b.

Yulhwa Kim, Jaeyong Jang, Jehun Lee, Jihoon Park, Jeonghoon Kim, Byeongwook Kim, Se Jung Kwon, Dongsoo Lee, et al. Winning both the accuracy of floating point activation and the simplicity of integer arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023c.

Sangwoo Kwon, Seong Hoon Seo, Jae W Lee, and Yeonhong Park. Dp-llm: Runtime model adaptation with dynamic layer-wise precision assignment. *arXiv preprint arXiv:2508.06041*, 2025.

Dongyeop Lee, Kwanhee Lee, Jinseok Chung, and Namhoon Lee. Safe: Finding sparse and flat minima to improve pruning. *arXiv preprint arXiv:2506.06866*, 2025.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

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

Pranav Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. *arXiv preprint arXiv:2502.06786*, 2025.

Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

Gunho Park, Hyeokjun Kwon, Jiwoo Kim, Jeongin Bae, Baeseong Park, Dongsoo Lee, and Youngjoo Lee. Figlut: An energy-efficient accelerator design for fp-int gemm using look-up tables. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1098–1111. IEEE, 2025.

Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W Lee. Any-precision llm: Low-cost deployment of multiple, different-sized llms. *arXiv preprint arXiv:2402.10517*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

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

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, et al. Llm pruning and distillation in practice: The minitron approach. *arXiv preprint arXiv:2408.11796*, 2024.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

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

Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. *arXiv preprint arXiv:1802.00150*, 2018.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35:27168–27183, 2022.

Haoran You, Yipin Guo, Yichao Fu, Wei Zhou, Huihong Shi, Xiaofan Zhang, Souvik Kundu, Amir Yazdanbakhsh, and Yingyan Celine Lin. Shiftaddllm: Accelerating pretrained llms via post-training multiplication-less reparameterization. *Advances in Neural Information Processing Systems*, 37: 24822–24848, 2024.

Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S Huang, and Gang Hua. Any-precision deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10763–10771, 2021.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.