MoBiQuant: Mixture-of-Bits Quantization for Token-Adaptive Any-Precision LLM

Dongwei Wang[1,4,*], Jinhee Kim[2,*], Seokho Han[3,*],
Denis Gudovskiy[4], Yohei Nakata[4], Tomoyuki Okuno[4], KhayTze Peong[4],
Kang Eun Jeon[5], Jong Hwan Ko[3], Yiran Chen[2], Huanrui Yang[1,†]
[1] University of Arizona [2]Duke University [3]Sungkyunkwan University
[4] Panasonic AI Lab [5]Korea Advanced Institute of Science and Technology
{dongweiw,huanruiyang}@arizona.edu
*Equal contribution. †Corresponding author.

Abstract

Dynamic runtime latency and memory constraints necessitate flexible large language model (LLM) deployment, where an LLM can be inferred with various quantization precisions based on available computational resources. Recent work on such any-precision quantization either relies on hardware-inefficient vector quantization or induces additional scaling factors when switching between bit-widths. Meanwhile, existing post-training quantization (PTQ) methods calibrated for a fixed low precision show poor generalizability under runtime precision change. In this work, we attribute the source of poor generalization across bit-widths to a precision-dependent *outlier migration* phenomenon where the distribution of PTQ-sensitive tokens changes across precisions. Motivated by this observation, we propose `MoBiQuant`, a novel any-precision Mixture-of-Bits quantization framework that adjusts weight precision for flexible LLM inference based on token sensitivity. Specifically, we propose a many-in-one recursive residual quantization that can iteratively reconstruct higher-precision weights at runtime and mitigates *outlier migration* with a token-aware router to dynamically select the optimal inference precision of each token. Extensive experiments show that `MoBiQuant` matches or surpasses frontier single-precision PTQ while exhibiting strong elasticity, achieving significant memory savings and throughput gains of up to $1.34\times$ over state-of-the-art any-precision methods.

1 Introduction

Recent deployments of large language models (LLMs) [1–5] typically use low-precision quantization to meet latency and memory constraints. Although ongoing work on post-training quantization (PTQ) [6, 7] constantly improves LLM performance at predetermined fixed precisions, the tradeoff between precision and performance remains critical.

The impact of such a tradeoff is more significant for edge devices where available computational resources can vary dynamically at runtime. Under sufficient resources, the device can run the model at higher precision for better performance, while under resource contention from other applications, it may need to reduce precision to satisfy latency and memory constraints. Storing LLM weights for each precision is impractical, and, hence, recent research explores any-precision quantization where a single model can support multiple precisions [8–10]. However, existing approaches remain limited in practice, as they either rely on hardware-unfriendly vector quantization (VQ) [8] or introduce additional calibration parameters that lead to training and inference overhead [9]. Moreover, they often lack flexibility for runtime adaptation, requiring offline repacking and kernel redeployment when switching precisions [10].

Preprint.

In this work, we overcome these limitations and propose a more advanced any-precision framework with three key properties: (1) it supports hardware-friendly scalar quantization, (2) it introduces minimal overhead compared to a single set of PTQ calibration parameters, and (3) it enables efficient runtime precision switching without offline repacking or kernel relaunching. To achieve this, we start by investigating the root cause of why existing single-precision scalar PTQ methods fail to generalize across bit-widths with fixed calibration parameters.

We identify a critical phenomenon of precision-dependent *"outlier migration"* in LLMs. Although it is well-established that outliers in activations govern quantization errors [11], we find that the specific subset of tokens that are responsible for high quantization errors are not static for each precision. Consequently, static PTQ methods that calibrate quantization parameters (e.g., scaling factors or clipping ranges) for a single precision, often fail to generalize. They inevitably overfit to an improper set of tokens when the bit-width changes, leading to suboptimal performance in any-precision inference setting.

Motivated by this, we propose `MoBiQuant`, an any-precision quantization framework that mitigates "*outlier migration*" with a single set of calibration parameters by incorporating token-adaptive precision adjustment during calibration. `MoBiQuant` learns to dynamically assign proper inference precision to tokens on the fly based on their sensitivity while maintaining overall inference cost within an average budget. Our approach consists of two core components. The first one is `MoBiSlice`, a "many-in-one" recursive residual quantization that decomposes weights into a low-precision base and successive residual bit slices. This allows a single model to support multiple precisions (e.g., 2, 4, 6-bit) via activating fixed-precision bit slices (e.g., 2-bit slice) at runtime, thereby supporting any-precision inference with a single type of low-precision bit slice GEMM kernel without the need for repacking and redeployment. The second component is `MoBiRoute`, a lightweight learnable router that implements token-aware inference precision selection. Acting as a gate, the `MoBiRoute` dynamically activates the optimal number of `MoBiSlice` residual components for each token based on learned token sensitivity behavior and current resource conditions, thereby preventing the calibration parameters from overfitting to the outlier distribution of a single precision. Runtime precision switching is achieved by adjusting the routing threshold, without introducing extra scaling factors and training overhead.

In summary, our paper makes the following contributions:

- The `MoBiSlice` recursive quantization explicitly reconstructs higher precisions using low-precision bit slices. It enables a single model to flexibly realize multiple precision levels via hierarchical reconstruction, without requiring offline repacking and redeployment.
- The `MoBiRoute` router mitigates the bit-dependent *outlier migration* phenomenon by dynamically selecting the number of bit slices for each token. It also enables efficient runtime precision switching by a threshold adjustment, eliminating the need to store additional scaling factors and extra training overhead.
- Extensive experiments demonstrate that `MoBiQuant` matches or surpasses the performance of frontier single-precision PTQ while exhibiting strong elasticity. Compared to deploying multiple models, it significantly reduces memory footprint by $3.5\times$, and our tailored kernel design delivers $1.34\times$ higher throughput over state-of-the-art any-precision methods [8, 9].

2 Related Work

Single-precision PTQ. Most existing methods focus on statically-defined quantization schemes, where LLM performance is particularly optimized for the selected bit-width configuration. Representative works include: GPTQ [12] with second-order information to minimize quantization error, OmniQuant [7] with learnable weight clipping to achieve superior results, SmoothQuant [13] and SpinQuant [14] with special linear transformations to mitigate peaking outliers, and AWQ [15] for selectively quantizing salient weights. Recent advancements like the QuIP family of methods [16, 17] achieve extreme compression ratios using VQ, yet these methods suffer from limited practicality, as their VQ scheme is not well supported by modern GEMM kernels for runtime acceleration. While these methods achieve strong performance at a given precision, they still exhibit an inherent trade-off between accuracy and efficiency across bit-widths. Moreover, they lack flexibility, as switching between precisions typically requires offline recalibration, making them unsuitable for dynamic edge scenarios with changing resource constraints.

Any-precision PTQ. To address the inflexibility of static schemes, recent works propose any-precision quantization, enabling a single model to support multiple bit-widths for dynamic system requirements. AnyPrecisionLLM [8] incrementally upscales a low-bit model to higher precisions, AnyBCQ [9] expands base quantization via per-precision scaling factors, and MatQuant [10] derives low-bit models by truncating higher-bit representations. However, these approaches remain limited in practice: they either rely on hardware-unfriendly vector quantization [8], introduce additional overhead from per-precision calibration parameters [9], or require offline repacking when increasing precision [10]. Moreover, they lack efficiency for runtime adaptation, as switching between precisions requires loading extra scaling factors and launching distinct kernels (e.g., from 3-bit to 4-bit). These limitations motivate our `MoBiQuant` framework, which mitigates precision-dependent outlier migration and enables efficient runtime precision switching without additional overhead.

3 Motivation

Limitations of static PTQ. Typical PTQ calibration process introduces a set of parameters $\mathbf{\Theta}_q$ to minimize the quantization error of the pretrained weights $\mathbf{W}$ at a target bit-width b. For example, OmniQuant [7] learns to clip and rescale $\mathbf{W}$ and SpinQuant [14] learns to rotate $\mathbf{W}$ to reduce outliers, respectively. Given a calibration set $\mathcal{X} = \{\mathbf{X}_i\}_{i=1}^{N}$, where each sequence $\mathbf{X}_i \in \mathbb{R}^{T\times d}$ consists of T tokens, the calibration process for static PTQ methods can be formulated as

$$\arg\min_{\mathbf{\Theta}_q} \mathbb{E}_{\mathbf{X}\sim\mathcal{X}} \mathcal{D}\big(\mathbf{X}, Q(\mathbf{W}|\mathbf{\Theta}_q, b)\big), \tag{1}$$

where $Q(\cdot)$ represents the weight quantizer and $\mathcal{D}(\cdot)$ denotes the quantization error function, respectively.

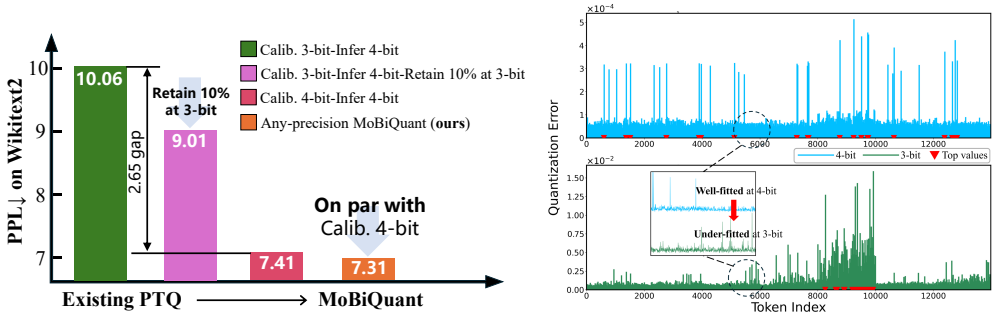

Figure 1: (Left) Existing PTQ methods poorly generalize under calibration–inference precision mismatch, e.g., inference at 4-bit with 3-bit calibrated parameters (green vs. red). Incorporating token-aware bit adjustment partially mitigates this degradation (pink). Our `MoBiQuant` improves generalization across precisions (orange). (Right) Per-token quantization error distribution at layer 5 in LLaMA3-8B: quantization outliers are highly non-uniform at each bit-width. Tokens well-fitted at 4-bit can become outliers at 3-bit, while 4-bit outliers may not be the primary error source at 3-bit.

The Eq. (1) objective causes the parameters $\mathbf{\Theta}_q$ to overfit for the selected target bit-width b. We illustrate this limitation in Fig. 1 (left): if we use the calibration parameters of 3-bit PTQ to quantize the LLaMA-3-8B model for 4-bit inference, the perplexity increases by 2.65 points compared to a 4-bit calibrated model. This clearly highlights the lack of generalization across quantization precisions.

Analysis of the poor generalization. Previous work [18] has shown that the token sensitivity is highly non-uniform. Consequently, the calibration process optimizes $\mathbf{\Theta}_q$ to specifically fit the distribution of sensitive tokens using the overall objective in (1). We pinpoint the poor generalization of calibration parameters to the distribution shift of outlier tokens. We plot the per-token quantization error distributions when the target weight precisions are 3/4-bit for the same query in Fig. 1 (right) and observe very different distributions. Specifically, tokens that are accurately represented by 4-bit weights may emerge as dominant outliers for 3-bit weights, whereas outliers with 4-bit precision may no longer be the primary source of quantization error for 3-bit PTQ. We define this phenomenon as *"outlier migration"* and it appears universally across different PTQ calibration methods either gradient-based OmniQuant [7] or Hessian-based AWQ [15] (see Appendices E.1 through E.3).

To further motivate our approach, we revisit the 3-bit calibrated 4-bit inference experiment, but now retain the top 10% of token outliers in the 3-bit calibration to be inferred with 3-bit weights. Counterintuitively, it shows that inferring tokens at a lower precision can yield higher overall performance (pink bar). This observation suggests a promising direction: ***token-adaptive precision adjustment*** *can improve the generalization of PTQ calibration parameters across bit-widths.*

4 The MoBiQuant Framework

Inspired by our previous observations, we present `MoBiQuant`, an any-precision quantization framework that mitigates *outlier migration* through token-aware bit-width adaptation while enabling efficient runtime precision switching. An overview of the framework is shown in Fig. 2.

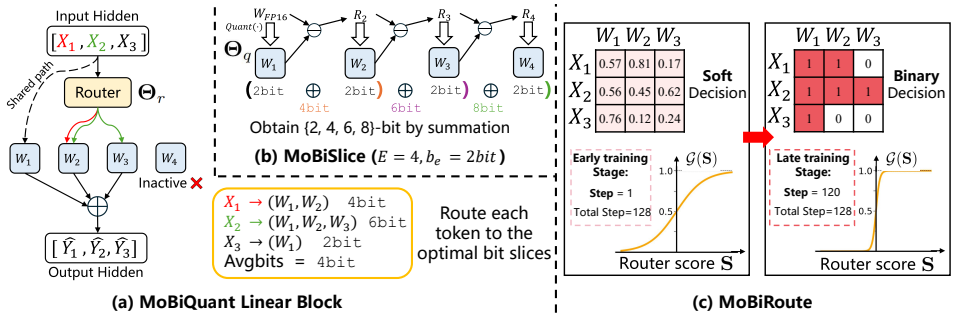

Figure 2: `MoBiQuant` enables token-adaptive any-precision inference for linear blocks in LLMs (a). It consists of two main components: `MoBiSlice` that supports multiple precisions by decomposing FP16 weights into bit slices via recursive quantization (b); and `MoBiRoute` with token-adaptive precision adjustment that is trained to route each token to the optimal bit slice under a budget (c).

4.1 Bit Slicing via Recursive Quantization

Achieving token-level granularity requires a model to efficiently support multiple quantization levels (e.g., 2–8-bit range). A naïve solution with separate models for each precision incurs high memory overhead. Existing any-precision approaches [8, 19, 10] explicitly derive multiple precision levels from a single seed model by weight slicing or scaling. However, each precision remains a monolithic unit: switching a token to a different precision requires an offline reloading of the seed model and re-slicing, followed by redeployment. This is ill-suited for latency-sensitive settings where all adaptation is expected to occur at runtime.

To address this limitation, we propose a novel many-in-one quantization scheme `MoBiSlice`, where multiple precisions can be efficiently obtained at runtime. Our `MoBiSlice` decomposes the weight matrix $\mathbf{W}$ of each linear layer in the LLM into E slices $\{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_E\}$, each containing a slice of b_e bits from the quantized weight. This decomposition is implemented by recursively quantizing the residuals as

$$\mathbf{R}_1 = \mathbf{W},\ \mathbf{W}_e = Q(\mathbf{R}_e \mid \mathbf{\Theta}_q, b_e),\ \text{and}\ \mathbf{R}_{e+1} = \mathbf{R}_e - \mathbf{W}_e,\ \forall e \geq 1, \tag{2}$$

where the first shared most significant bit (MSB) slice $\mathbf{W}_1$ is obtained by quantizing the original $\mathbf{W}$ and the remaining slices $\mathbf{W}_{e>1}$ quantize the residuals $\mathbf{R}_e$ between the weight and existing slices in a recursive fashion. All slices are quantized using scalings derived from the shared $\mathbf{\Theta}_q$, where the scaling factor of the $(e+1)$-th slice is obtained by dividing that of the e-th slice by 2^{b_e-1}. This avoids the training overhead of multiple sets of scaling factors in existing methods [8, 9]. Moreover, dequantization can be performed via efficient shift-and-add operations (Sec. 4.3). Consequently, a weight $\mathbf{W}^{(b)}$ at a target precision b can be constructed by summing a selection of k bit slices as

$$\mathbf{W}^{(b)} = \sum_{e=1}^{k} \mathbf{W}_e,\ \text{where}\ b = \sum_{e=1}^{k} b_e. \tag{3}$$

In this paper, we adopt a default configuration with $E = 4$ slices, each quantized to 2-bit, as shown in Fig. 2(b). In this way, weights with 4-bit and 6-bit precision can be constructed using 2 and 3 slices, respectively. This design enables any-precision inference using a fixed 2-bit kernel, thereby avoiding kernel relaunch overhead. `MoBiSlice` can be paired with existing PTQ pipelines, and we use OmniQuant [7] as our PTQ backbone in experiments. Further details on the `MoBiSlice` formulation and analysis are provided in Appendix B.

4.2 Runtime Precision Switching via Learned Token Routing

Given bit slices provided by `MoBiSlice`, we propose a lightweight `MoBiRoute` router that dynamically assigns a different number of slices to each token with binary decisions. Specifically, given a sequence of T tokens $\mathbf{X} \sim \mathcal{X}$, the routing score matrix $\mathbf{S} \in \mathbb{R}^{T \times E}$ is obtained as

$$\mathbf{S} = \mathcal{R}(\mathbf{X}, \mathbf{\Theta}_r), \tag{4}$$

where $\mathcal{R}(\cdot)$ is a learnable 2-layer MLP with the router parameters $\mathbf{\Theta}_r$, and each element of the score matrix $\mathbf{S}_{i,j}$ represents the confidence that the i-th token will require the j-th slice.

Two key challenges exist when training the router in Eq. (4):

Challenge 1: Learning binary routing decisions. To determine whether a token i should be routed to the j-th slice, one simple approach is to adopt a threshold-based routing strategy. However, fixed thresholds cannot adapt to a varying score distribution during the training stage. Therefore, our `MoBiRoute` learns a decision boundary implicitly using a differentiable gating function $\mathcal{G}(\cdot)$. When training is complete, $\mathcal{G}(\mathbf{S})$ acts as a learned binary mask applied to tokens. We formulate the proposed gating function as

$$\mathcal{G}(\mathbf{S}) = \sigma\left(\tau(t) \cdot \mathbf{S}\right),\ \tau(t) = \ln(L) / \left(\ln(L) - \ln(t)\right), \tag{5}$$

where σ is the conventional sigmoid, and τ is the temperature defined by the current training step t and the total number of steps L.

At the final step, the temperature $\tau(L) = \infty$, effectively converting the continuous gating function into a binary mask $\mathcal{G}(\mathbf{S}) = \mathbb{I}\left(\mathbf{S} > 0\right)$. The change in scores and the gating function during training are illustrated in Fig. 2(c). Using the learned mask, the input token $\mathbf{X}_i$ is routed through the selected slices, and the corresponding outputs are aggregated to produce the output token $\hat{\mathbf{Y}}_i$ as

$$\hat{\mathbf{Y}}_i = \sum\nolimits_{e=1}^{E} \mathbf{W}_e^{\top} \left(\mathcal{G}(\mathbf{S})_{i,e} \odot \mathbf{X}_i\right), \tag{6}$$

where input tokens are gated by the router's binary scores, and $\mathbf{W}_e$ is the bit slice with b_e bits.

Challenge 2: Generalizing across precisions. Any-precision inference requires the router to effectively accommodate a dynamic target precision budget (i.e., average bit-width). To achieve this, we apply a *target precision scheduling* mechanism in the training process to encourage the router to explore different precision combinations. Specifically, we control router convergence through a budget-aware regularization objective defined by

$$\mathcal{L}_{\text{reg}}(t) = \left(\text{AvgBits} - b(t)\right) \cdot \left\|\mathcal{G}(\mathbf{S})\right\|_1,\ b(t) = b_{\text{init}} - \left(b_{\text{init}} - b\right) \cdot \ln(t) / \ln(L), \tag{7}$$

where the term $\left(\text{AvgBits} - b(t)\right)$ adaptively encourages bit slice pruning when the current average bit-width exceeds the target, and promotes slice activation otherwise. The AvgBits in (7) is estimated during training by counting activated bit slices per token. We define entries with scores exceeding 0.5 in $\mathcal{G}(\mathbf{S})$ as active, which can be formulated as

$$\text{AvgBits} = \frac{1}{T} \sum\nolimits_{i=1}^{T} \sum\nolimits_{j=1}^{E} \mathbb{I}(\mathcal{G}(\mathbf{S})_{i,j} > 0.5)\, b_j, \tag{8}$$

where b_j denotes the bit slice size (e.g., 2 bits). The scheduling term $b(t)$ starts with a higher precision b_{init} (e.g., 8-bit) and gradually decays to the target b. This forces the router to explore a wide spectrum of token-slice assignments before converging to binary routing decisions. During training, the target b is a hyperparameter and is set to 3-bit by default in our experiments. While `MoBiQuant` remains robust across different target precisions, varying the target bit-width may introduce minor performance variations, which we analyze in Appendix D.3.

Joint optimization. Our framework consists of two sets of trainable parameters: $\mathbf{\Theta}_q$ and $\mathbf{\Theta}_r$ for `MoBiSlice` and `MoBiRoute`, respectively. They are jointly optimized during the PTQ process by

$$\arg\min_{\mathbf{\Theta}_{q,r}} \mathbb{E}_{\mathbf{X} \sim \mathcal{X}} \left[\mathcal{D}\left(\mathbf{W}^{\top}\mathbf{X}, \hat{\mathbf{Y}} | \mathbf{\Theta}_{q,r}\right) + \lambda\, \mathcal{L}_{\text{reg}}\right], \tag{9}$$

where $\mathcal{D}$ is the L_2 quantization error i.e. $\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2$. We freeze all weights from the pretrained LLM and calibrate only $\mathbf{\Theta}_q$ and $\mathbf{\Theta}_r$. We also treat $\mathbf{W}_1$ as a *shared-expert slice* such that tokens always pass through for stable training. We replace all linear layers in LLM transformer blocks with the proposed `MoBiQuant` block. Furthermore, we adopt a layer-wise calibration strategy from [7]. A detailed calibration strategy is presented in the Appendix A.

Efficient runtime precision switching. At the end of training, the score is away from zero if the router is confident about selection or exclusion of a bit slice, whereas the score remains at 0 with the lack of confident decision. Importantly, this separation allows us to achieve arbitrary precision adjustment at runtime by shifting the threshold of the score conversion process. During inference, we convert the router score into a binary selection mask using an adjustable threshold δ as

$$\mathcal{G}_\delta(\mathbf{S}) = \mathbb{I}\left((\mathbf{S} - \delta) > 0\right), \tag{10}$$

where δ can be globally adjusted for all layers at runtime. Increasing δ reduces the number of activated slices per token, thereby lowering the effective precision, and vice versa. This design enables `MoBiQuant` to adapt precision at runtime without repeated weight repacking or dequantization.

4.3 MoBiQuant Kernel Design

The `MoBiQuant` kernel in Fig. 3 enables efficient inference by addressing the following challenges:

❶ **Redundant Memory Access.** Conventional static low-bit kernels typically load all slices regardless of the runtime precision, leading to unnecessary memory bandwidth and limiting the inference speedups. To address this issue, we adopt the bit-major packing representation from [8, 9], which decomposes k-bit weights into independent bit-planes. Then, only the required slices are fetched that enables on-demand memory access with proportional speedups.

Although `MoBiQuant` also relies on bit-major packing, it differs from AnyPrecisionLLM [8] by avoiding bit transposition and centroid lookup. Instead, our kernel performs Binary Matrix Multiplication (BMMA) directly on packed bit-planes. Compared to AnyBCQ [9], our kernel uses a shared set of scaling factors across slices. During dequantization, the lower-bit slice is first shifted and added to the higher-bit slice at the bit-plane level, then multiplied by the shared scaling factor as shown in Fig. 3. By default, the `MoBiSlice` adopts a configuration with four 2-bit slices. These design choices in the `MoBiQuant` kernel lead to an improved end-to-end throughput.

❷ **Router Overhead.** `MoBiRoute` is an MLP-based router that introduces additional GEMM operations. To minimize its latency overhead, we employ a persistent single-kernel design that fuses all projections in a layer into a single GPU launch with shared-memory input reuse. This eliminates redundant kernel launches and reduces memory traffic.

❸ **Non-contiguous Memory Access and Load Imbalance.** Since each token is dynamically assigned to different bit slices, executing BMMA across slices requires gathering tokens from non-contiguous memory locations, which degrades memory bandwidth utilization. To address this, we apply token permutation after routing. Then, the tokens that are assigned to the same bit slice are stored contiguously, thereby improving memory bandwidth utilization.

In addition, the activation patterns of the bit slices are inherently skewed e.g., low-precision slices $\mathbf{W}_{1,2}$ are loaded more frequently. To mitigate the imbalance, we employ a parallel execution strategy using independent CUDA streams to overlap the computation of the first slice with subsequent ones.

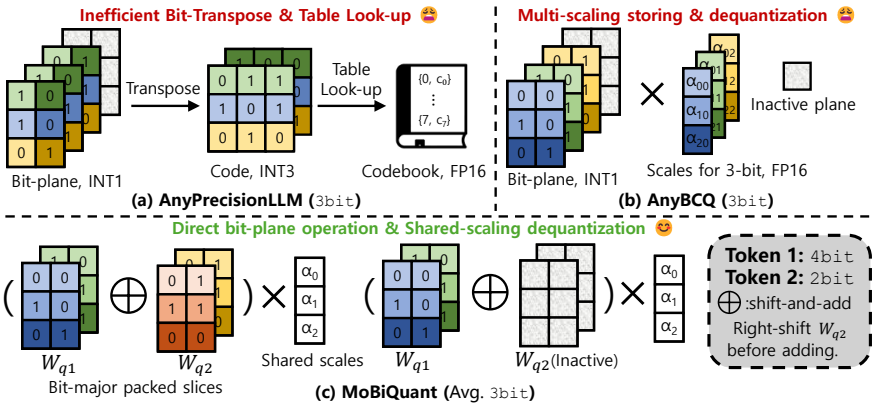

Figure 3: (a) AnyPrecisionLLM [8] with the inefficient bit-plane transposes and table lookups, (b) AnyBCQ [9] with the multi-scale storage and dequantization overhead, and (c) our `MoBiQuant` that performs direct bit-plane operations with the shared scales for efficient any-precision dequantization.

5 Experiments

5.1 Experimental Setup

Models and benchmarks. We conduct experiments using several open-source LLMs including LLaMA2-7B/13B [1], LLaMA3-8B, and LLaMA3.2-1B/3B. We evaluate the quantized models by reporting perplexity (PPL) scores on WikiText2 [20]. In addition, we assess performance on six zero-shot commonsense reasoning tasks: BoolQ [21], PIQA [22], HellaSwag [23], WinoGrande [24], and both ARC-Easy and ARC-Challenge [25] as well as the reasoning benchmark GSM8K [26].

Baselines and implementation. We focus on weight-only quantization and compare `MoBiQuant` to static PTQ methods such as GPTQ [12], OmniQuant [7], AWQ [15], SmoothQuant [13], SpinQuant [14], QuaRot [27], VQ-based QUIP# [17] and QTIP [28]. We also compare with any-precision baselines: AnyPrecisionLLM [8] and AnyBCQ [9]. All kernel results are measured on NVIDIA A100 GPUs with CUDA 12.9. We employ 128 sequences from WikiText2 as the calibration set. We provide additional details about the experiments in Appendix C.

5.2 Main Results

Improved Cross-Bit Generalization. We first investigate whether `MoBiQuant` improves the cross-bit generalization of our integrated PTQ baseline, OmniQuant [7]. Specifically, we set the calibration bit-width for both OmniQuant and `MoBiQuant` to 3-bit and evaluate their performance across unseen higher and lower bit-widths. As shown in Fig. 4, `MoBiQuant` consistently outperforms OmniQuant at all unseen bit-widths across various LLaMA models. Importantly, even at extremely low 2–3-bit range, `MoBiQuant` maintains smooth and stable precision scaling, whereas OmniQuant exhibits significant performance degradation. We show that `MoBiQuant` can also improve rotation-based PTQ methods (Appendix E.3). Similar trend is also observed in activation quantization (Appendix E.4).

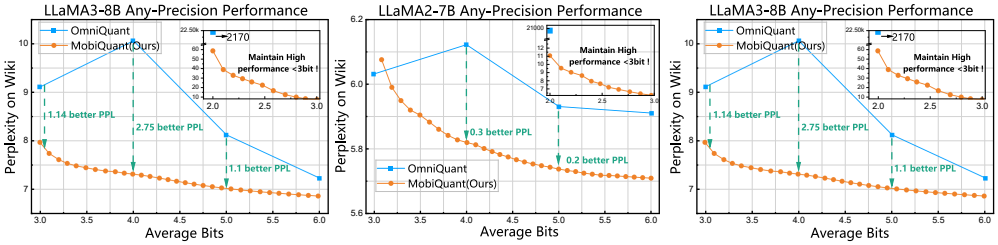

Figure 4: **Any-precision inference performance comparison.** `MoBiQuant` outperforms the PTQ baseline across multiple unseen target bit-widths on various LLaMA models, maintaining smooth and fine-grained precision scaling even at extremely low 2–3-bit precision range.

Table 1: End-to-end comparison with AnyPrecisionLLM (AP), AnyBCQ (ABCQ), QUIP# and QTIP. `MoBiQuant` achieves higher decoding throughput while maintaining a favorable performance.

LLaMA	Quant. bits	WikiText2, PPL ↓					Throughput, tokens/sec. ↑				
		Static VQ		Any-precision			Static VQ		Any-precision		
		QUIP#	QTIP	AP	ABCQ	MoBiQ	QUIP#	QTIP	AP	ABCQ	MoBiQ
2-7B	2	12.3	**5.86**	2e3	15.38	**10.91**	191	**218**	279	312	**404**
	3	6.19	**5.28**	6.13	6.25	**6.07**	177	**196**	244	268	**311**
	4	5.66	**5.17**	6.05	5.91	**5.82**	155	**189**	211	225	**256**
3-8B	2	15.20	**7.25**	1e3	**19.01**	58.12	157	**180**	228	247	**352**
	3	8.27	**7.05**	8.60	8.08	**7.97**	142	**158**	195	216	**257**
	4	6.64	**6.06**	**6.70**	6.84	7.31	126	**154**	169	182	**209**

Comparison with any-precision baselines. We further compare `MoBiQuant` with the recent AnyPrecisionLLM [8] and AnyBCQ [9] in Tab. 1, considering both the performance–precision trade-off and the decoding throughput. At 2-bit, `MoBiQuant` significantly outperforms [8], reducing WikiText2 perplexity from 2e3 to 10.91 on LLaMA2-7B and from 1e3 to 58.0 on LLaMA3-8B. At 3/4-bit, `MoBiQuant` also consistently surpasses state-of-the-art AnyBCQ. Importantly, `MoBiQuant` kernel without centroid table lookup and multi-scaling dequantization leads to substantially higher decoding throughput with an average speedup of 33.8% over AnyPrecisionLLM and 22.8% over AnyBCQ.

Table 2: The proposed any-precision MoBiQuant **matches** or **surpasses** the static scalar quantization (SQ) PTQ baselines with the same number of average bits on WikiText2, PPL.

Method	Type	LLaMA2-7B		LLaMA2-13B		LLaMA3.2-1B		LLaMA3.2-3B		LLaMA3-8B	
		3-bit	4-bit	3-bit	4-bit	3-bit	4-bit	3-bit	4-bit	3-bit	4-bit
FP16	—	5.5		5.0		13.4		9.2		6.1	
SmoothQuant	Static SQ	8.62	7.50	5.43	6.10	2e2	2e2	3e2	1e2	12.50	10.70
AWQ		6.24	5.83	5.32	5.07	20.69	15.02	15.51	12.76	8.22	7.36
GPTQ		6.29	5.90	5.42	5.60	20.81	15.19	15.77	13.14	8.28	7.43
SpinQuant		6.13	**5.60**	5.24	**5.00**	19.62	14.45	**14.73**	11.17	8.14	**6.40**
QuaRot		6.13	5.61	**5.22**	5.03	20.15	14.77	14.82	11.55	9.01	7.20
OmniQuant		**6.03**	5.74	5.28	5.02	23.92	16.79	15.21	12.71	9.11	7.41
MoBiQuant	Any-prec.	6.07	5.82	5.31	5.08	**19.40**	**14.02**	14.76	**9.41**	**7.97**	7.31

We also compare with the recent VQ-based static methods QuIP# [17] and QTIP [28]. Although these methods excel in bits-performance trade-off, their actual decoding throughput is limited by the centroid table lookups. For example, 2-bit QuIP and QTIP have 191 and 218 tokens/sec. throughput with 12.3 and 5.86 perplexity, respectively. In contrast, the 4-bit MoBiQuant with higher 256 tokens/sec. throughput achieves lower 5.82 perplexity and, thus, has better performance–throughput trade-off.

Comparison with scalar PTQ methods. We evaluate prior static scalar PTQ baselines that have been separately trained for fixed 3- and 4-bit precisions. For fair comparison, we keep our MoBiQuant elastic during inference but restrict its average precision to be either 3- or 4-bit. As shown in Tab. 2, MoBiQuant consistently matches the performance of state-of-the-art PTQ methods on LLaMA2 models and even surpasses them on LLaMA-3 models. Zero-shot and GSM8K results are reported in Appendix E.5. Quantitative results demonstrate that the MoBiQuant attains the performance of static quantization methods while preserving flexible precision adjustment.

5.3 Analytical Results

Improved generalization with reduced outlier migration. As discussed in Section 3, the failure to generalize across precisions mainly stems from *outlier migration*. To verify that MoBiQuant mitigates this issue, we plot the token-wise quantization error increments caused by switching OmniQuant from 4-bit calibration to 3-bit inference, and compare it with the router scores produced by MoBiRoute. As shown in Fig. 5 (left), tokens with larger error increments receive higher router scores and, thus, are retained at higher precision, while less sensitive tokens are routed to 2-bit computation. This shows that the router accurately captures token sensitivity to precision changes. Fig. 5 (right) further visualizes the token-wise quantization error distributions with different precisions. Compared with static PTQ in Fig. 1, MoBiQuant exhibits more consistent error distributions across bit-widths with reduced *outlier migration*, leading to improved cross-precision generalization.

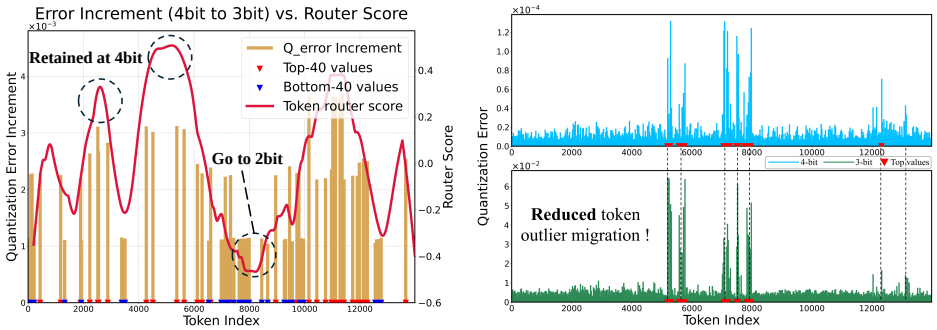

Figure 5: **Visualization of outlier migration mitigation.** MoBiQuant router scores correlate with the actual token-wise error increments under precision switching (left), and it produces more consistent token-wise error distributions across bit-widths with reduced outlier migration (right).

Token- and block-wise assignments. We collect statistics on token bit-width assignments for any-precision inference. As shown in Fig. 6 (right), tokens have assignment distributions that shift with target precision e.g., most tokens are assigned to 2- or 4-bit precisions under 3-bit inference,

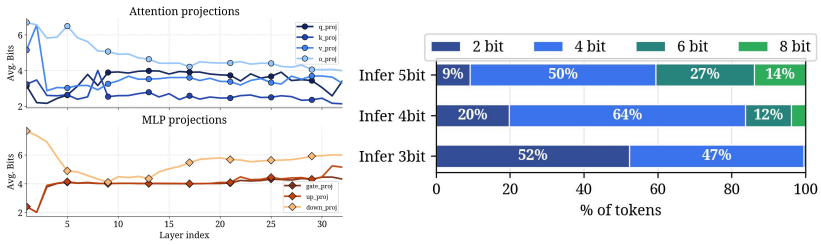

Figure 6: (Left) Average precision assignments of different linear blocks by MoBiQuant, showing block-level variation and higher deviations in initial layers. (Right) Precision distributions for any-precision inference, showing heterogeneous token assignments across target bit-widths.

whereas the majority of tokens shift toward the 4–6-bit range for the 5-bit inference. Overall, the average precision remains within the prescribed target token-aware bit adjustment

We also observe that the average precision varies across different linear blocks, even though each layer is trained for the same target bit-width. As shown in Fig. 6 (left), MLP blocks exhibit divergent assignments across layers, and the first layers tend to have deviations in bit-width assignments. This pattern is consistent with the [29] study that allocates different precisions to blocks based on their sensitivity, indicating that MoBiQuant also benefits from block-level precision adjustments.

5.4 Kernel Evaluation

We evaluate MoBiQuant kernel in real-world single-batch decoding scenarios on LLaMA2-7B. We first compare the end-to-end decoding latency against FP16 and the latest low-bit kernel ABQ-LLM [30] in Fig. 7 (left). With the same average precision, MoBiQuant consistently outperforms the latter baseline and achieves $4\times$ end-to-end latency reduction for different decoding lengths when compared to FP16. Since the router introduces additional parameters and operations, we report the latency breakdown for single token decoding in Fig. 7 (middle). The routing and packing overhead remains small, accounting for only 18% and 10% of total latency for 4- and 8-bit inference, respectively. These results show that the routing overhead is negligible, while the overall acceleration from low-bit computation remains significant. Finally, we evaluate the memory savings of MoBiQuant in Fig. 7 (right). MoBiQuant supports multiple precisions within a single model, reducing the total memory footprint by up to $3.5\times$ compared to a separate multi-precision deployment.

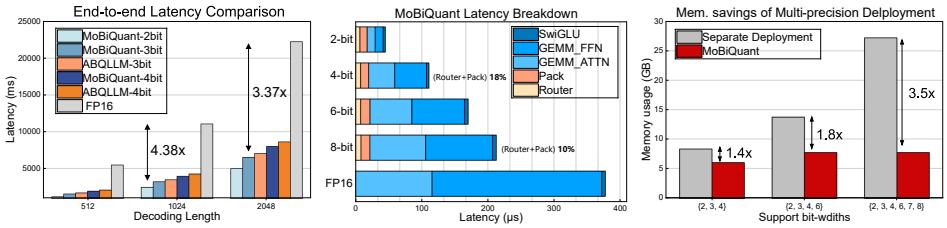

Figure 7: **Kernel evaluation of** MoBiQuant**.** (Left) End-to-end decoding latency comparison. (Middle) Latency breakdown of MoBiQuant. (Right) Memory savings under multi-precision deployment.

5.5 Ablation Study

We evaluate the sensitivity to the calibration data by replacing WikiText2 with two alternative datasets: C4 [31] and PTB [32]. MoBiQuant consistently achieves better results in most calibration–evaluation pairs, demonstrating robustness to calibration set selection. The full results are shown in the Appendix D.1. We also report ablation studies on router regularization scheduling in Appendix D.2, and examine router generalization for different target precision training in Appendix D.3.

6 Conclusion

In this paper, we identify the "*outlier migration*" in conventional PTQ methods, which limits generalization of calibration across precisions. To address it, we proposed MoBiQuant, a quantization framework that integrates MoBiSlice for fine-grained precision selection and MoBiRoute for token-adaptive routing. Extensive experiments demonstrate its effectiveness and practicality.

References

[1] Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023.

[2] OpenAI et al. Gpt-4 technical report, 2023.

[3] Gemini-Team et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.

[4] DeepSeek-AI et al. Deepseek-v3 technical report, 2025.

[5] Soumye Singhal et al. Llama-Nemotron: Efficient reasoning models. In *The Exploration in AI Today Workshop at ICML*, 2025.

[6] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *ICLR*, 2023.

[7] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *ICLR*, 2024.

[8] Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. Any-precision LLM: Low-cost deployment of multiple, different-sized LLMs. In *ICML*, 2024.

[9] Gunho Park, Jeongin Bae, Beomseok Kwon, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. AnyBCQ: Hardware efficient flexible binary-coded quantization for multi-precision LLMs. In *ICLR*, 2026.

[10] Pranav Ajit Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. In *ICML*, 2025.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *NeurIPS*, 2022.

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

[13] 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 *ICML*, 2023.

[14] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations. In *ICLR*, 2025.

[15] 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. In *Proceedings of Machine Learning and Systems*, 2024.

[16] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. QuIP: 2-bit quantization of large language models with guarantees. In *NeurIPS*, 2023.

[17] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *ICML*, 2024.

[18] Mengzhao Chen, Yi Liu, Jiahao Wang, Yi Bin, Wenqi Shao, and Ping Luo. PrefixQuant: Static quantization beats dynamic through prefixed outliers in LLMs. *arXiv:2410.05265*, 2024.

[19] Haodong Wang, Qihua Zhou, Zicong Hong, and Song Guo. D2MoE: Dual routing and dynamic scheduling for efficient on-device MoE-based LLM serving. In *Proceedings of the Annual International Conference on Mobile Computing and Networking*, 2025.

[20] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

[21] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[22] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

[23] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *AAAI*, 2019.

[24] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 2021.

[25] 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:1803.05457*, 2018.

[26] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021.

[27] 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. In *NeurIPS*, 2024.

[28] Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. QTIP: Quantization with trellises and incoherence processing. In *NIPS*, 2024.

[29] Coleman Hooper, Charbel Sakr, Ben Keller, Rangharajan Venkatesan, Kurt Keutzer, Sophia Shao, and Brucek Khailany. FGMP: Fine-grained mixed-precision weight and activation quantization for hardware-accelerated LLM inference. *arXiv:2504.14152*, 2025.

[30] Chao Zeng, Songwei Liu, Yusheng Xie, Hong Liu, Xiaojian Wang, Miao Wei, Shu Yang, Fangmin Chen, and Xing Mei. ABQ-LLM: Arbitrary-bit quantized inference acceleration for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[31] 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.

[32] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):154, 2020.

[33] Jinhee Kim, Seoyeon Yoon, Taeho Lee, Joo Chan Lee, Kang Eun Jeon, and Jong Hwan Ko. TruncQuant: Truncation-ready quantization for dnns with flexible weight bit precision. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2025.

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

Appendix

Contents

A Overall Algorithm of MoBiQuant

As shown in Alg. 1, `MoBiQuant` performs layer-wise quantization by matching each quantized layer to a full precision reference output using a two-stage iterative process. For each layer, it first runs the original full-precision layer to obtain full precision outputs, then initializes the shared quantization parameters $\mathbf{\Theta_q}$ and the router $\mathbf{\Theta_r}$. In *Stage 1*, it stabilizes the first slice path by optimizing the quantization parameters to minimize the reconstruction error to the full-precision target. In *Stage 2*, it initializes residual slices' quantization parameters from *Stage 1*, computes the router score for each token, and produces the overall quantized outputs. This stage jointly optimizes the objective that combines reconstruction loss with a router score regularization loss. After the optimization converges for the layer, `MoBiQuant` commits the learned quantization parameters, propagates both full-precision and quantized outputs to the next layer, and repeats the same procedure until all layers are quantized.

B MoBiSlice Construction: Formulation and Further Analysis

In this section, we formalize the `MoBiSlice` quantization scheme and describe how we decompose a full-precision weight into a set of composable low-bit slices, enabling a many-in-one representation for multiple precisions.

Quantizer design for bit slice accumulation. `MoBiSlice` requires that precision be adjustable by either accumulating residual bits or truncating them. To satisfy this property, we adopt a floor-aligned mapping following the truncation-ready quantization principle [33], where a lower precision code is obtained by simply dropping least significant bits (LSB) rather than re-rounding. Concretely, for

bit-width b, scale s, and continuous zero point z, we quantize as

$$x_{\text{int}} = \text{clamp}\Big(\Big\lfloor \frac{x}{s} + z \Big\rfloor, 0, 2^b - 1\Big), \tag{11}$$

$$x_{\text{deq}} = s\,(x_{\text{int}} - z + 0.5)\,. \tag{12}$$

The floor operator enforces hierarchical nesting of integer codes, so switching bit width corresponds to adding or dropping bit slices without changing previously formed higher order bits. The $+0.5$ dequantization shift centers each bin, which reduces bias when residual slices are accumulated.

Algorithm 1 MoBiQuant Calibration

Input: Input tokens $\mathbf{X}$, model $\mathcal{M}$ with L layers $\{f_{\text{fp}}^{(\ell)}\}_{\ell=1}^{L}$, epochs `ep`, number of bit slices E, regularization weight λ

Output: Quantized model $\widehat{\mathcal{M}}$

Notation: We omit the layer superscript (ℓ) for readability.
$\mathbf{H}_{\text{fp}}, \mathbf{H}_{\text{q}}$: Full precision and quantized input for layer ℓ
$\mathbf{Y}_{\text{fp}}, \mathbf{Y}_{\text{q}}$: Full precision and quantized output for layer ℓ
$\mathbf{G}$: Router scores, $\odot$: element wise product

Initialize activations $\mathbf{H}_{\text{fp}}, \mathbf{H}_{\text{q}} \leftarrow \mathbf{X}$
for $\ell = 1$ **to** L **do**
  FP output $\mathbf{Y}_{\text{fp}} \leftarrow f_{\text{fp}}(\mathbf{H}_{\text{fp}}; \mathbf{W})$
  Initialize quantization parameters $\mathbf{\Theta_q}$ and router parameters $\mathbf{\Theta_r}$
  for $t = 1$ **to** `ep` **do**
    Stage 1: First slice (FS) stabilization
    $\mathbf{Y}_{\text{FS}} \leftarrow f_{\text{FS}}(\mathbf{H}_{\text{q}}; \mathbf{\Theta_q})$ ▷ First slice-only forward pass
    $\mathcal{L}_{\text{FS}} \leftarrow \|\mathbf{Y}_{\text{FS}} - \mathbf{Y}_{\text{fp}}\|_2^2$ ▷ match FP reference output
    Update $\mathbf{\Theta_q}$
    Stage 2: Joint training
    Slices $\{\Theta_{q,e}\}_{e=1}^{E} \leftarrow \text{Derive}(\mathbf{\Theta_q})$ ▷ construct MoBi slices
    Router score $\mathbf{S} \leftarrow \text{Router}(\mathbf{H}_{\text{q}}; \mathbf{\Theta_r})$ ▷ token wise gating weights
    $\mathbf{Y}_{\text{q}} \leftarrow \mathbf{Y}_{\text{FS}} + \sum_{e=1}^{E} \mathbf{G}_e \odot f_e(\mathbf{H}_{\text{q}}; \Theta_{q,e})$ ▷ sum all slices' outputs
    $\mathcal{L}_{\text{MoBi}} \leftarrow \|\mathbf{Y}_{\text{q}} - \mathbf{Y}_{\text{fp}}\|_2^2 + \lambda\,\text{Reg}(\mathbf{S}, \mathbf{\Theta_r})$ ▷ reconstruction loss and router regularization
    Update $\mathbf{\Theta_q}, \mathbf{\Theta_r}$ ▷ joint update of quantization and router params
  end for
  $\widehat{\mathbf{W}} \leftarrow \text{Quantize}(\mathbf{W}; \mathbf{\Theta_q}, \mathbf{\Theta_r})$
  $\mathbf{H}_{\text{fp}} \leftarrow \mathbf{Y}_{\text{fp}}$
  $\mathbf{H}_{\text{q}} \leftarrow f_{\text{q}}(\mathbf{H}_{\text{q}}; \widehat{\mathbf{W}})$
end for
return $\widehat{\mathcal{M}}$

To ensure consistent refinement across slices, `MoBiSlice` iteratively refines the scaling factor so that each slice represents progressively finer increments. Let (s_0, z_0) be the calibrated parameters of the first slice. After assigning b_e bits to slice e, the next slice refines the resolution as $s_{e+1} = s_e / 2^{b_e}$. Finally, while the first slice uses the calibrated zero point z_0, slices fix $z_e = 2^{b_e - 1}$ for $e \geq 1$, placing the midpoint code at the center of the integer range so that positive and negative residual corrections are represented symmetrically, which avoids systematic drift during slice accumulation. Together, the floor aligned mapping, scale refinement, and centered residual quantization make the slice decomposition compatible with truncation based precision control, enabling stable elastic reconstruction by activating a subset of slices.

Bias and Error Bounds for Bit Slice Activation. We now analyze truncation-based bit slice activation and show that activating residual slices performs a true refinement: it leaves the coarse quantization code unchanged and only adds a bounded zero-mean remainder term that recovers finer

detail. First, we assume a two slice scenario, consisting of a shared base slice with $(b-k)$ bits and a single k bit residual slice that can be activated or dropped.

Let W be the original weight. The base slice produces an MSB code with parameters (s_1, z_1), and the residual slice quantizes the residual R using parameters (s_2, z_2) derived from the base slice:

$$\mathrm{MSB}_{b-k} \;=\; \Big\lfloor \frac{W}{s_1} + z_1 \Big\rfloor, \tag{13}$$

$$\mathrm{LSB}_k \;=\; \Big\lfloor \frac{R}{s_2} + z_2 \Big\rfloor, \qquad s_2 = \frac{s_1}{2^k}, \qquad z_2 = 2^{k-1}. \tag{14}$$

Using the centered dequantization, the two-slice reconstruction is:

$$\widehat{W} = s_1(\mathrm{MSB}_{b-k} - z_1 + 0.5) \;+\; s_2(\mathrm{LSB}_k - z_2 + 0.5)\,. \tag{15}$$

Although the reconstruction in Eq. (15) is written for a base $(b-k)$ bit slice plus a k bit residual slice, our goal is to characterize *any* intermediate precision obtained by activating only a subset of the residual information. We therefore introduce a generic truncation level p that drops p least significant bits from the merged code. This captures all partial activation cases: $p=0$ corresponds to using all k residual bits, and larger p corresponds to deactivating progressively more of the finest bits. We truncate p least significant bits of the merged integer code $\mathrm{INT}_8 = (\mathrm{MSB}_{b-k} \ll k) + \mathrm{LSB}_k$ using a right shift:

$$I \;=\; \mathrm{INT}_8 \gg p \;=\; \Big\lfloor \frac{\mathrm{INT}_8}{2^p} \Big\rfloor, \qquad \mathrm{INT}_8 \;=\; 2^p I + r, \qquad r \in \{0, 1, \ldots, 2^p - 1\} \tag{16}$$

Substituting Eq. (16) into Eq. (15) and noting that $s_2 = s_1/2^k$, we can rearrange terms to express $\widehat{W}$ as a coarse quantizer applied to the truncated code I, plus an additive truncation noise term determined by the dropped remainder r:

$$\begin{aligned}
\widehat{W} &= s_2\big(\mathrm{INT}_8 - 2^k z_1 + 0.5\big) \\
&= s_2\big(2^p I + r - 2^k z_1 + 0.5\big) \\
&= \underbrace{(2^p s_2)}_{s'}\left(I - \underbrace{2^{k-p} z_1}_{z'} + \underbrace{0.5}_{\mathrm{mid}} \right) \;+\; \underbrace{s_2\big(r + 0.5 - 2^{p-1}\big)}_{E_p}.
\end{aligned} \tag{17}$$

Therefore, truncating p least significant bits is equivalent to quantizing with coarser parameters:

$$s' = 2^p s_2, \qquad z' = 2^{k-p} z_1, \qquad \mathrm{mid} = 0.5, \tag{18}$$

Eq. (17) makes the role of p explicit: truncation does not change the coarse code I, and its only effect is an additive perturbation E_p determined by the discarded remainder r. Thus, analyzing E_p directly characterizes the impact of partially activating bit slices. Assuming the residue r is approximately uniform over $\{0, 1, \ldots, 2^p - 1\}$, we obtain:

$$\mathbb{E}[E_p] = s_2\big(\mathbb{E}[r] + 0.5 - 2^{p-1}\big) = s_2\left(\frac{2^p - 1}{2} + 0.5 - 2^{p-1} \right) = 0, \tag{19}$$

This yields $\mathbb{E}[E_p] = 0$, indicating that slice-based reconstruction does not introduce bias in expectation. Moreover, since $r \in \{0, 1, \ldots, 2^p - 1\}$,

$$-\Big(2^{p-1} - 0.5\Big) \;\le\; r + 0.5 - 2^{p-1} \;\le\; \Big(2^{p-1} - 0.5\Big), \tag{20}$$

which implies the error bound:

$$|E_p| \;\le\; s_2\Big(2^{p-1} - 0.5\Big) \;<\; 2^{p-1} s_2 \quad\Longrightarrow\quad \frac{|E_p|}{s_2 2^p} \;<\; \frac{1}{2}. \tag{21}$$

Eq. (21) implies that the truncation error is strictly smaller than one half step of the coarser quantizer. Equivalently, the perturbation E_p cannot move $\widehat{W}$ across a decision boundary of the base MSB quantizer, and therefore cannot flip any bit of the coarse code. As a result, activating residual slices performs a true residual refinement: it only fills in finer bit slices without altering the coarser representation, yielding an accurate and stable activation with controlled error. Therefore, this establishes that bit slice activation yields a guaranteed residual refinement: it adds finer information while preserving the coarse code, and its only effect is an additive truncation noise term that is bounded and zero-mean.

C Experiment Details

C.1 MoBiQuant Training Setup

We use weight-only quantization in all reported runs in the main paper, with `abits=16` and `group_size=128`, while the base bit slice uses `wbits=2`. Weight-activation quantization results and their experimental setup are reported in Appendix E.4. Slice decomposition is enabled by `slice_bits_list`, and our default configuration uses four bit slices with `slice_bits_list = 2 2 2 2`. Training proceeds for `epochs=20` and `nsamples=128`, with `batch_size=1` for all models. The total number of router annealing steps is used by the router temperature schedule and calculated as follows:

$$\texttt{global\_steps} = \left(\frac{\texttt{nsamples}}{\texttt{batch\_size}}\right) \cdot \texttt{epochs},$$

For each layer, we optimize three parameter groups with AdamW: learnable weight clipping parameters (LWC), learnable equivalent transformation parameters (LET), and MoBiQuant parameters. Group specific learning rates are denoted as follows: `lwc_lr`, `let_lr`, and `mobi_lr`. We typically use `lwc_lr` in the range `1e-3` to `1e-2`, and `mobi_lr` in the range `5e-6` to `4e-5`, depending on model size.

C.2 Flexible Inference and Effective Bit Control

A core feature of `MoBiQuant` is flexible inference, where the effective precision is controlled by activating a subset of residual bit slices. At evaluation time, this is exposed by `target_activation_ratio_eval` or `target_bit_eval`. When `target_bit_eval` is used, we convert it into a desired routing ratio based on the base MSB bits and the sum of residual slice bits:

$$\rho \;=\; \frac{\texttt{target\_bit\_eval} - b_{\mathrm{msb}}}{\sum_{e\geq 1} b_e}, \qquad b_{\mathrm{msb}} = \texttt{slice\_bits\_list}[0].$$

Hard binary gating is applied in evaluation using router scores and a threshold selected to satisfy the target ratio.

Layer-wise threshold calibration. To make the realized routing ratio stable across layers and modules, we calibrate a layer specific threshold from calibration samples whenever `target_bit_eval` requests residual activation. This calibration collects router scores over the calibration set and sets each layer's threshold to the appropriate quantile so that approximately a fraction ρ of routed experts are active. We set the calibration sample to be the same as `nsamples`.

C.3 Baseline Results

The baseline results reported in our experiments are either taken from the corresponding published papers or obtained by running their official open-source implementations. All reproduced experiments are conducted on NVIDIA A100 GPUs. Specifically, for OmniQuant, SpinQuant, AWQ, GPTQ, SmoothQuant, QUIP#, and QTIP, we run their open-source code to obtain WikiText2 PPL results on the LLaMA-3.2-1B, LLaMA-3.2-3B, and LLaMA-3-8B model series under a context length of 2048. In particular, for QUIP#, we reproduce and report its non-finetuned variant to align its computational cost with the other post-training quantization baselines. These reproduced results are used to ensure a consistent evaluation setting with our method. For kernel measurement, to ensure a consistent evaluation environment with our customized kernel, we uniformly evaluate the kernels of QUIP#, QTIP, AnyPrecisionLLM, and AnyBCQ on an NVIDIA A100 GPU with CUDA 12.9.

D Ablation Studies

D.1 Ablation of Calibration Dataset

We study the impact of different calibration datasets in Tab. 3. `MoBiQuant` remains stable across calibration choices and consistently achieves lower perplexity than OmniQuant on most calibration–evaluation pairs. In particular, WikiText2 and PTB calibration provide the strongest improvements

Table 3: **Ablation study on different calibration datasets for LLaMA3.2-1B-3bit.**

Dataset	Method	Perplexity (PPL ↓)			Downstream Tasks (Acc ↑)							
		Wiki	C4	PTB	ARC E	ARC E(n)	ARC C	ARC C(n)	BoolQ	Hella	Hella(n)	Wino
Wiki	Omni.	21.597	32.860	37.467	0.510	0.472	0.245	0.284	0.593	0.378	0.488	**0.551**
	MoBiQ	**18.947**	**27.924**	**34.915**	**0.521**	**0.481**	**0.261**	**0.294**	**0.613**	**0.382**	**0.497**	0.533
C4	Omni.	**22.611**	**33.198**	**38.096**	0.503	**0.470**	0.245	0.282	0.583	**0.378**	**0.488**	**0.548**
	MoBiQ	26.360	33.382	43.888	**0.522**	0.452	**0.282**	**0.293**	**0.619**	0.369	0.477	0.538
PTB	Omni.	22.689	33.519	35.353	0.503	0.471	0.241	0.281	0.586	0.376	0.488	**0.556**
	MoBiQ	**16.909**	**24.849**	**29.268**	**0.588**	**0.532**	**0.284**	**0.312**	**0.589**	**0.402**	**0.526**	**0.556**
Mix	Omni.	22.335	33.327	36.763	0.502	0.474	0.246	0.277	0.599	0.374	0.488	**0.553**
	MoBiQ	**21.070**	**30.072**	**33.646**	**0.534**	**0.502**	**0.258**	**0.300**	**0.612**	**0.382**	**0.500**	0.530

across WikiText2, C4, and PTB, while the mixed calibration setting maintains similar average bits with favorable cross-domain generalization.

On zero-shot tasks, `MoBiQuant` generally matches or outperforms OmniQuant across ARC Easy, ARC Challenge, BoolQ, HellaSwag, and WinoGrande [25, 21, 23]. The most consistent gains are again observed with PTB and WikiText2 calibration, indicating that `MoBiQuant` is robust to calibration set selection across both language modeling and downstream reasoning tasks.

D.2 Ablation of Scheduling Methods for Router Regularization

We conduct an ablation study to identify the most effective scheduling strategy for router regularization. We compare four schedules, Linear, Cosine, Exponential, and Logarithmic, and measure their impact on model perplexity (PPL) on WikiText2, C4, and PTB. As shown in Fig. 8, performance differences are most pronounced in the low-precision regime of 2.5 to 3.0 average effective bits. **In this range, both Logarithmic and Exponential scheduling consistently outperform the Linear and Cosine alternatives.** The relative ordering of the two best schedules is dataset dependent: Exponential offers a slight advantage on WikiText2, while Logarithmic performs better on C4 and PTB within the same bit-width range.

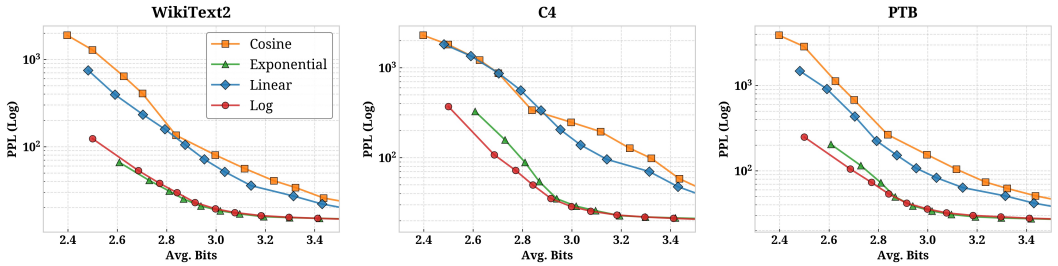

Figure 8: Comparison of PPL (↓) sweep between different scheduling methods across WikiText2, C4, PTB datasets.

We adopt a logarithmic schedule for regularization, as it naturally aligns with the logarithmic temperature annealing used in router gating and converges smoothly to the target precision during inference. This synchronization between routing sharpness and precision allocation stabilizes training and enables smoother adaptation to resource constraints.

D.3 Ablation of Different Training Target Bit

We train LLaMA3.2-1B with five target bit budgets (2.5, 3.0, 3.5, 4.0, and 5.0) to study how the training target affects router generalization across precisions. For each configuration, we sweep inference-time budgets and evaluate PPL on WikiText2, C4, and PTB.

As shown in Fig. 9, consistent trends are observed across datasets, with the largest differences appearing near the transition region around 3.0 average bits. Among all settings, the 3.0-bit target achieves

the best overall trade-off. Lower targets (e.g., 2.5-bit) improve low-bit performance but degrade higher-bit PPL, while higher targets (3.5–5.0-bit) significantly worsen sub-3.0-bit performance with only marginal gains at higher precisions. Elastic inference requires the router to accommodate a

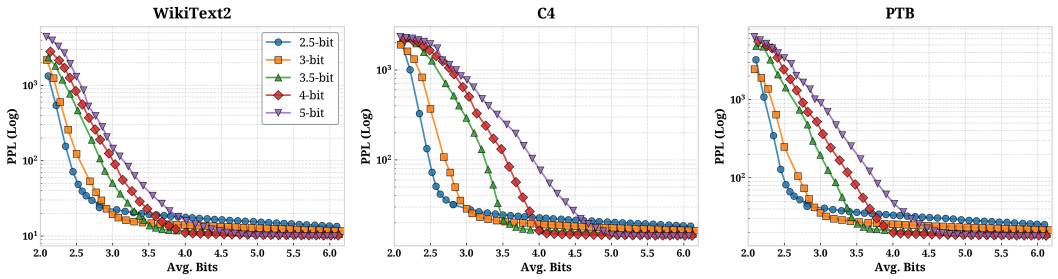

Figure 9: Comparison of PPL (↓) sweep between different target bit-widths for training across WikiText2, C4, PTB datasets.

changing target budget, and the target precision scheduling objective in Eq. 7 is designed to encourage exploration of diverse token slice allocations before converging. However, the training target still sets the operating point around which the router learns stable binary decisions: a low target biases the router toward early and persistent slice pruning, while a high target biases it toward maintaining many active slices. The 3.0 bit target sits near a balanced regime and improves calibration of routing decisions across a broader range of average bits. This motivates us to adopt 3.0 bits as the default training target throughout all experiments.

E Additional Experiments

E.1 Outlier migration in Non-Gradient-Based PTQ

To further validate the generality of our observation, we analyze the behavior of non-gradient-based PTQ methods under calibration–inference mismatch. We take AWQ [15] as a representative baseline and evaluate its generalization performance across different bit-widths. As shown in Table 4, AWQ

Table 4: Generalization gap (PPL) of AWQ on LLaMA2-7B.

Calibration Bits	Infer @ 3-bit	Infer @ 4-bit
3-bit	8.22	7.85
4-bit	8.45	7.36

exhibits noticeable degradation when the inference precision differs from the calibration setting, indicating limited generalization across bit-widths. In addition, we observe that the top outlier tokens overlap by only 41% between 3-bit and 4-bit settings, suggesting that token sensitivity shifts significantly across precisions. This behavior is consistent with the *outlier migration* phenomenon described in the main text.

E.2 Outlier migration on Mistral-7B

We further verify that the outlier migration phenomenon generalizes beyond the LLaMA family. Specifically, we observe similar behavior on Mistral-7B. For example, in a middle layer of Mistral-7B, the overlap of top token outliers between 3-bit and 4-bit is only 16%, indicating significant precision-dependent variation in token sensitivity. This migration leads to poor generalization for existing PTQ methods. As shown in Table 5, when the inference bit-width differs from the calibration setting, performance degrades substantially. In contrast, `MoBiQuant` effectively mitigates this issue.

E.3 Compatibility with rotation-based methods

We further investigate whether our method can be combined with rotation-based PTQ techniques. We consider two representative methods: (1) the non-gradient-based QuaRot [27], and (2) the gradient-based DuQuant [34].

Table 5: Mistral-7B performance gap (PPL) under calibration–inference mismatch.

Method	Calib 3 → Infer 4	Calib 4 → Infer 3
OmniQuant	5.61	23.19
MoBiQuant	**5.42**	**19.38**

Table 6: Generalization gap (PPL) of QuaRot on LLaMA2-7B.

Method	Calib 4 → Infer 4	Calib 4 → Infer 3
OmniQ	5.74	32.80
OmniQ + QuaRot	5.65	27.55
MoBiQuant + QuaRot	**5.65**	**22.24**

Table 6 shows that while QuaRot improves static performance, the degradation under bit-width mismatch remains significant. In contrast, combining `MoBiQuant` with QuaRot substantially improves generalization, demonstrating that our approach is complementary to rotation-based techniques. For

Table 7: Generalization performance of DuQuant and `MoBiQuant` on unseen W-A configurations.

	LLaMA2-7B			LLaMA3-8B		
W-A	W3A4	W4A4	W5A4	W3A4	W4A4	W5A4
DuQuant	6.77	6.51	6.49	17.12	15.31	12.12
MoBiQuant + DuQuant	6.81	**6.40**	**6.30**	**13.71**	**12.25**	**11.69**

DuQuant, we calibrate the model at W3A4 and evaluate on unseen weight–activation configurations. As shown in Table 7, `MoBiQuant` consistently improves generalization across all settings, with significant gains on LLaMA3-8B. Overall, these results demonstrate that (1) the outlier migration phenomenon is not limited to a specific PTQ method, and (2) `MoBiQuant` effectively improves generalization when combined with existing PTQ techniques.

E.4 Any-precision performance with Activation Quantization

The elasticity of `MoBiQuant` also extends to activation quantization, as shown in Fig. 10, where we report results on LLaMA2-13B. We keep the same slice decomposition as the weight-only setting and the same calibration hyperparameters. We also keep the same regularization parameter and learning rate. A key practical detail in this setting is the interaction between learnable equivalent transforms (LET) from OmniQuant [7] and router training. LET preserves the main linear path by transforming the input activation and compensating the linear weights so that the layer output remains equivalent. For a linear layer,

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{B}, \qquad \tilde{\mathbf{X}} = (\mathbf{X} - \boldsymbol{\delta}) \oslash \mathbf{s}, \qquad \tilde{\mathbf{W}} = \mathbf{s} \odot \mathbf{W}, \qquad \tilde{\mathbf{B}} = \mathbf{B} + \boldsymbol{\delta}\mathbf{W}, \tag{22}$$

where $\mathbf{X} \in \mathbb{R}^{T\times d}$ denote the input activations with T tokens and d input channels; $\mathbf{W} \in \mathbb{R}^{d\times m}$ the weight matrix; and $\mathbf{B} \in \mathbb{R}^m$ the bias. LET introduces a learnable per-channel shift $\boldsymbol{\delta} \in \mathbb{R}^d$ and scale $\mathbf{s} \in \mathbb{R}^d$, producing the transformed input $\tilde{\mathbf{X}}$ and the compensated parameters $\tilde{\mathbf{W}}$, $\tilde{\mathbf{B}}$ so that the layer output is preserved. However, the router is a separate linear branch whose weights are not re-parameterized by LET, so feeding $\tilde{\mathbf{X}}$ directly changes the routing function and can distort router scores. This is important because LET parameters evolve during calibration, so the router would otherwise see a moving input distribution, which can drive logits into saturation under sharp gating schedules and small calibration budgets, especially when the router input is quantized. For stability, we undo LET for the router input by reconstructing the activation in the original space before routing as follows:

$$\mathbf{X}_r = \tilde{\mathbf{X}} \odot \mathbf{s} + \boldsymbol{\delta}, \tag{23}$$

This keeps the router operating in a fixed activation space while LET continues to re-parameterize the main linear path, improving routing stability.

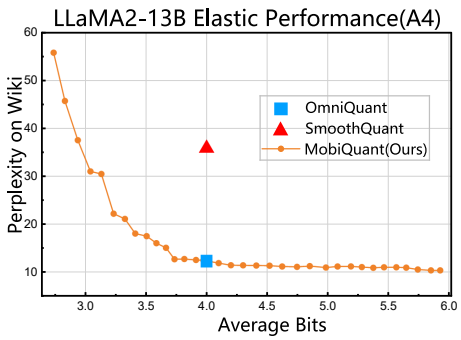

Figure 10: PPL (↓) tradeoff under 4 bit activation quantization on LLaMA2-13B. We compare with SmoothQuant [13] and OmniQuant [7] W4A4 baselines.

Table 8: Zero-shot comparison with static scalar quantization (SQ) methods. We report average accuracy over six zero-shot commonsense reasoning tasks.MoBiQuant matches or surpasses static PTQ baselines. "-" denotes that no direct results are reported in the published papers.

Method	Avg. Bits	Type	LLaMA-2-7B 0-shot Avg.(↑)	LLaMA-2-13B 0-shot Avg.(↑)	LLaMA-3.2-1B 0-shot Avg.(↑)	LLaMA-3.2-3B 0-shot Avg.(↑)	LLaMA-3-8B 0-shot Avg.(↑)
Floating-point	16	-	69.68	73.19	59.42	66.71	74.19
RTN	4	Static SQ	64.72	64.19	-	-	70.40
SmoothQuant			63.87	67.46	-	-	67.47
AWQ			**69.40**	72.64	-	-	**72.64**
GPTQ			65.54	70.22	-	-	70.21
SpinQuant			68.00	**72.68**	-	-	72.62
OmniQuant			68.41	71.68	**56.99**	**63.8**	71.66
MoBiQuant	3.9-4.0	AnyPrec.	67.22	71.54	55.49	63.2	71.84

E.5 Additional Comparison with Static PTQ Methods

We further compare elastic `MoBiQuant` against static PTQ baselines on downstream zero-shot evaluation. We report average accuracy over six zero shot commonsense reasoning tasks in Tab. 8, where prior methods are evaluated at fixed 4-bit precision and `MoBiQuant` is restricted to the 3.9 to 4.0-bit regime for a fair comparison. `MoBiQuant` is consistently competitive and improves over multiple widely used static PTQ baselines, while remaining close to the strongest results. In particular, `MoBiQuant` delivers clear gains over several static baselines on LLaMA2 models, and on LLaMA3-8B it matches or exceeds a number of strong static PTQ methods on the same task suite. Overall, these results support that `MoBiQuant` preserves elasticity while maintaining strong zero shot task performance at 4-bit precision. We report the GSM8K performance in Tab. 9. As a multi-precision method, `MoBiQuant` even outperforms static 4-bit OmniQuant, showing strong robustness across tasks.

F Limitations

While `MoBiQuant` demonstrates strong accuracy–efficiency trade-offs across multiple model families and bit budgets, several limitations remain. First, our experiments are primarily conducted on small- to medium-scale LLMs (up to 13B parameters). We have not yet evaluated the method on ultra-large

Table 9: GSM8K performance of LLaMA3.2-1B under 4-bit setting.

Method	**Flexible-Extract**	**Strict-Match**
FP16	46.47%	42.15%
OmniQuant-4bit	30.86%	30.86%
Ours (Elastic)	**32.83%**	**32.45%**

models such as 70B-scale LLMs due to the substantial computational and memory requirements of training and benchmarking elastic multi-precision quantization systems at that scale. We leave large-scale evaluation as future work. Second, although we implement optimized custom kernels for our method, the current system has not yet been fully integrated with mature production inference frameworks such as vLLM or FlashInfer. Finally, our evaluation mainly focuses on single-request or small-batch decoding scenarios, which are common in latency-sensitive edge and local deployment settings. The behavior of token-wise adaptive precision under large-scale cloud serving workloads with heterogeneous batching strategies remains an important direction for future investigation.