Q-Palette: Fractional-Bit Quantizers Toward Optimal Bit Allocation for Efficient LLM Deployment

Deokjae Lee[1,2] **Hyun Oh Song**[1,2*]
[1]Seoul National University, [2]Neural Processing Research Center
{bdbj, hyunoh}@mllab.snu.ac.kr

Abstract

We study weight-only post-training quantization (PTQ), which quantizes the weights of a large language model (LLM) without retraining, using little or no calibration data. Weight-only PTQ is crucial for reducing the memory footprint and latency of LLM inference, especially in memory-bound, small-batch inference scenarios, such as personalized inference on edge devices. Despite its importance, irregular weight distributions with heavy-tailed outliers in LLMs complicate quantization, recently motivating rotation-based methods that transform weights into near-Gaussian distributions, which are more regular with fewer outliers, thereby reducing quantization error. In this work, we first derive the information-theoretically optimal bit allocation for Gaussianized weights under given bit budgets, revealing that fine-grained fractional-bit quantizers approaching the Gaussian distortion-rate bound are essential to achieve near-optimal quantization performance. To bridge this theoretical insight and practical implementation, we introduce *Q-Palette*, a versatile collection of fractional-bit quantizers that range from trellis-coded quantizers offering near-optimal distortion to simpler vector and scalar quantizers optimized for faster inference, all efficiently implemented with optimized CUDA kernels across various bitwidths. Furthermore, leveraging Q-Palette as a foundational component, we propose a novel mixed-scheme quantization framework, jointly optimizing quantizer choices and layer fusion decisions given resource constraints. The code is available at https://github.com/snu-mllab/Q-Palette.

1 Introduction

Large language models (LLMs) have recently achieved significant success across diverse tasks and are increasingly being deployed on resource-limited edge devices, such as laptops or smartphones [39, 9, 33]. However, these edge devices typically have limited memory resources and often process small-batch workloads, making inference severely memory-bound. Weight-only quantization has thus become essential, allowing models to achieve significantly greater compression at similar levels of performance compared to quantizing both weights and activations. Moreover, recent studies have demonstrated that weight-only quantization, beyond its well-known compression advantages, can also significantly accelerate inference speed in small-batch decoding scenarios by alleviating memory bottlenecks [28, 40, 20]. Specifically, we address weight-only post-training quantization (PTQ), enabling model quantization without costly retraining or extensive calibration data, which are common constraints in real-world deployments [5, 24].

However, quantizing LLM weights remains challenging due to inherently irregular, heavy-tailed distributions containing outliers that significantly broaden quantization ranges [23, 28, 29, 54]. To address this, recent research introduced a theoretically grounded approach known as *incoherence*

*Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

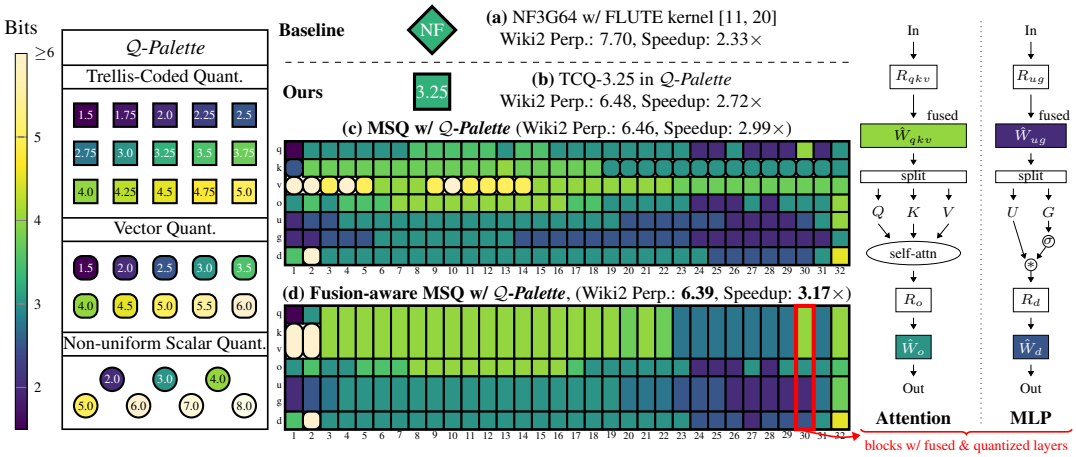


Figure 1: Qualitative comparison of quantization frameworks based on Q-Palette against the NormalFloat baseline with FLUTE kernels [11, 20], evaluated on the LLaMA 3.1-8B model using an RTX4090 GPU with a batch size of 1. Compared to (a) NormalFloat, (b) single-scheme quantization with TCQ-3.25 achieves a 17% inference speedup, (c) MSQ with Q-Palette provides a 28% speedup, and (d) fusion-aware MSQ further yields a 36% speedup alongside reduced WikiText2 perplexity, highlighting the practical effectiveness of Q-Palette and our MSQ framework. In the MSQ visualizations, columns represent transformer blocks, and rows represent linear layers, with colors indicating selected quantization bitwidths. The right visualization illustrates fused layers within the 30-th transformer block from configuration (d). Refer to Appendix F for the experimental details.

processing, which applies rotation matrices (*e.g.*, random Hadamard transforms) to weight matrices, reducing outliers by modifying distributions into approximately Gaussian forms [21, 6, 1, 49].

We begin with a natural question: if ideal Gaussian quantizers were available at arbitrary fractional bitwidths, how should we allocate bits across layers to minimize performance degradation under a fixed memory budget? Building upon the *linearity theorem* [35], which approximates performance degradation by quantization as a weighted sum of layer-wise mean squared errors, we derive an information-theoretically optimal bit allocation strategy for Gaussianized weights. Our analysis reveals that fine-grained fractional-bit quantizers that closely match their theoretical distortion bounds are essential to approaching the quantization performance predicted by theory. However, existing sophisticated Gaussian quantizers, such as trellis-coded quantization, have only been implemented with fused kernels for a limited set of integer bitwidths (*e.g.*, 2, 3, 4 bits), with little or no support for batch sizes larger than one [19, 15, 49, 50].

To bridge theoretical insights with practical quantization, we introduce *Q-Palette*, a versatile set of fractional-bit quantizers, ranging from trellis-coded quantizers (TCQ) for near-optimal distortion to simpler vector and scalar quantizers for low latency, covering diverse accuracy-latency trade-offs. We provide optimized CUDA kernels supporting a wide range of fractional bitwidths with broader batch size support than prior sophisticated quantization methods (*e.g.*, QTIP [50]). To enable even finer bitwidth control, we further propose half-TCQ, a novel TCQ variant that mixes two TCQ quantizers of different bitwidths within a single layer (*e.g.*, 2.5 and 3.0 bits) to realize intermediate bitwidths such as 2.75 bits, and extend our CUDA kernels to support this variant.

We integrate Q-Palette into a resource-constrained mixed-scheme quantization (MSQ) framework to demonstrate its practical utility. To further improve accuracy-latency trade-offs, we propose *fusion-aware MSQ*, the first MSQ approach that jointly optimizes quantizer selection and layer fusion, introducing a new optimization dimension (see Figure 1). Here, linear layers sharing the same input (*e.g.*, query, key, and value projections in a Transformer block) can be fused into a single linear layer, reducing memory accesses and kernel launches [26, 12, 40]. By incorporating layer fusion, fusion-aware MSQ achieves significant gains in accuracy-latency trade-offs. Extensive experiments on LLaMA 2, LLaMA 3, and Qwen models demonstrate that our MSQ framework with Q-Palette consistently outperforms strong data-free and data-aware weight-only PTQ baselines under both memory- and latency-constrained settings.

2 Preliminaries

2.1 Linearized surrogate objective for post-training quantization

Previous studies have approximated the performance degradation in neural networks induced by PTQ as a linear combination of layer-wise surrogate losses derived from second-order Hessian approximations [14, 7]. In the context of LLMs, where performance is typically measured by perplexity, the *linearity theorem* shows that the perplexity increase induced by quantization can be accurately approximated as a weighted sum of per-layer quantization errors [35]. This surrogate is especially valuable for data-free quantization scenarios where Hessian-based surrogate losses are unavailable. Formally, the linearized surrogate is expressed as:

$$\mathcal{L}(\{Q(W_l)\}_{l=1}^L) - \mathcal{L}(\{W_l\}_{l=1}^L) \approx \sum_{l=1}^{L} a_l \underbrace{\|Q(W_l) - W_l\|^2 / \|W_l\|^2}_{=:\,\mathrm{err}(Q;W_l)},$$

where $\mathcal{L}(\cdot)$ denotes the perplexity loss, $W_l \in \mathbb{R}^{d_l^{\text{in}} \times d_l^{\text{out}}}$ is the weight matrix of layer l, $Q(\cdot)$ is a quantization function, a_l is the empirically estimated sensitivity coefficient for layer l, and $\mathrm{err}(Q; W_l)$ is the normalized quantization error of layer l.

Leveraging this surrogate, the memory-constrained MSQ problem with a set of candidate quantizers $\mathcal{Q}$ can be formulated as a multiple-choice knapsack problem (MCKP):

$$\begin{aligned} \underset{P_{lq}\in\{0,1\}}{\text{minimize}} \quad & \sum_{l=1}^{L} a_l \left(\sum_{q=1}^{|\mathcal{Q}|} P_{lq} \cdot \mathrm{err}(Q_q; W_l) \right) \\ \text{subject to} \quad & \sum_{q=1}^{|\mathcal{Q}|} P_{lq} = 1, \quad \forall 1 \le l \le L, \\ & \sum_{l=1}^{L} \sum_{q=1}^{|\mathcal{Q}|} P_{lq} \cdot \mathrm{bit}(Q_q; W_l) d_l^{\text{in}} d_l^{\text{out}} \le M, \end{aligned} \tag{1}$$

where Q_q denotes a candidate quantizer, $\mathrm{bit}(Q_q; W_l)$ is the average number of bits per weight component for the weight matrix W_l quantized by Q_q, $P_{lq} \in \{0, 1\}$ is a binary indicator selecting quantizer Q_q for layer l, and M denotes the total memory budget (in bits) allocated for quantized model [46]. This formulation explicitly casts MSQ as a combinatorial optimization problem grounded in a linearized performance surrogate, providing a principled framework for optimal bit allocation under strict memory constraints [35, 7].

3 Q-Palette: fractional-bit quantizers

3.1 Motivation and design goals

Building on the theoretical foundation introduced in Section 2, we now derive the information-theoretically optimal bit allocation strategy and discuss its implications for the design of practical quantizers. Under the assumption that weight matrices are Gaussianized via incoherence processing, the quantization problem can be viewed as a Gaussian source coding problem. In this setting, classical rate-distortion theory establishes a fundamental lower bound on expected quantization error as $\mathbb{E}[\mathrm{err}(Q)] \ge 2^{-2\mathrm{bit}(Q)}$[10]. Assuming ideal Gaussian quantizers that achieve this bound at arbitrary fractional bitwidths $b_l \ge \eta$, the memory-constrained MSQ problem (1) simplifies to:

$$\begin{aligned} \underset{b_l \ge \eta}{\text{minimize}} \quad & \sum_{l=1}^{L} a_l 2^{-2b_l} \\ \text{subject to} \quad & \sum_{l=1}^{L} b_l d_l^{\text{in}} d_l^{\text{out}} \le M, \end{aligned} \tag{2}$$

where b_l is the fractional bitwidth allocated to layer l, and $\eta > 0$ is a minimum bitwidth threshold introduced to avoid degenerate cases such as assigning 0-bit to a layer. This formulation admits a closed-form solution as stated in Theorem 3.1.

Table 1: Quantizers, kernel implementations, and supported bitwidth intervals in Q-Palette.

Quantization scheme	Kernel implementations	Supported bitwidths (bits)
Non-uniform scalar quantization (NUQ)	Tensor Core, CUDA Core	2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0
Vector quantization (VQ)	Tensor Core, CUDA Core	1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0
Trellis-coded quantization (TCQ)	Tensor Core (TCQ) Tensor Core (Half-TCQ)	1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0 1.75, 2.25, 2.75, 3.25, 3.75, 4.25, 4.75

Theorem 3.1 (Optimal bit allocation with ideal Gaussian quantizers)**.** *If the budget M is feasible, i.e., $M \geq \eta \sum_{l=1}^{L} d_l^{in} d_l^{out}$, then the optimal fractional bit allocation $\{b_l^*\}$ for problem* (2) *is given by*

$$b_l^* = \max\left\{\eta, \frac{1}{2\ln(2)}\left(\ln\frac{a_l}{d_l^{\text{in}} d_l^{\text{out}}}\right) + C\right\}, \quad \forall\, 1 \leq l \leq L,$$

for the constant C that satisfies the memory constraint $\sum_l b_l^* d_l^{\text{in}} d_l^{\text{out}} = M$.

Proof. See Appendix A. □

In practice, however, quantization must be performed using a finite set of non-ideal quantizers $\mathcal{Q} = \{Q_1, \ldots, Q_N\}$, which introduces a gap between theoretical optimality and actual achievable performance. The extent of this gap depends primarily on two factors: (1) how closely each quantizer approaches the ideal distortion bound $2^{-2\text{bit}(Q)}$, and (2) how finely the available bitwidths can approximate the optimal fractional bit allocations b_l^*. Please refer to Appendix B for further analysis.

This motivates the need for practical quantizers that are both accurate and available at fine-grained fractional-bit intervals. Moreover, quantizers often exhibit a trade-off between distortion and computational efficiency: more sophisticated quantizers may offer lower error but incur higher inference costs. These considerations motivate the design of a practical quantization suite that supports fine-grained fractional bitwidths while also accounting for trade-offs between quantization error and computational efficiency. Guided by these goals, we design Q-Palette as a versatile collection of quantizers tailored to balance quantization error and computational efficiency across deployment scenarios.

3.2 Quantization schemes in Q-Palette

Q-Palette includes three quantizer families, non-uniform scalar quantization (NUQ), vector quantization (VQ), and trellis-coded quantization (TCQ), spanning a range of quantization error and inference latency trade-offs. In this section, we briefly introduce these quantization schemes.

Non-uniform scalar quantization. NUQ quantizes each scalar weight by assigning it to an entry in a non-uniformly spaced lookup table (LUT), in contrast to uniform scalar quantizers, which use equally spaced intervals [27]. We construct NUQ codebooks via k-means clustering on random Gaussian samples [34], thus optimizing the codebook entries for Gaussianized weights. NUQ incurs low dequantization overhead and enables efficient inference [20, 40].

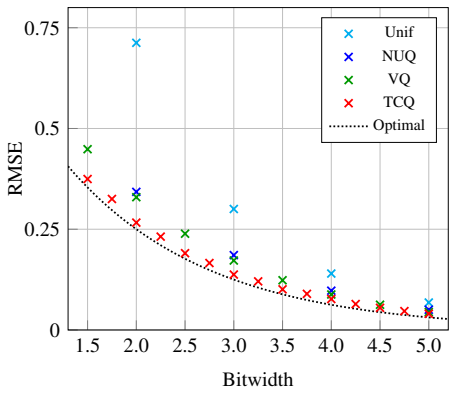


Figure 2: Gaussian quantization error of Q-Palette quantizers (NUQ, VQ, TCQ) compared to the uniform baseline.

Vector quantization. VQ partitions weight vectors into groups of fixed dimension, assigning each group to the nearest vector entry in a precomputed codebook [51]. In Q-Palette, we specifically implement 2D VQ, generating codebooks via k-means clustering on random 2D Gaussian samples. Efficient kernel implementations rely on LUTs whose sizes are powers of two, enabling compact bit-level representations. Consequently, fractional bitwidths at 0.5-bit intervals become natural candidates for efficient implementations. For instance, a $(2^3, 2)$-shaped LUT (eight 2D vectors) encodes two elements with 3 bits, effectively achieving 1.5 bits per scalar weight. Such constructions enable VQ to support fractional bitwidths at intervals of 0.5 bits.

Table 2: Decoding-latency speedup of quantized LLaMA 3.1-8B models relative to the FP16 baseline on an RTX 4090 GPU. 'TC' and 'CC' denote Tensor Core and CUDA Core kernels, respectively.

Decoding-latency speedup compared to FP16 (batch size = 1)										
Quantizer	2.0	2.25	2.5	2.75	3.0	3.25	3.5	3.75	4.0	4.25
NF w/ FLUTE [11, 20]	–	–	–	–	–	2.33×	–	–	–	2.20×
QTIP [50]	2.91×	–	–	–	2.49×	–	–	–	2.23×	–
Ours-NUQ-TC	3.64×	–	–	–	2.97×	–	–	–	2.57×	–
Ours-NUQ-CC	**3.70**×	–	–	–	**3.07**×	–	–	–	**2.61**×	–
Ours-VQ-TC	3.63×	–	3.24×	–	2.97×	–	2.69×	–	2.57×	–
Ours-VQ-CC	3.69×	–	**3.36**×	–	**3.07**×	–	**2.82**×	–	2.60×	–
Ours-TCQ-TC	3.57×	**3.23**×	3.13×	**2.95**×	2.96×	**2.72**×	2.70×	**2.61**×	2.59×	**2.37**×
Decoding latency speedup compared to FP16 (batch size = 8)										
Quantizer	2.0	2.25	2.5	2.75	3.0	3.25	3.5	3.75	4.0	4.25
NF w/ FLUTE [11, 20]	–	–	–	–	–	2.28×	–	–	–	2.13×
QTIP [50]	0.74×	–	–	–	0.55×	–	–	–	0.47×	–
Ours-NUQ-TC	**3.28**×	–	–	–	**2.78**×	–	–	–	2.46×	–
Ours-NUQ-CC	2.80×	–	–	–	2.56×	–	–	–	2.11×	
Ours-VQ-TC	**3.28**×	–	**2.87**×	–	2.74×	–	2.34×	–	**2.47**×	–
Ours-VQ-CC	2.94×	–	2.72×	–	2.66×	–	2.46×	–	2.37×	–
Ours-TCQ-TC	3.16×	**2.92**×	2.80×	**2.75**×	2.74×	**2.56**×	**2.47**×	**2.49**×	2.46×	**2.27**×

Trellis-coded quantization. TCQ is known as a sophisticated Gaussian source quantizer, achieving near-optimal distortion performance [19]. Recently, QTIP introduced optimized CUDA kernels for the bitshift variant of TCQ [36, 50], which encodes each high-dimensional real-valued vector $\mathbf{v}$ into a binary bitshift representation $\mathbf{r}$:

$$\hat{\mathbf{v}}[i \cdot V : (i+1) \cdot V] = \mathrm{LUT}\left(\mathbf{r}[i \cdot s : i \cdot s + L]\right),$$

where each V-dimensional subvector is represented by a sliding bit-window of length L, shifted by s bits at each step. This encoding achieves an effective fractional bitwidth of s/V. While previous TCQ kernel (QTIP) supported integer bitwidths (*e.g.*, 2, 3, 4 bits with shifts $s = 4, 6, 8$ for $V = 2$) and were limited to single batch, we significantly expand practical applicability by introducing fractional bitwidth support (*e.g.*, 1.5, 2.5, 3.5, 4.5, 5.0 bits corresponding to shifts $s = 3, 5, 7, 9, 10$ for $V = 2$) and optimized kernels supporting inference at batch sizes up to 8. For constructing the LUT, we follow the protocol of QTIP, which is also based on k-means clustering of Gaussian samples.

Furthermore, we introduce *half-TCQ*, a simple extension enabling quantization at intermediate fractional bitwidth intervals. Specifically, given a weight $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, half-TCQ partitions the matrix row-wise and applies different bitwidth quantization to each partition. For example, to achieve 2.75 bit quantization, half-TCQ quantizes $W[: d_{\text{in}}/2]$ at 2.5 bits and the remaining half $W[d_{\text{in}}/2 :]$ at 3 bits. To preserve computational efficiency, we implement a dedicated CUDA kernel that performs fused dequantization and matrix multiplication for half-TCQ in a single kernel call. As illustrated in Figure 2, TCQ-based schemes, including half-TCQ, consistently achieve quantization error close to theoretical lower bounds, outperforming simpler quantizers. For a more detailed explanation of these quantizers, including quantization algorithms, please refer to Appendix C.

3.3 Implementation details for efficiency

Reducing rotation overhead. Incoherence processing rotates weights along both input and output dimensions ($W \rightarrow RWR'$) with per-tensor scaling, yielding approximately standard Gaussian distributions. However, each rotation also requires rotating activations online during inference (*e.g.*, $X \rightarrow XR$), incurring significant computational overhead [6, 49, 50]. We reduce this overhead by rotating weights only along the input dimension ($W \rightarrow RW$) and applying per-output-channel scaling, normalizing each rotated column to an approximately standard Gaussian distribution. Additionally, we share rotation matrices among linear layers with identical inputs (*e.g.*, query/key/value or gate/up projections in Transformer blocks). Combining these techniques reduces the number of online rotations per Transformer block from 14 to 4.

Kernel implementation. We implement two types of CUDA kernels optimized for different inference scenarios: (i) Tensor Core-based kernels and (ii) CUDA Core-based kernels. Our Tensor Core-based kernels, supporting TCQ, NUQ, and VQ, extend the single batch implementation from QTIP,

which leverages warp-level `mma` (matrix-multiply-accumulate) instructions [38]. Integrating efficient dequantization logic into this framework involves non-trivial engineering efforts, including the precise mapping of quantized weights to `mma` instruction fragments. Overall, our implementation extends kernel support from integer bitwidths (2, 3, 4 bits) to fractional bitwidths (1.5, 2.5, 3.5, 4.5, 5.0 bits). To further minimize overhead at larger batch sizes, we traverse each quantized weight exactly once, directly loading and performing register-level dequantization without intermediate storage. Input activations are cached in shared memory for efficient reuse across multiple weight multiplications, significantly improving inference efficiency.

Our CUDA Core-based kernels, supporting NUQ and VQ, extend the Any-Precision LLM kernels [40], originally designed for NUQ. Specifically, we replace bit-plane encoding with simpler bit-packing encoding to simplify the dequantization process. Additionally, we incorporate Any-precision's table lookup merging technique into both Tensor Core and CUDA Core-based NUQ kernels for bitwidths 2, 3, and 4, further reducing dequantization overhead. Table 1 summarizes the supported kernel implementations and bitwidths for quantization schemes in Q-Palette. For transparency and reproducibility, the full implementations of both kernel types are included in our code release, and additional kernel analysis is provided in Appendix D.

Table 2 summarizes the end-to-end decoding-latency speedup for LLaMA 3.1 8B models quantized at various bitwidths (2.0-4.25 bits) using quantizers in Q-Palette, compared against two baselines: NormalFloat with FLUTE kernels and QTIP [11, 20, 50]. Our quantizers consistently deliver superior inference speed while supporting a wide range of finer-grained fractional bit quantization. For smaller batch sizes (batch size $= 1$), CUDA Core-based kernels typically outperform Tensor Core-based kernels, whereas Tensor Core-based kernels often provide better latency at larger batch sizes (batch size $= 8$). Although TCQ incurs slightly higher latency overhead compared to NUQ and VQ, our TCQ quantizers still achieve significantly faster decoding speed compared to baseline quantizers, clearly demonstrating practical efficiency improvements for real-world inference workloads. Note that our TCQ quantizers extend QTIP and, at batch size 1, primarily differs by supporting a wider range of bitwidths and reducing the number of online Hadamard transforms, thereby lowering rotation cost and improving decoding-latency speedup, *e.g.*, from 2.91× to 3.57× at 2-bit.

4 Mixed-scheme quantization with Q-Palette

4.1 Mixed-scheme quantization under resource constraints

The memory-constrained MSQ formulation introduced in Section 2 naturally generalizes to broader resource constraints such as inference latency. Following prior one-shot mixed-precision quantization frameworks [14, 7, 35], we formulate resource-constrained MSQ as MCKP:

$$\begin{aligned} \underset{P_{lq}\in\{0,1\}}{\text{minimize}} \quad & \sum_{l=1}^{L}\sum_{q=1}^{|\mathcal{Q}|} P_{lq}\cdot \ell_{lq} \\ \text{subject to} \quad & \sum_{q=1}^{|\mathcal{Q}|} P_{lq} = 1, \quad \forall 1 \le l \le L, \\ & \sum_{l=1}^{L}\sum_{q=1}^{|\mathcal{Q}|} P_{lq}\cdot c_{lq} \le C, \end{aligned} \tag{3}$$

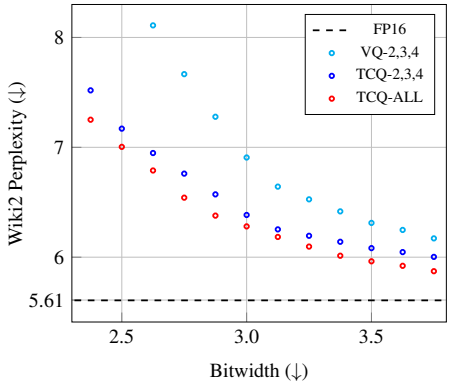


Figure 3: Performance comparison of memory-constrained MSQ for different quantizer sets in Q-Palette on LLaMA 3.1-8B.

where the loss term ℓ_{lq} denotes the estimated loss in performance incurred by selecting quantizer Q_q for layer l, the cost term $c_{lq} \triangleq \text{cost}(Q_q; W_l)$ represents the profiled resource cost (*e.g.*, memory or latency), and the total resource constraint is denoted by C. The loss term ℓ_{lq} can be instantiated in various ways depending on available information. For example, in data-free settings, we approximate the loss term as $\ell_{lq} = a_l \cdot \text{err}(Q_q; W_l)$, as in Equation (1), where a_l is a layer-wise sensitivity coefficient computed following the protocol of HIGGS [35]. We estimate $\text{err}(Q_q; W_l)$ using the precomputed distortion of Q_q, obtained by quantizing a random Gaussian matrix. This enables fast estimation of loss terms in data-free scenarios without fully realizing the quantization pipeline.

While a dynamic programming algorithm exists for solving MCKP [43], we simply use the SCIP solver provided by Google OR-Tools for flexibility and ease of implementation [42]. Please refer to Appendix E for additional details on loss term computation and cost profiling.

We illustrate the effectiveness of Q-Palette in Figure 3 by comparing memory-constrained MSQ results across different quantizer subsets within Q-Palette. Specifically, *TCQ-ALL* includes all TCQ quantizers available in Q-Palette, while *TCQ-2,3,4* and *VQ-2,3,4* reflect integer-bitwidth quantizers supported in QTIP and HIGGS, respectively [50, 35]. Although QTIP is originally a single-scheme baseline, our constructed integer-bitwidth subset *TCQ-2,3,4* serves as a reasonable reference to evaluate the benefit of broadened quantizer support. The results clearly highlight the advantage of *TCQ-ALL*, demonstrating that the broadened quantizer support provided by Q-Palette consistently yields superior performance.

4.2 Fusion-aware mixed-scheme quantization

Layer fusion is a widely used optimization for accelerating inference speed of DNN models [26, 12]. Within each Transformer block, certain linear layers, such as {query, key, value} projections or {up, gate} projections, share the same input and can be fused into a single linear layer. For example, instead of separately computing XW_q, XW_k, and XW_v, we can concatenate the weight matrices and compute $X(W_q \oplus W_k \oplus W_v)$, followed by splitting the output (see the right side of Figure 1). Layer fusion can reduce the number of kernel launches and memory accesses, thereby providing further opportunities for inference speedup.

We propose *fusion-aware MSQ*, a novel MSQ framework that jointly optimizes quantization with the additional design dimension of layer fusion. Fusion-aware MSQ simultaneously determines 1) how to group layers for fusion and 2) which quantizer to assign to each fused group. Whereas standard MSQ (Equation (3)) introduces one binary decision variable per (layer, quantizer) pair, fusion-aware MSQ instead defines one binary variable per (fusible layer group, quantizer) pair. Here, a fusible layer group is a set of layers sharing the same input. For generic Transformer models, we can write the set of all fusible layer groups for each Transformer block b as

$$\mathcal{G}_b = \big\{\{q_b\}, \{k_b\}, \{v_b\}, \{q_b, k_b\}, \{q_b, v_b\}, \{k_b, v_b\}, \{q_b, k_b, v_b\}, \{o_b\}, \{u_b\}, \{g_b\}, \{u_b, g_b\}, \{d_b\}\big\}.$$

The overall set of fusible layer groups is $\mathcal{G} = \bigcup_{b=1}^{B} \mathcal{G}_b$ where B is the number of Transformer blocks. For each $g \in \mathcal{G}$ and quantizer $Q_q \in \mathcal{Q}$, we introduce a binary variable $P_{gq} \in \{0, 1\}$ indicating that all layers in g are fused and quantized by Q_q.

The fusion-aware MSQ problem is formulated as:

$$\underset{P_{gq} \in \{0,1\}}{\text{minimize}} \quad \sum_{g \in \mathcal{G}} \sum_{q=1}^{|\mathcal{Q}|} P_{gq} \cdot \sum_{l \in g} \ell_{lq} \tag{4}$$

$$\text{subject to} \quad \sum_{g \in \mathcal{G}: l \in g} \sum_{q=1}^{|\mathcal{Q}|} P_{gq} = 1, \quad \forall l \in \bigcup_{b=1}^{B} \{q_b, k_b, v_b, o_b, u_b, g_b, d_b\}, \tag{C1}$$

$$\sum_{g \in \mathcal{G}} \sum_{q=1}^{|\mathcal{Q}|} P_{gq} \cdot c_{gq} \leq C, \tag{C2}$$

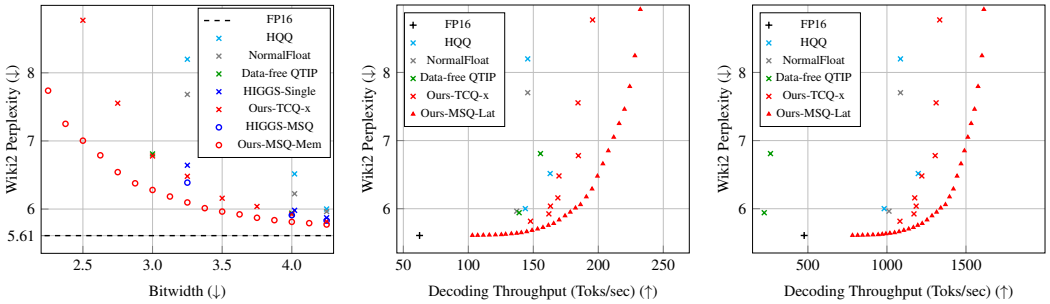
where ℓ_{lq} is the loss term, c_{gq} represents the profiled cost (*e.g.*, latency) of the fused layer corresponding to the group g quantized by Q_q.

To ensure valid solutions, two constraints are imposed. *Exclusive assignment* (C1): every layer must belong to exactly one active (group, quantizer) pair; among all fusible groups g that contain a given layer l, only one associated variable P_{gq} can be 1. *Resource constraint* (C2): the total profiled cost of all activated groups must not exceed the resource budget C.

This formulation explicitly captures latency improvements enabled by fusion, thus providing improved accuracy-latency trade-offs compared to the MSQ formulation that neglects layer fusion (see Figure 1). Since both the objective and constraints are linear in P_{gq}, Equation (4) is also an ILP. We solve this ILP using the SCIP solver in OR-Tools [42]. Note that, compared to non-fusion-aware MSQ (Equation (3)), fusion-aware MSQ introduces $1.71\times$ more decision variables while maintaining the same number of constraints.

Table 3: Data-free quantization results on LLaMA 3 models for various bitwidths.

Method	LLaMA 3.1-8B			LLaMA 3.2-1B			LLaMA 3.2-3B		
	Bits (↓)	Wiki2 (↓)	Acc (↑)	Bits (↓)	Wiki2 (↓)	Acc (↑)	Bits (↓)	Wiki2 (↓)	Acc (↑)
FP16	16.00	5.61	69.3	16.00	8.64	55.9	16.00	6.98	63.7
Data-free QTIP	3.00	6.81	**66.9**	3.00	13.35	49.0	3.00	8.89	58.2
Ours-TCQ-3	3.00	**6.78**	66.0	3.00	**12.59**	**50.7**	3.00	**8.67**	**60.3**
Ours-MSQ-Mem	3.00	**6.28**	**67.5**	3.00	**10.51**	**53.2**	3.00	**7.81**	**61.7**
NF	3.25	7.70	64.3	3.25	17.73	46.8	3.25	10.06	59.3
HQQ	3.25	8.29	63.2	3.25	26.42	42.9	3.25	11.68	54.2
HIGGS	3.25	6.64	66.4	3.25	12.19	51.1	3.25	8.67	60.1
Ours-TCQ-3.25	3.25	**6.48**	**66.4**	3.25	**11.30**	**51.8**	3.25	**8.15**	**61.0**
HIGGS-MSQ	3.25	6.39	66.7	3.25	11.08	52.5	3.25	8.01	61.1
Ours-MSQ-Mem	3.25	**6.10**	**67.6**	3.25	**10.00**	**53.7**	3.25	**7.60**	**61.9**
NF	4.02	6.22	67.8	4.02	10.70	52.9	4.02	7.82	62.1
HQQ	4.02	6.52	67.5	4.02	13.47	51.4	4.02	8.67	60.2
HIGGS	4.02	5.98	**68.7**	4.02	9.64	53.6	4.02	7.46	62.3
Data-free QTIP	4.00	5.94	68.4	4.00	9.53	**54.9**	4.00	7.41	62.8
Ours-TCQ-4	4.00	**5.92**	68.2	4.00	**9.45**	54.3	4.00	**7.37**	**63.4**
HIGGS-MSQ	4.00	5.91	68.3	4.00	9.52	55.0	4.00	7.40	62.2
Ours-MSQ-Mem	4.00	**5.81**	**69.0**	4.00	**9.14**	**55.2**	4.00	**7.22**	**63.2**



(a) Memory (b) Throughput (b.s.= 1) (c) Throughput (b.s.= 8)

Figure 4: Performance trade-offs of quantized LLaMA 3.1-8B models under different constraints in the data-free setting on an RTX 4090 GPU: (a) memory constraint; (b) latency constraint (single batch); (c) throughput evaluation (batch size $= 8$) of the quantized models in (b).

5 Experiments

We evaluate the quantization performance of our methods against baselines on the LLaMA 3 series (LLaMA 3.1-8B, 70B, 3.2-1B, 3B), LLaMA 2 series (LLaMA 2-7B, 13B), and Qwen 2.5-7B [18, 48, 55]. For the data-free scenario, we consider single-scheme quantization baselines—HQQ (uniform), NormalFloat (NUQ), HIGGS-Single (VQ), and data-free QTIP (TCQ)—as well as the MSQ baseline HIGGS-MSQ [2, 11, 50, 35]. For the data-aware scenario, we use QTIP as the baseline. For all experiments, both our methods and baselines are evaluated strictly in the PTQ setting, without any retraining. Performance is measured primarily via WikiText2 perplexity and average zero-shot accuracy across ARC-easy, ARC-challenge, HellaSwag, PiQA, and WinoGrande [37, 8, 56, 3, 44]. For latency evaluations, we use Gemlite kernels for HQQ at higher bitwidths (4.02, 4.25 bits), and FLUTE kernels for NormalFloat (3.25, 4.25 bits) and HQQ at 3.25 bits [22, 20]. Because QTIP supports only single batch inference, we simulate larger batch sizes by repeated kernel invocation.

We denote *Ours-TCQ-x* as single-scheme quantization using TCQ-x from Q-Palette, *Ours-MSQ-Mem* as memory-constrained MSQ (Section 4.1) using Q-Palette's TCQ quantizers, and *Ours-MSQ-Lat* as latency-constrained fusion-aware MSQ (Section 4.2) using all Q-Palette quantizers with Tensor-Core kernels. Please refer to Appendix G for additional results on other models, ablation studies, and detailed experimental settings.

Table 4: Data-aware quantization results on LLaMA 2 models (throughput on an RTX4090 GPU).

	LLaMA 2 7B					LLaMA 2 13B				
				Throughput (Toks/s)					Throughput (Toks/s)	
Method	Bits	Wiki2 (↓)	Acc (↑)	$B=1$	$B=8$	Bits	Wiki2 (↓)	Acc (↑)	$B=1$	$B=8$
FP16	16.00	5.12	64.9	71	527	16.00	4.57	67.9	OOM	OOM
QTIP	2.00	6.84	58.9	209	386	2.00	5.62	63.6	131	154
Ours-MSQ-Mem	2.00	**6.47**	**60.3**	**272**	**1684**	2.00	**5.35**	**64.2**	**152**	**928**
QTIP	3.00	5.39	63.3	184	304	3.00	4.76	**67.0**	110	153
Ours-MSQ-Mem	3.00	**5.34**	**63.9**	**224**	**1489**	3.00	**4.74**	**67.0**	**126**	**738**

5.1 Data-free quantization results

Table 3 summarizes data-free quantization performance on LLaMA 3 models. *Ours-TCQ-x* consistently outperforms all single-scheme baselines in WikiText2 perplexity and achieves competitive zero-shot accuracy. Notably, *Ours-MSQ-Mem* surpasses all baseline methods, clearly demonstrating the effectiveness of Q-Palette. Figure 4a evaluates *Ours-MSQ-Mem* across a broader memory range (2.25-4.25 bits). Our method achieves Pareto-dominant performance, significantly outperforming baseline methods. Remarkably, our 2.875-bit model achieves comparable WikiText2 perplexity to the 3.25-bit HIGGS-MSQ model, resulting in a $1.13\times$ higher compression ratio and superior perplexity. Figures 4b and 4c compare the throughput-perplexity trade-offs of *Ours-MSQ-Lat* and *Ours-TCQ-x* against baseline methods. Both variants achieve significant throughput improvements over baseline methods, substantially expanding the Pareto frontier in both batch sizes 1 and 8.

5.2 Data-aware quantization results

We further evaluate our methods in the data-aware setting by comparing our *Ours-MSQ-Mem* approach against the state-of-the-art QTIP baseline (without retraining) on the LLaMA 2-7B and 13B models. For this setting, we utilize the same proxy Hessian used in QTIP during the quantization and compute the loss term ℓ_{lq} for our MSQ as the actual validation perplexity degradation induced by quantizing the weight W_l using the quantizer Q_q. As summarized in Table 4, our method consistently achieves superior perplexity and zero-shot accuracy compared to the baseline. Additionally, our optimized kernels achieve over $4\times$ throughput improvements at batch size 8 for both LLaMA 2 models at both 2 and 3 bits, demonstrating the practical benefits of our optimized kernel for batch size 8.

6 Related works

Incoherence processing. Previous methods for handling outliers in LLM quantization have primarily relied on heuristic techniques [28, 54, 29]. Recently, a theoretically grounded approach, incoherence processing, has been introduced to systematically address weight irregularities [6]. Incoherence processing applies rotation matrices to weight matrices prior to quantization, significantly suppressing outliers and transforming distributions into approximately Gaussian forms [6, 1, 45]. This Gaussianization enables the use of sophisticated Gaussian quantizers such as lattice vector quantization [49] and trellis-coded quantization [50]. However, current implementations support efficient kernels only for limited integer bitwidths and small batch sizes, constraining their practicality, a limitation that our proposed Q-Palette directly addresses by introducing fractional-bit quantizers and optimized CUDA kernels with broader batch size support.

More recent rotation-based approaches further enhance quantization performance by applying learned matrix transforms such as scaling or affine transformations [32, 30, 23, 47]. However, these methods mainly target weight-activation quantization and require calibration data to learn the transforms, whereas our work focuses on weight-only PTQ, which remains applicable even in data-free settings and is particularly suited for memory-bound, small-batch inference.

Other weight-only post-training quantization methods. Several simpler PTQ methods prioritize computational and implementation efficiency. HQQ employs data-free uniform quantization via half-quadratic optimization [2]. NormalFloat constructs lookup tables for non-uniform scalar quantization using Gaussian quantiles [11]. FLUTE offers state-of-the-art CUDA kernels for LUT-based non-

uniform quantizers with per-group scaling [20]. Despite their efficiency, these approaches generally incur higher quantization errors compared to sophisticated quantizers such as TCQ.

Mixed-precision and mixed-scheme quantization. Mixed-precision quantization (MPQ) optimizes layer-wise bit allocation under given constraints [53]. For vision models, HAQ and HAWQ-V2 introduced surrogate objectives based on second-order information for MPQ [13, 14]. Chen et al. generalized these approaches by explicitly incorporating diverse resource constraints, such as latency, and formulated the problem as an MCKP [7]. Recently, HIGGS introduced the linearity theorem, a data-free linear surrogate specifically tailored for LLM quantization [35]. Building upon these works and drawing insights from compiler optimization research [26, 12], we propose a novel fusion-aware mixed-scheme quantization framework that jointly optimizes quantizer selection and layer fusion decisions, achieving superior accuracy-latency trade-offs.

7 Conclusion

In this paper, we have investigated weight-only PTQ as a solution for compressing LLMs, particularly beneficial for memory-bound inference tasks with small batch sizes. Considering that irregular weight distributions in LLMs have complicated quantization, we leveraged recent rotation-based methods that Gaussianize weight distributions, enabling a theoretical analysis of optimal bit allocation. Based on this perspective, we derived an information-theoretically optimal bit allocation strategy under fixed bit budgets, demonstrating that fine-grained fractional-bit quantizers closely approaching the Gaussian distortion-rate bound are essential for achieving near-optimal quantization efficiency. To translate this theoretical finding into practical benefits, we introduced Q-Palette, a versatile suite of fractional-bit quantizers, from sophisticated trellis-coded quantization schemes offering near-optimal distortion to simpler vector and scalar quantizers optimized for fast inference, each efficiently implemented with optimized CUDA kernels across a wide range of bitwidths. We further integrated Q-Palette into an MSQ framework, proposing a novel fusion-aware MSQ approach that jointly optimizes quantizer selection and layer fusion decisions under given resource constraints, effectively improving inference latency. Experimental evaluations validated that our MSQ framework with Q-Palette and fusion-aware optimization consistently outperforms existing baseline methods, achieving superior accuracy-memory and accuracy-latency trade-offs on LLaMA 2 and LLaMA 3 models.

Impact statement

Q-Palette introduces a versatile set of quantizers with broad fractional-bitwidth support, which can serve as a foundational building block for evaluating and developing MSQ algorithms. Q-Palette's quantizers are usable in data-free scenarios, offering off-the-shelf applicability like NormalFloat and HQQ [11, 20, 2], which lowers the barrier for practitioners lacking calibration data. Importantly, Q-Palette supports a wide spectrum of performance-efficiency trade-offs, enabling practitioners to select quantization configurations that best match their specific deployment workloads. This adaptability is valuable for real-world applications where resource constraints and performance requirements vary significantly. Moreover, our results challenge the misconception that sophisticated quantizers such as TCQ are computationally prohibitive for practical use beyond batch size 1 [35]. We demonstrate that TCQ achieves efficient decoding speeds for batch sizes up to 8, making it practically suitable for edge-device workloads. By correcting this misunderstanding, our work may encourage further investigation into TCQ and other quantizers previously considered computationally expensive.

Acknowledgements and Disclosure of Funding

We would like to thank Jinuk Kim for insightful discussions and helpful feedback on this work. This work was supported by Samsung Electronics Co., Ltd. (IO250418-12669-01), Mobile eXperience (MX) Business, Samsung Electronics Co., Ltd., Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [No. RS-2020-II200882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning, No. RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)], and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00354036). Hyun Oh Song is the corresponding author.

References

[1] 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.

[2] Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, 2023. URL `https://mobiusml.github.io/hqq_blog/`.

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

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *CVPR*, 2020.

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

[7] Weihan Chen, Peisong Wang, and Jian Cheng. Towards mixed-precision quantization of neural networks via constrained optimization. In *ICCV*, 2021.

[8] 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:1803.05457v1*, 2018.

[9] LMDeploy Contributors. Lmdeploy: A toolkit for compressing, deploying, and serving llm. `https://github.com/InternLM/lmdeploy`, 2023.

[10] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition*. Wiley-Interscience, 2006.

[11] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *NeurIPS*, 2023.

[12] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. Ios: Inter-operator scheduler for cnn acceleration. In *MLSys*, 2021.

[13] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *ICCV*, 2019.

[14] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. In *NeurIPS*, 2020.

[15] T.R. Fischer, M.W. Marcellin, and M. Wang. Trellis-coded vector quantization. *IEEE Transactions on Information Theory*, 1991.

[16] G.D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 1973.

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

[18] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and et al. The llama 3 herd of models. In *arXiv:2407.21783*, 2024.

[19] R.M. Gray and D.L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 1998.

[20] Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P. Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized llms. In *EMNLP findings*, 2024.

[21] A. Hedayat and W. D. Wallis. Hadamard Matrices and Their Applications. *The Annals of Statistics*, 1978.

[22] Appu Shaji Hicham Badri. Gemlite: Towards building custom low-bit fused cuda kernels, 2024. URL https://mobiusml.github.io/gemlite_blog/.

[23] Xing Hu, Yuan Cheng, Dawei Yang, Zukang Xu, Zhihang Yuan, Jiangyong Yu, Chen Xu, Zhe Jiang, and Sifan Zhou. Ostquant: Refining large language model quantization with orthogonal and scaling transformations for better distribution fitting. In *ICLR*, 2025.

[24] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Accurate post training quantization with small calibration sets. In *ICML*, 2021.

[25] Jake Hyun. Log-time k-means clustering for 1d data: Novel approaches with proof and implementation. In *arXiv:2412.15295*, 2024.

[26] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *SOSP*, 2019.

[27] 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. In *ICML*, 2025.

[28] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In *ICML*, 2024.

[29] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *AAAI*, 2024.

[30] 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.

[31] Yujun Lin*, Haotian Tang*, Shang Yang*, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. In *MLSys*, 2025.

[32] 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 *The Thirteenth International Conference on Learning Representations*, 2025.

[33] llama.cpp Contributors. llama.cpp. https://github.com/ggml-org/llama.cpp, 2023.

[34] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 1982.

[35] Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. Higgs: Pushing the limits of large language model quantization via the linearity theorem. In *ACL*, 2025.

[36] Mark Mao and Robert Gray. Stationary and trellis encoding for iid sources and simulation. *Data Compression Conference Proceedings*, 2010.

[37] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *arXiv:1609.07843*, 2016.

[38] NVIDIA Corporation. Nvidia tensor cores. https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/, 2018.

[39] OpenAI. Gpt-4 technical report. In *arXiv.2303.08774*, 2023.

[40] 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.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[42] Laurent Perron and Vincent Furnon. Or-tools. URL `https://developers.google.com/optimization/`.

[43] Ulrich Pferschy and Rosario Scatamacchia. Improved dynamic programming and approximation results for the knapsack problem with setups: ıu. pferschy and r. scatamacchia. *International Transactions in Operational Research*, 2017.

[44] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *arXiv:1907.10641*, 2019.

[45] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *NeurIPS*, 2024.

[46] Prabhakant Sinha and Andris A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 1979.

[47] Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, et al. Flatquant: Flatness matters for llm quantization. In *ICML*, 2025.

[48] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, and et al. Llama 2: Open foundation and fine-tuned chat models. In *arXiv:2307.09288*, 2023.

[49] 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.

[50] Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. Qtip: Quantization with trellises and incoherence processing. In *NeurIPS*, 2024.

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

[52] Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

[53] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. In *arXiv:1812.00090*, 2018.

[54] 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.

[55] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, and et al. Qwen2.5 technical report. In *arXiv:2412.15115*, 2025.

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

A Optimal bitwidth proof

In this section, we formally derive the optimal bit allocation result stated in the main paper. Under the assumption that weight matrices have been Gaussianized through incoherence processing, the quantization problem can be viewed as a Gaussian source coding problem. We recall the memory-constrained mixed-scheme quantization (MSQ) formulation as:

$$\begin{aligned} \underset{P_{lq}\in\{0,1\}}{\text{minimize}} \quad & \sum_{l=1}^{L} a_l \left(\sum_{q=1}^{|\mathcal{Q}|} P_{lq} \cdot \mathrm{err}(Q_q; W_l) \right) \\ \text{subject to} \quad & \sum_{q=1}^{|\mathcal{Q}|} P_{lq} = 1, \quad \forall 1 \le l \le L, \\ & \sum_{l=1}^{L} \sum_{q=1}^{|\mathcal{Q}|} P_{lq} \cdot \mathrm{bit}(Q_q; W_l) d_l^{\text{in}} d_l^{\text{out}} \le M, \end{aligned} \tag{1}$$

where a_l is the empirically estimated sensitivity coefficient for layer l, $\mathrm{err}(Q; W_l)$ is the normalized quantization error of layer l, Q_q denotes a candidate quantizer, $\mathrm{bit}(Q_q; W_l)$ is the average number of bits per weight component for the weight matrix W_l quantized by Q_q, $P_{lq} \in \{0, 1\}$ is a binary indicator selecting quantizer Q_q for layer l, and M denotes the total memory budget (in bits) allocated for quantized model [46, 35, 7].

We recall that classical rate-distortion theory provides a fundamental lower bound on the expected quantization error for Gaussian sources: $\mathbb{E}[\mathrm{err}(Q)] \ge 2^{-2\mathrm{bit}(Q)}$ [10]. Further assuming we have access to ideal Gaussian quantizers capable of exactly achieving this theoretical distortion bound $\mathbb{E}[\mathrm{err}(Q)] = 2^{-2\mathrm{bit}(Q)}$ at any fractional bitwidth $b_l \ge \eta$, the memory-constrained MSQ (problem (1)) can again be written as the continuous optimization problem:

$$\begin{aligned} \underset{b_l\ge\eta}{\text{minimize}} \quad & \sum_{l=1}^{L} a_l 2^{-2b_l} \\ \text{subject to} \quad & \sum_{l=1}^{L} b_l d_l^{\text{in}} d_l^{\text{out}} \le M, \end{aligned} \tag{2}$$

where b_l is the fractional bitwidth allocated to layer l, and $\eta > 0$ is a minimum bitwidth threshold introduced to avoid degenerate cases such as assigning 0-bit to a layer. Here, we replace the actual quantization error $\mathrm{err}(Q)$ with its expectation $\mathbb{E}[\mathrm{err}(Q)]$. We empirically justify this replacement by demonstrating extremely low variance in quantization errors for typical weight matrix dimensions encountered in LLMs (see Table 5). Additionally, we assume that the sensitivity coefficients a_l are non-negative ($a_l \ge 0$), a reasonable assumption given that pretrained weights typically represent local optima. Given this simplified optimization problem, we now derive the closed-form solution for the optimal fractional bit allocation.

Theorem 3.1 (Optimal bit allocation with ideal Gaussian quantizers)**.** *If the budget M is feasible, i.e., $M \ge \eta \sum_{l=1}^{L} d_l^{in} d_l^{out}$, then the optimal fractional bit allocation $\{b_l^*\}$ for problem* (2) *is given by*

$$b_l^* = \max\left\{ \eta, \frac{1}{2\ln(2)} \left(\ln \frac{a_l}{d_l^{\text{in}} d_l^{\text{out}}} \right) + C \right\}, \quad \forall\, 1 \le l \le L,$$

for the constant C that satisfies the memory constraint $\sum_l b_l^* d_l^{\text{in}} d_l^{\text{out}} = M$.

Proof. Let's start by formulating the Lagrangian function for problem (2), explicitly including the constraint $b_l \ge \eta$ via Lagrange multipliers $\mu_l \ge 0$ and the budget constraint via $\lambda \ge 0$:

$$\mathcal{L}(\{b_l\}, \lambda, \{\mu_l\}) \coloneqq \sum_{l=1}^{L} a_l 2^{-2b_l} + \lambda \left(\sum_{l=1}^{L} b_l d_l^{\text{in}} d_l^{\text{out}} - M \right) - \sum_{l=1}^{L} \mu_l (b_l - \eta).$$

Table 5: Empirical distortion statistics for quantizing random Gaussian matrices (4096×4096) with Q-Palette quantizers over 32 trials.

Quantizer	Mean distortion	Std. deviation
Ours-TCQ-2	0.07101	8.36E-06
Ours-NUQ-2	0.11747	4.24E-05
Ours-VQ-2	0.10857	2.93E-05

By differentiating the Lagrangian with respect to b_l and setting it equal to zero to find the stationary points, we have:

$$\frac{\partial \mathcal{L}}{\partial b_l} = -2\ln(2)\, a_l 2^{-2b_l} + \lambda d_l^{\text{in}} d_l^{\text{out}} - \mu_l = 0.$$

Since, for all l, $\mu_l \geq 0$ and complementary slackness requires $\mu_l(b_l^* - \eta) = 0$ [4], we have two cases:
Case 1: If $b_l^* > \eta$, complementary slackness implies $\mu_l = 0$, and thus:

$$2^{-2b_l^*} = \frac{\lambda d_l^{\text{in}} d_l^{\text{out}}}{2\ln(2)\, a_l}.$$

Taking logarithms on both sides and rearranging terms explicitly, we obtain:

$$b_l^* = \frac{1}{2\ln(2)} \left(\ln \frac{a_l}{d_l^{\text{in}} d_l^{\text{out}}} \right) + \frac{1}{2\ln(2)} \left(\ln(2\ln(2)) - \ln(\lambda) \right) = \frac{1}{2\ln(2)} \left(\ln \frac{a_l}{d_l^{\text{in}} d_l^{\text{out}}} \right) + C(\lambda).$$

where, the constant term $C(\lambda)$ is defined as $C(\lambda) \coloneqq \frac{1}{2\ln(2)}(\ln(2\ln(2)) - \ln(\lambda))$.

Case 2: If $b_l^* = \eta$, we directly have:

$$b_l^* = \eta.$$

Combining these cases yields the optimal fractional bit allocation:

$$b_l^* = \max\left\{ \eta, \frac{1}{2\ln(2)} \left(\ln \frac{a_l}{d_l^{\text{in}} d_l^{\text{out}}} \right) + C(\lambda) \right\},$$

where the constant $C(\lambda)$ is chosen such that the memory constraint

$$\sum_{l=1}^{L} b_l^* d_l^{\text{in}} d_l^{\text{out}} \leq M,$$

is tight (*i.e.*, equality holds). This equality condition emerges naturally, as the objective function (2) is non-increasing in b_l due to the non-negativity assumption ($a_l \geq 0$). Therefore, increasing $C(\lambda)$ until the constraint is exactly met cannot worsen the objective, completing the proof. □

To empirically validate our approximation of quantization errors by their expectation, we quantized random Gaussian matrices multiple times and observed consistently low variance in the quantizataion errors (distortion). Specifically, we performed quantization on 32 random standard Gaussian matrices of shape $(4096, 4096)$, consistent with the LLaMA 3.1-8B self-attention query projection layer. Table 5 reports the mean and standard deviation of the normalized quantization error ($\|\hat{W} - W\|_2^2 / \|W\|_2^2$) values. The results demonstrate low variance, supporting our assumption.

B Analysis of the quantization optimality gap

In practice, we typically have access only to a finite set of quantizers $\mathcal{Q} = \{Q_1, \ldots, Q_N\}$, which may not achieve the theoretical distortion-rate optimality. Under this constraint, the original memory-constrained MSQ problem (1) can still be solved, but the resulting solution may deviate from the optimal solution derived under the assumption of ideal Gaussian quantizers (Theorem 3.1). In this section, we formally analyze this quantization optimality gap.

Table 6: Optimality gap analysis for different quantizer sets on LLaMA 3.1-8B. TCQ-ALL includes all fractional TCQ bitwidths from 1.5 to 5.0 in Q-Palette.

Quantizer pool	Bitwidth	Distortion gap (↓)	Bit allocation gap (↓)	Total gap (↓)	Surrogate objective (↓)
VQ-2,3,4	3.25	0.0586	0.0130	0.0716	0.1219
TCQ-2,3,4	3.25	0.0198	0.0129	0.0327	0.0830
TCQ-ALL	3.25	0.0145	0.0023	0.0168	0.0671
Ideal Gaussian quantizer	3.25	0	0	0	0.0503
VQ-2,3,4	2.50	0.1282	0.0178	0.1460	0.2883
TCQ-2,3,4	2.50	0.0306	0.0178	0.0484	0.1907
TCQ-ALL	2.50	0.0260	0.0034	0.0294	0.1717
Ideal Gaussian quantizer	2.50	0	0	0	0.1423

Let Q_l^* denote the optimal quantizer selected from the quantizer set $\mathcal{Q}$ for each l, obtained by solving problem (1). Then, the quantization optimality gap, defined as the difference in the objective values between the practical optimal solution and the theoretically optimal fractional bitwidth solution b_l^* (derived in Theorem 3.1), can be expressed as:

$$\sum_{l=1}^{L} a_l \left(\mathrm{err}(Q_l^*; W_l) - 2^{-2b_l^*} \right). \tag{5}$$

We decompose the total gap into two intuitive terms, the distortion gap and the bit allocation gap as follows:

$$\underbrace{\sum_{l=1}^{L} a_l \left(\mathrm{err}(Q_l^*) - 2^{-2b_l^*} \right)}_{\text{Total gap}} = \underbrace{\sum_{l=1}^{L} a_l \left(\mathrm{err}(Q_l^*) - 2^{-2\mathrm{bit}(Q_l^*)} \right)}_{\text{Distortion gap}} + \underbrace{\sum_{l=1}^{L} a_l \left(2^{-2\mathrm{bit}(Q_l^*)} - 2^{-2b_l^*} \right)}_{\text{Bit allocation gap}},$$

where we abbreviate $\mathrm{err}(Q_l^*; W_l)$ by $\mathrm{err}(Q_l^*)$ and $\mathrm{bit}(Q_l^*; W_l)$ by $\mathrm{bit}(Q_l^*)$ for the simplicity.

Due to classical rate-distortion theory and the optimality of b_l^* as the solution of the continuous optimization problem (2), each term in this decomposition is non-negative. Specifically, the first term, $\left(\mathrm{err}(Q_l^*) - 2^{-2\mathrm{bit}(Q_l^*)}\right)$, quantifies how closely each practical quantizer Q_l^* approaches the theoretical Gaussian distortion bound. The second term, $\left(2^{-2\mathrm{bit}(Q_l^*)} - 2^{-2b_l^*}\right)$, measures how well the available bitwidths $\{\mathrm{bit}(Q) \mid Q \in \mathcal{Q}\}$ approximate the optimal bit allocation $\{b_l^*\}$.

To investigate how quantizer-set design affects each component of the gap, Table 6 reports the distortion and bit allocation gaps for LLaMA 3.1-8B under 2.5- and 3.25-bit constraints. Two key observations emerge:

- **Effect of quantizer quality.** VQ-2,3,4 and TCQ-2,3,4 show comparable bit allocation gaps, but TCQ-2,3,4 yields much smaller distortion gaps. Thus, their performance difference mainly stems from quantizer quality rather than bit allocation.
- **Effect of broader bitwidth support.** Comparing TCQ-2,3,4 to TCQ-ALL reveals a substantial reduction in bit allocation gap, demonstrating that richer fractional-bitwidth support enables more accurate bit allocation and a closer match to the theoretical ideal.

This analysis motivates the design of *Q-Palette*, which provides high-quality TCQ quantizers and broad fractional-bitwidth coverage to reduce both distortion and bit allocation gaps.

Analyzing these factors provides insight into improving practical quantizer designs and selecting more effective quantizer sets to reduce the quantization optimality gap. Motivated by this analysis, we specifically designed *Q-Palette* as a versatile set of fractional-bit quantizers, including TCQ, which closely approaches the theoretical distortion bound, and providing broad fractional bitwidth support.

C Additional details on quantizers in Q-Palette

In this section, we provide implementation details for each quantizer family in Q-Palette. For each quantizer, we describe: (i) codebook construction, (ii) dequantization, and (iii) quantization

procedures. The quantization step relies on quantizer-specific round-to-nearest (RTN) operators, with procedures differing based on data availability:

- **Data-free scenario:** We partition each weight matrix into scalar elements (NUQ) or vectors (VQ, TCQ). Each partition is independently quantized using the RTN operator, and their resulting binary representations are concatenated to form the final quantized weight representation.
- **Data-aware scenario:** We adopt a block LDLQ framework as introduced in previous methods [17, 49, 50]. Specifically, for each weight matrix, we perform quantization in a block-wise manner guided by Hessian approximations, with quantizer-specific block sizes: 1 for NUQ, 2 for VQ, and 16 for TCQ. This method sequentially processes weight rows from first to last, iteratively updating weights based on the Hessian and cumulative quantization errors, and quantizing each updated weight via the RTN operator. For detailed formulations and additional theoretical background, please refer to QUIP# and QTIP [49, 50].

C.1 Non-uniform scalar quantization (NUQ)

Codebook construction. For NUQ at bitwidth b, we construct the LUT using `flash1dkmeans`, a fast 1D k-means algorithm [25], applied to 10^8 randomly sampled standard Gaussian values. We set the number of clusters to $k = 2^b$, resulting in a LUT $\in \mathbb{R}^{2^b}$.

Dequantization. Given the LUT, a binary representation $\mathbf{r} \in \{0,1\}^b$ is dequantized as:

$$\mathrm{dq}(\mathbf{r}; \mathrm{LUT}) \coloneqq \mathrm{LUT}[\mathrm{int}(\mathbf{r})],$$

where $\mathrm{int}(\mathbf{r})$ converts the binary representation $\mathbf{r} \in \{0,1\}^b$ to its corresponding integer index in the range $[0, 2^b - 1]$.

Quantization. NUQ's RTN operator $\mathrm{RTN} : \mathbb{R} \to \{0,1\}^b$ maps a scalar input $v \in \mathbb{R}$ to the nearest LUT entry:

$$\mathrm{RTN}(v; \mathrm{LUT}) \coloneqq \operatorname*{argmin}_{\mathbf{r} \in \{0,1\}^b} |v - \mathrm{LUT}[\mathrm{int}(\mathbf{r})]|.$$

Quantization follows the general procedures described above for data-free and data-aware scenarios.

C.2 Vector quantization (VQ)

Codebook construction. For VQ at bitwidth b, we construct the codebook using the scikit-learn implementation of the 2D k-means algorithm, which employs Lloyd's algorithm [41, 34]. We set the hyperparameters to `max_iter=300` and `tol=1e-6`, and apply the algorithm to random standard Gaussian samples, using 10^8 samples for bitwidths $b \leq 5$ and 10^7 samples for bitwidths $b > 5$. The number of clusters is set to $k = 2^{2b}$, resulting in a LUT $\in \mathbb{R}^{2^{2b} \times 2}$ consisting of 2^{2b} number of 2D vectors.

Dequantization. Given the LUT, a binary representation $\mathbf{r} \in \{0,1\}^{2b}$ is dequantized similarly to NUQ, now mapping to a vector:

$$\mathrm{dq}(\mathbf{r}; \mathrm{LUT}) \coloneqq \mathrm{LUT}[\mathrm{int}(\mathbf{r})] \in \mathbb{R}^2,$$

where $\mathrm{int}(\mathbf{r})$ converts the binary representation $\mathbf{r} \in \{0,1\}^{2b}$ into its corresponding integer index in the range $[0, 2^{2b} - 1]$.

Quantization. The RTN operator specific to VQ, $\mathrm{RTN} : \mathbb{R}^2 \to \{0,1\}^{2b}$, maps a 2D input vector $\mathbf{v} \in \mathbb{R}^2$ to the nearest LUT entry:

$$\mathrm{RTN}(\mathbf{v}; \mathrm{LUT}) \coloneqq \operatorname*{argmin}_{\mathbf{r} \in \{0,1\}^{2b}} \|\mathbf{v} - \mathrm{LUT}[\mathrm{int}(\mathbf{r})]\|_2.$$

Quantization then follows the general procedures described above for data-free and data-aware scenarios.

C.3 Trellis-coded quantization (TCQ)

C.3.1 Generic TCQ

Codebook construction. We follow the same protocol as QTIP [50], using scikit-learn's k-means implementation based on Lloyd's algorithm [41, 34]. Specifically, we cluster 2^{20} randomly sampled 2D standard Gaussian vectors (with appropriate scaling) into $2^{\texttt{tlut\_bits}}$ clusters, obtaining cluster centroids $\texttt{tlut} \in \mathbb{R}^{2^{\texttt{tlut\_bits}} \times 2}$. We then construct the final codebook $\mathrm{LUT} \in \mathbb{R}^{2^L \times 2}$ using the *hybrid* codebook construction using the following `quantlut_sym` function from the QTIP codebase:

```
def quantlut_sym(tlut, L, tlut_bits):
    with torch.no_grad():
        lut = torch.arange(1 << L, device=tlut.device)
        lut = (lut + 1) * lut
        sflp = 1 - ((lut >> 15) & 1) * 2
        lut = (lut >> (16 - tlut_bits - 1)) & ((1 << tlut_bits) - 1)
    lut = tlut[lut]
    lut[:, 0] = lut[:, 0] * sflp
    return lut
```

Following QTIP, we set $L = 16$ for all TCQ quantizers. We set `tlut_bits` to 9 for bitwidths $b \leq 4$, and to $10, 11$ for new fractional bitwidths $4.5, 5.0$, respectively.

Dequantization. We adopt the bitshift variant of TCQ with tail-biting from QTIP. Given a binary representation $\mathbf{r} \in \{0,1\}^{sT/V}$, we define parameters explicitly as follows:

- s: shift size, set as $s = 2b$ for bitwidth b,
- V: vector size, fixed to $V = 2$,
- L: codebook length (or sliding window size), fixed to $L = 16$,
- T: trellis size, set as $T = 256$.

The dequantization then proceeds via sliding-window LUT indexing with tail-biting:

$$\mathrm{dq}(\mathbf{r}; \mathrm{LUT}) \coloneqq \mathrm{concat}_{i=0}^{T/V-1} \mathrm{LUT}\left[\mathbf{r}[i \cdot s : i \cdot s + L]\right] \in \mathbb{R}^T,$$

where indices exceeding the length of $\mathbf{r}$ wrap around due to tail-biting, resulting in s/V bitwidth.

Quantization. Given a target vector $\mathbf{v} \in \mathbb{R}^T$ to quantize, we use the same RTN operator as detailed in QTIP, which leverages the Viterbi algorithm to find the optimal binary representation $\mathbf{r} \in \{0,1\}^{sT/V}$ that is dequantized into the vector $\hat{\mathbf{v}} = \mathrm{dq}(\mathbf{r}; \mathrm{LUT})$ closest to the vector $\mathbf{v}$ [16, 50]. Quantization procedures follow the general data-free and data-aware frameworks described earlier.

C.3.2 Half-TCQ

Codebook construction. For half-TCQ, which quantizes half of the weight using bitwidth b and the other half using $b + 0.5$, we follow exactly the same codebook construction procedure described above for TCQ at bitwidth $b + 0.5$.

Dequantization. Dequantization separately processes two partitions of the weight matrix: the first half using binary representations of bitwidth b, and the second half using bitwidth $b + 0.5$. The resulting vectors from each half are then concatenated into a complete dequantized weight vector.

Quantization. We apply the RTN operator corresponding to TCQ-b to the first half of the weights, and the RTN operator corresponding to TCQ-$(b + 0.5)$ to the second half. This procedure is consistently used in both the data-free and data-aware (block LDLQ) scenarios.

D Additional details and performance analysis of CUDA kernels

We implemented two types of CUDA kernels: (i) Tensor Core-based kernels and (ii) CUDA Core-based kernels. Here, we first detail our Tensor Core-based kernel implementations. Then, we describe

Table 7: Decoding-latency speedup of quantized LLaMA 3.1-8B models relative to the FP16 baseline on an RTX3090 GPU. 'TC' and 'CC' denote Tensor Core and CUDA Core kernels, respectively.

Decoding-latency speedup compared to FP16 (batch size = 1)										
Quantizer	2.0	2.25	2.5	2.75	3.0	3.25	3.5	3.75	4.0	4.25
NF w/ FLUTE [11, 20]	-	-	-	-	-	1.63×	-	-	-	1.63×
QTIP [50]	2.17×	-	-	-	2.02×	-	-	-	1.92×	-
Ours-NUQ-TC	2.82×	-	-	-	2.53×	-	-	-	2.28×	-
Ours-NUQ-CC	3.07×	-	-	-	**2.75**×	-	-	-	**2.38**×	-
Ours-VQ-TC	2.83×	-	2.54×	-	2.54×	-	2.10×	-	2.28×	-
Ours-VQ-CC	**3.11**×	-	**2.94**×	-	2.74×	-	**2.52**×	-	2.36×	-
Ours-TCQ-TC	2.56×	**2.32**×	2.29×	**2.24**×	2.36×	**2.13**×	2.14×	**2.16**×	2.25×	**1.94**×
Decoding latency speedup compared to FP16 (batch size = 8)										
Quantizer	2.0	2.25	2.5	2.75	3.0	3.25	3.5	3.75	4.0	4.25
NF w/ FLUTE [11, 20]	-	-	-	-	-	1.55×	-	-	-	1.55×
QTIP [50]	0.43×	-	-	-	0.39×	-	-	-	0.36×	-
Ours-NUQ-TC	**2.20**×	-	-	-	**2.00**×	-	-	-	1.85×	-
Ours-NUQ-CC	1.75×	-	-	-	1.70×	-	-	-	1.49×	
Ours-VQ-TC	**2.20**×	-	**1.99**×	-	1.97×	-	1.65×	-	1.84×	-
Ours-VQ-CC	1.88×	-	1.81×	-	1.78×	-	1.69×	-	1.69×	-
Ours-TCQ-TC	2.04×	**1.89**×	1.84×	**1.88**×	1.93×	**1.79**×	**1.72**×	**1.83**×	**1.89**×	**1.67**×

our CUDA Core-based kernel implementations. Finally, we provide additional performance analysis on our kernels.

D.1 Tensor Core-based kernel implementation

Our Tensor Core-based kernels support various quantization schemes (TCQ, NUQ, and VQ) and are implemented by extending the QTIP kernels, which originally supported TCQ at integer bitwidths (2, 3, 4 bits) [50]. Specifically, for TCQ, we introduce optimized support for fractional bitwidths at fine-grained intervals (*e.g.*, 1.5, 2.5, 3.5, 4.5, 5.0 bits) by carefully extending the warp-level `mma` instruction-based implementation provided by QTIP. Additionally, we adapted the QTIP's kernel design principles to implement efficient Tensor Core-based kernels for NUQ and VQ. These extensions involved non-trivial engineering efforts, particularly for precisely mapping quantized weights into Tensor Core `mma` instruction fragments. To further reduce overhead at larger batch sizes, we traverse each quantized weight exactly once, directly performing register-level dequantization upon loading without intermediate storage. Input activations are cached in shared memory to enable efficient reuse across multiple weight multiplications, substantially improving inference efficiency.

Simplified Kernel Structure. Below, we provide a brief kernel structure highlighting key functions, their purposes, and file locations:

```
// kernels/tcq-kernels/src/inference.cu
device void load_reg_cs<R>(compressed, idx, laneId, &regs) {
  // Maps quantized TCQ weights (bitwidth R/2) to mma fragments
  // Supports fractional bitwidths (1.5,2.0,2.5,...,4.5,5.0)
}

// kernels/vq-tensor-kernels/src/inference.cu
device void load_reg_cs<R, LUT_TYPE L>(compressed, idx, laneId, &regs) {
  if (L == LUT_TYPE::SQ_LUT) {
    // Maps quantized NUQ weights (bitwidth R) to mma fragments
  } else if (L == LUT_TYPE::VQ_LUT_2) {
    // Maps quantized VQ weights (bitwidth R/2) to mma fragments
  }
}


// General Tensor Core kernel structure
// - kernels/tcq-kernels/src/inference.cu: kernel_decompress_gemm
```

```
//     for TCQ fused kernel
// - kernels/tcq-kernels/src/inference.cu: kernel_decompress_gemm_combt
//     for TCQ-Half fused kernel
// - kernels/vq-tensor-kernels/src/inference.cu: kernel_decompress_gemm
//     for VQ and NUQ fused kernel
global void tensor_core_kernel(...) {
  // Manages LUT and inputs in shared memory
  // Manages quantized weights in registers
  // Maps quantized weights to Tensor Core mma fragments via 'load_reg_cs'
  // Uses Tensor Core mma instructions for matmul routine
  // Writes final results after reduction
}


// Dequantization-only kernels
// - kernels/tcq-kernels/src/inference.cu: kernel_decompress
//     for TCQ dequantization kernel
// - kernels/tcq-kernels/src/inference.cu: kernel_decompress_combt
//     for TCQ-Half dequantization kernel
// - kernels/vq-tensor-kernels/src/inference.cu: kernel_decompress
//     for VQ and NUQ dequantization kernel
global void kernel_decompress(...) {
  // Dequantizes weights independently (no matmul)
}
```

Full Implementation. The complete Tensor Core-based kernel implementations are included in our public code release (directories `kernels/tcq-kernels` and `kernels/vq-tensor-kernels`).

D.2 CUDA Core-based kernel implementation

Our CUDA Core-based kernels explicitly leverage CUDA Core instructions to support NUQ and VQ quantization schemes, extending the Any-Precision LLM kernels originally developed for NUQ [40]. Specifically, we replaced the original bit-plane encoding with simpler bit-packing encoding to streamline the dequantization procedure.

Kernel Structure and Implementation Below, we provide a brief kernel structure highlighting key functions, their purposes, and file locations:

```
// - kernels/vq-cuda-kernels/src/gemm_routines.cu
device void vq_pack_dequant_routine<nbits, vec_sz>(Bcode, B_row, shC) {
  // Unpack quantized VQ weights 'Bcode' (bitwidth nbits/vec_sz)  \
  //      to the half2 array 'B_row' using the lookup-table 'shC' \
  //     (e.g., 1.5-bit quantization: nbits=3, vec_sz=2)
}

// - kernels/sq-cuda-kernels/gemm_routines.cu
device void pack_dequant<nbits>(Bcode_row, B_row, shC) {
  // Unpack quantized NUQ weights 'Bcode' (bitwidth nbits) \
  //      to the half2 array 'B_row' using the lookup-table 'shC'
}


// General CUDA Core kernel structure
// - kernels/sq-cuda-kernels/gemm_routines.cu: sq_gemm_fp16
//     for NUQ fused kernel
// - kernels/vq-cuda-kernels/src/gemm_routines.cu: vq_pack_gemm_fp16
//     for VQ fused kernel
global void cuda_core_kernel(...) {
  // Manages LUT in shared memory
```

```
  // Manages quantized weights in registers
  // Unpacks quantized weights to half2 array via corresponding
  //     pack_dequant_routine
  // Uses CUDA Core half-precision FMA (hfma2) instructions for matmul
  // Writes final results after reduction
}


// Dequantization-only kernels
// - kernels/sq-cuda-kernels/gemm_routines.cu: pack_dequant_kbit_store
//     for NUQ dequantization kernel
// - kernels/vq-cuda-kernels/src/gemm_routines.cu: \
//                                          vq_pack_dequant_kbit_store
//     for VQ dequantization kernel
global void kernel_decompress(...) {
  // Dequantizes weights independently (no matmul)
}
```

Full Implementation. The complete CUDA Core-based kernel implementation is included in our public code release (directories `kernels/sq-cuda-kernels` and `kernels/vq-cuda-kernels`).

D.3 Additional performance analysis

This section complements Table 2 in the main paper by providing additional performance analysis on a different hardware configuration (with RTX3090 GPU). Table 7 summarizes the decoding-latency speedup of quantized LLaMA 3.1-8B models relative to the FP16 baseline on an RTX3090 GPU, complementing Table 2 in the main paper, which reports results on an RTX4090 GPU. Our quantizers consistently outperform baseline methods across both evaluated batch sizes (1 and 8).

Notably, for batch size 1 on RTX3090, the latency speedup gap between our CUDA Core-based ('CC') and Tensor Core-based ('TC') kernels is larger than observed on RTX4090 (Table 2). This difference highlights significant hardware dependencies in kernel performance, justifying the importance of providing multiple kernel implementations. Such flexibility enables optimal kernel selection tailored to specific hardware platforms and workload requirements.

E Implementation details of mixed-scheme quantization

E.1 Loss term computation

Data-free loss term. In data-free scenarios, we estimate the loss term via the linearity theorem [35], *i.e.*, $\ell_{lq} = a_l \cdot \mathrm{err}(Q_q; W_l)$. Specifically, we approximate the quantization error $\mathrm{err}(Q_q; W_l)$ using pre-computed distortion values obtained by quantizing random standard Gaussian matrices, thereby avoiding explicit quantization for each weight matrix W_l. To determine the sensitivity coefficient a_l, we adopt the procedure introduced in HIGGS [35]. First, we randomly generate 128K tokens from the given LLM. Then, for each layer l, we inject random Gaussian noise scaled to specific norms $n_{li} = \frac{\sqrt{i}}{16}$ for $1 \le i \le 16$ and measure the resulting increase in the KL-divergence loss computed over these 128K tokens:

$$\Delta\mathcal{L}_{li} = \mathcal{L}\left(\{W_{l'} + \delta_{l'l} \cdot n_{li} \cdot \|W_l\|_2 \cdot \epsilon_l / \|\epsilon_l\|_2\}_{l'=1}^{L}\right) - \mathcal{L}\left(\{W_{l'}\}_{l'=1}^{L}\right), \quad \epsilon_l \sim \mathcal{N}(0, I),$$

where $\delta_{l'l}$ is the Kronecker delta (1 if $l' = l$, else 0), ensuring noise is injected exclusively into layer l, and ϵ_l is a standard Gaussian noise matrix matching the dimensions of layer l. Due to the linearity theorem, the increase in loss approximately follows the linear relation $\Delta\mathcal{L}_{li} \simeq n_{li}^2 a_l$, enabling us to estimate a_l by linearly fitting the data points $(n_{li}^2, \Delta\mathcal{L}_{li})$. This procedure requires $16 \times L$ computations of the KL-divergence loss, where L is the total number of layers, but it can be performed in an embarrassingly parallel manner. Furthermore, the computed sensitivity coefficients a_l can be reused for all data-free MSQ scenarios, incurring only a one-time computational cost.

Data-aware loss terms. For data-aware scenarios, we employ two different types of loss terms: *linearity* and *actual*.

- **Linearity-based loss term.** Similar to the data-free scenario, we utilize the linearity theorem-based approximation. However, we replace the KL-divergence loss computed over randomly generated 128K tokens with the perplexity loss computed over 1M tokens from the RedPajama dataset [52].
- **Actual loss term.** Here, we explicitly calculate the actual perplexity increase caused by quantization. Specifically, we use 256K tokens from the RedPajama dataset and compute the loss term as follows:

$$\ell_{lq} \coloneqq \mathcal{L}(\{Q^{l'}_{\text{default}}(W_{l'})\}, \text{replace } l\text{-th layer with } Q_q(W_l)) - \mathcal{L}(\{Q^{l'}_{\text{default}}(W_{l'})\}),$$

  where Q^l_{default} denotes the default quantizer for layer l (*e.g.*, we use TCQ-2 for 2-bit quantization, TCQ-3 for 3-bit quantization). This explicitly measures the actual empirical perplexity increase caused by quantize layer l with a quantizer Q_q.

For the data-aware experiments in Table 4 of the main paper, we exclusively employ the 'actual' loss term. Additionally, in the ablation study provided in Table 9 (see Appendix G), we explicitly compare the performance using the two types of loss terms, 'actual' and 'linearity'.

E.2 Latency profiling

For latency profiling, we measure the execution time of the CUDA kernels corresponding to each quantizer. Since computations such as normalization layers, rotations, and self-attention operations remain identical across quantizers, we specifically profile the latency of each fused dequantization and matrix multiplication kernel. To accurately estimate the overall inference latency, we first measure the end-to-end inference latency of several randomly selected quantization configurations using `torch.compile`. We then subtract the sum of kernel latencies to estimate the latency overhead caused by common computations (*e.g.*, normalization layers), uniformly distributing this overhead across all quantizer profiles.

When considering fused kernels, we separately measure kernel latencies corresponding to each fusion pattern. To account for latency overhead variations due to different fusion patterns, we adjust this overhead bias accordingly.

E.3 Optimizer

To solve the mixed-scheme quantization optimization problem, we employ Google's OR-Tools optimization suite [42], specifically utilizing the SCIP solver with a time limit of 60 seconds.

F Settings for Figure 1

In Figure 1, we present qualitative comparisons among quantization frameworks based on Q-Palette and the NormalFloat baseline with FLUTE kernels [11, 20], evaluated on the LLaMA 3.1-8B model using an RTX4090 GPU at a batch size of 1. Specifically, we validate WikiText2 perplexity at a sequence length of 8192 and measure the inference speedup compared to the FP16 baseline under the following detailed settings:

NormalFloat (Baseline). For NormalFloat, we employ FLUTE with a codebook size of 2^3 and a group size of 64, resulting in an average bitwidth of 3.25. We utilize the FLUTE kernels released in the FLUTE codebase [11, 20], optimized for inference on an RTX4090 GPU, to measure inference latency and compute the speedup.

Single-scheme quantization with TCQ-3.25 (Ours). We apply data-free quantization uniformly to all layers of the LLaMA 3.1-8B model using our TCQ-3.25 quantizer from Q-Palette. This corresponds to the half-TCQ scheme which quantizes half of the weight matrix at bitwidth 3.0 and the other half at 3.5.

MSQ with Q-Palette (Ours). We leverage the full set of quantizers available in Q-Palette as our search space. For NUQ and VQ quantizers, both Tensor-Core and CUDA-Core kernel implementations are considered during optimization. Sensitivity coefficients a_l are computed following the

HIGGS protocol [35] as detailed in Appendix E, and we utilize pre-computed distortion values as explained in Section 4.1 of the main paper. Given these sensitivity coefficients, distortion values, and pre-profiled latency measurements, we solve the latency-constrained MSQ optimization (Equation (3) in the main paper) to identify Pareto-optimal quantizer selections under various latency constraints. From this resulting accuracy-latency trade-off curve, we select the configuration that clearly improves both latency and perplexity over the TCQ-3.25 baseline.

Fusion-aware MSQ with Q-Palette (Ours). We further incorporate layer fusion into our MSQ formulation by additionally profiling latency measurements for fused linear-layer combinations and solving the fusion-aware optimization (Equation (4) in the main paper). This approach explicitly captures the latency reductions achievable via layer fusion, enabling joint optimization of quantization schemes and layer fusion decisions. We select a quantization configuration that clearly improves both inference speed and perplexity compared to the MSQ baseline without layer fusion.

The detailed quantization configurations correspond to these scenarios are visualized in Figure 1.

G Experimental settings and additional results

G.1 Experimental settings

G.1.1 Evaluation metric details

For evaluating language modeling performance, we measure perplexity on the WikiText2 dataset [37], using sequence lengths of 4096 tokens for LLaMA 2 models and 8192 tokens for LLaMA 3 models. Additionally, we report zero-shot accuracy on five downstream tasks: ARC-easy, ARC-challenge, HellaSwag, PiQA, and WinoGrande [37, 8, 56, 3, 44]. Zero-shot evaluations are conducted using the `lm_eval` library (version 0.4.4).

G.1.2 Device details

RTX 4090 GPU experiments. We conduct our RTX 4090 GPU experiments using a cloud environment provided by RunPod, with the following hardware and software specifications:

- **GPU:** NVIDIA RTX 4090
- **CPU:** AMD EPYC 7B13 64-Core Processor
- **OS:** Ubuntu 22.04.5
- **CUDA Version:** 12.4

RTX 3090 GPU experiments. We conduct our RTX 3090 GPU experiments using our local machine, detailed as follows:

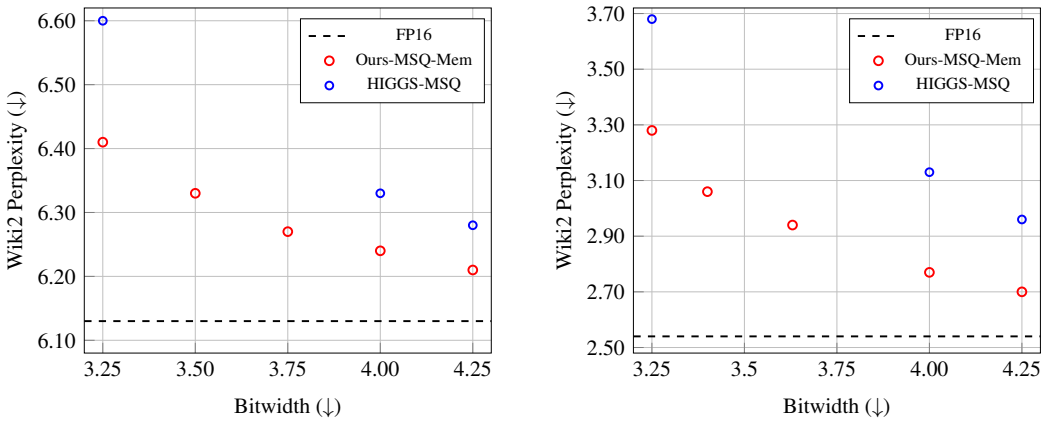
- **GPU:** NVIDIA RTX 3090
- **CPU:** AMD EPYC 7402 24-Core Processor
- **OS:** Ubuntu 22.04.1
- **CUDA Version:** 12.4

G.1.3 Baseline configurations

HQQ [2]. According to the official documentation, inference acceleration kernels (*e.g.*, Gemlite) are supported only for configurations with `axis=1`. Thus, we use the following configurations:

- 4.25-bit: `nbits=4, group_size=64, axis=1` (Gemlite kernel),
- 4.02-bit: `nbits=4, group_size=1024, axis=1` (Gemlite kernel),
- 3.25-bit: `nbits=3, group_size=64, axis=1` (FLUTE kernel).

For 4-bit instances, we utilize Gemlite kernels following best practices from the HQQ documentation [22, 2]; for the 3-bit instance, we report inference time using the FLUTE kernel [20].



(a) Qwen 2.5-7B

(b) LLaMA 3.1-70B

Figure 5: Mixed-scheme quantization results on Qwen 2.5-7B and LLaMA 3.1-70B models. To accommodate the broader sensitivity range in LLaMA 3.1-70B, we extended the quantizer set to include higher-bitwidth options (NUQ 7/8 bits and VQ 5.5/6 bits), in addition to the TCQ quantizers.

NormalFloat [11] We utilize FLUTE's NormalFloat implementation with configurations similar to HQQ [20, 2]:

- 4.25-bit: `nbits=4, group_size=64`,
- 4.02-bit: `nbits=4, group_size=1024`,
- 3.25-bit: `nbits=3, group_size=64`.

Since the publicly available optimized FLUTE kernel does not support a group size of 1024, we only report inference latency results for the 4.25-bit and 3.25-bit configurations.

QTIP [50]. For data-free QTIP, we approximate the Hessian as the identity matrix and follow the same algorithmic implementation as the original data-aware QTIP. For data-aware QTIP, we use the publicly available Hessian approximation from the `relax-ml` HuggingFace repository, computed using 6144×4096 tokens.

HIGGS [35]. As the implementation of HIGGS is not publicly available, we directly report results from their paper. Specifically, for the mixed-scheme baseline, HIGGS provides only a single result for each bitwidth, which we directly use in our comparison. For the single-scheme VQ baseline, HIGGS reports multiple configurations for each bitwidth; we select the configuration achieving the lowest WikiText2 perplexity. Although these single-scheme VQ configurations may not be efficiently realizable in practice due to non-power-of-two codebook sizes, we include them for completeness and comparison purposes.

G.2 Additional results

G.2.1 Memory-constrained mixed scheme quantization results on additional models

To demonstrate the generality of our method, we applied our MSQ method to Qwen 2.5-7B (non-LLaMA) and LLaMA 3.1-70B (large-scale), comparing against HIGGS-MSQ under various bitwidth constraints [55, 18, 35]. Here, for LLaMA 3.1-70B, due to its substantially larger model size, we used 64K tokens (*i.e.*, half of the default setting explained in Appendix E) to estimate the sensitivity coefficients while keeping the rest of the quantization pipeline unchanged.

As shown in Figure 5, our method consistently outperforms HIGGS-MSQ under the same bitwidth constraints (3.25, 4.00, 4.25) on both models. Additionally, our method achieves comparable or better perplexity at lower bitwidths compared to HIGGS-MSQ. For Qwen 2.5-7B, our 3.5-bit model matches the performance of HIGGS-MSQ at 4.00 bits, and our 3.75-bit result slightly improves upon the HIGGS-MSQ result at 4.25 bits, yielding up to 12.5% memory savings. A similar trend is observed

Table 8: Ablation study of layer fusion and CUDA-Core kernel usage on inference throughput and WikiText2 perplexity (LLaMA 3.1-8B, batch size=1, RTX4090). 'TC' and 'CC' denote Tensor Core and CUDA Core kernels, respectively.

Method	Throughput (Toks/s) (↑)	Wiki2 (↓)
FP16	62	5.61
Ours-VQ-2 (single scheme)	231	5905.08
Ours-MSQ-Lat (No Fusion, TC Only)	228	119.72
Ours-MSQ-Lat (Fusion-aware, TC only)	232	8.93
Ours-MSQ-Lat (Fusion-aware, TC and CC)	231	**8.71**
Ours-TCQ-2 (single scheme)	223	37.95
Ours-MSQ-Lat (No Fusion, TC Only)	223	20.33
Ours-MSQ-Lat (Fusion-aware, TC only)	224	7.79
Ours-MSQ-Lat (Fusion-aware, TC and CC)	223	**7.69**
Ours-TCQ-3 (single scheme)	185	6.78
Ours-MSQ-Lat (No Fusion, TC Only)	187	6.47
Ours-MSQ-Lat (Fusion-aware, TC only)	186	6.06
Ours-MSQ-Lat (Fusion-aware, TC and CC)	185	**6.03**

Table 9: Ablation comparing 'linearity' vs. 'actual' loss terms in data-aware MSQ (LLaMA 2-7B, RTX4090 GPU)

	LLaMA 2 7B				
				Throughput (Toks/s)	
Method	Bits	Wiki2 (↓)	Acc (↑)	$B = 1$	$B = 8$
FP16	16.00	5.12	64.9	71	527
QTIP	2.00	6.84	58.9	209	386
Ours-MSQ-Mem (linearity)	2.00	6.69	**60.3**	270	**1690**
Ours-MSQ-Mem (actual)	2.00	**6.47**	**60.3**	**272**	1684
QTIP	3.00	5.39	63.3	184	304
Ours-MSQ-Mem (linearity)	3.00	5.38	**64.3**	**225**	**1501**
Ours-MSQ-Mem (actual)	3.00	**5.34**	63.9	224	1489

on LLaMA 3.1-70B, where our 3.40-bit and 3.63-bit results slightly outperform HIGGS-MSQ at 4.00 and 4.25 bits, respectively, resulting in up to 15% memory savings at better perplexity. These results demonstrate the broad applicability of our MSQ method.

G.2.2 Ablation on layer fusion and CUDA-Core kernel integration

Table 8 presents an ablation study evaluating the impact of layer fusion and the integration of CUDA-Core kernels on inference throughput and WikiText2 perplexity for the quantized LLaMA 3.1-8B model. We compare single-scheme baselines (VQ-2, TCQ-2, and TCQ-3) in Q-Palette against several MSQ variants: MSQ without layer fusion (Tensor Core kernels only), fusion-aware MSQ using only Tensor Core kernels, and fusion-aware MSQ combining both Tensor Core and CUDA Core kernels. Results demonstrate significant improvements in WikiText2 perplexity when applying fusion-aware MSQ compared to single-scheme quantization and MSQ without fusion, highlighting the effectiveness of jointly optimizing quantization schemes and layer fusion. For example, MSQ without fusion achieves 20.33 perplexity at 223 tokens/sec, while our fusion-aware MSQ achieves a significantly reduced perplexity of **7.79** at 224 tokens/sec. Additionally, integrating CUDA Core kernels alongside Tensor Core kernels provides further performance improvement.

Table 10: Per-group quantization results on LLaMA 3.1-8B (group size $= 64$) without IP. For MSQ, we used a pool of per-group NUQ quantizers ranging from 2 to 8 bits with group size 64. All results are reported without IP.

Method	Bitwidth	Bit allocation strategy	Wiki2 (↓)
G64-NUQ-3	3.25	Uniform bitwidth	8063.91
G64-MSQ-Mem	3.25	MSQ with Gaussian-assumed error	7.34
G64-MSQ-Mem	3.25	MSQ with true quantization error	**7.33**
G64-NUQ-4	4.25	Uniform bitwidth	6.10
G64-MSQ-Mem	4.25	MSQ with Gaussian-assumed error	**5.89**
G64-MSQ-Mem	4.25	MSQ with true quantization error	5.90

G.2.3 Effect of loss-term choice in data-aware MSQ

Table 9 provides an ablation study comparing two different loss-term definitions ('linearity' vs. 'actual') used in data-aware MSQ quantization for LLaMA 2-7B. The 'linearity' loss term efficiently approximates the increase in perplexity loss via sensitivity coefficients measured using the linearity theorem, enabling reuse across multiple quantization configurations, similar to the data-free scenario (see Appendix E). In contrast, the 'actual' loss term explicitly computes the empirical (validation) perplexity increase caused by quantization. Our results demonstrate that using the computationally efficient 'linearity' loss term achieves comparable zero-shot accuracy improvements to those obtained with the 'actual' loss term, indicating that the simpler and reusable linearity-based approach is also effective in practice. Additionally, both loss-term approaches achieve similar inference throughput, reinforcing the practicality of the computationally efficient 'linearity' loss term.

G.2.4 Applicability of MSQ with linearity-theorem surrogate without incoherence processing

A natural question is whether the MSQ framework based on the linearity-theorem surrogate [35] remains applicable when incoherence processing (IP) is not available due to the hardware constraints. This surrogate objective requires sensitivity coefficients and per-layer quantization errors; with IP, weights are nearly Gaussian, allowing these errors to be precomputed from random Gaussian matrices as explained in Appendix E.

To examine the no-IP case, where per-group quantization is typically adopted to handle weight outliers, we disable IP and quantize on a per-group basis (group size $= 64$) using fixed Gaussian-trained codebooks. We compare three bit allocation strategies: 1) uniform bitwidth, 2) MSQ with *Gaussian-assumed error*, which relies on cached Gaussian distortion estimates for $\mathrm{err}(Q_q; W_l)$ and solves Equation (1), and 3) MSQ with *true quantization error*, which measures layerwise distortion $\mathrm{err}(Q_q; W_l)$ directly and solves Equation (1).

As shown in Table 10, MSQ significantly outperforms uniform bitwidth even in the no-IP case. Moreover, the Gaussian-assumed error achieves perplexity almost identical to that from the true quantization error (*e.g.*, 7.34 vs. 7.33 at 3.25 bits), providing preliminary evidence that cached Gaussian-based error estimates may remain reliable in the no-IP case.

H Limitations and future work

We introduce Q-Palette, a comprehensive suite of quantizers spanning a wide range of trade-offs across memory footprint, inference latency, and quantization error, offering versatile options suitable for diverse deployment scenarios. To demonstrate its effectiveness, we integrate Q-Palette into an MSQ framework and validate its ability to achieve improved performance-efficiency trade-offs under PTQ settings. However, our framework is designed around one-shot MSQ objectives, which rely on layer-wise second-order approximations of end-to-end loss and are primarily applicable to scenarios that do not involve retraining [35]. While Q-Palette can also serve as a building block for retraining-based quantization workflows such as quantization-aware training, which may be preferable in cases where larger computational budgets and data are available, we have not evaluated its effectiveness in that setting, as our primary focus is on data-free or calibration-light PTQ. Extending Q-Palette to retraining-based quantization workflows thus remains a promising direction for future work.

Another limitation lies in the cost of computing the sensitivity coefficients. Currently, evaluating these coefficients requires $O(L)$ computations of the KL-divergence loss in data-free setting as explained in Appendix E, which can become a bottleneck as the model size grows. Although this computation can be performed in an embarrassingly parallel manner and the resulting coefficients can be reused across all MSQ runs, thus representing a one-time cost, the overhead may still be non-negligible when the set of target memory or latency is fixed and reuse is limited. Developing methods to further reduce this cost is therefore an interesting direction for future research.

One promising direction is the extension of Q-Palette to weight-activation quantization. In this work we focus on weight-only PTQ, which is particularly effective in memory-bound inference settings with small batch sizes, such as on laptops or mobile devices where memory bandwidth, rather than compute, is the primary bottleneck. However, on some hardware accelerators, such as the Qualcomm Hexagon NPU, which natively support only integer (e.g., INT8) GEMM, activation quantization is essential for exploiting their full performance. Thus, extending Q-Palette to support weight-activation quantization is a natural direction for broader deployment. One potential approach is a two-stage scheme: (1) first quantize weights to INT8 using uniform W8A8 quantizers for hardware compatibility; and (2) then apply a secondary compression step that further quantizes the INT8 weights into x-bit representations using a variant of Q-Palette quantizers whose codebooks are constrained to the INT8 grid. During inference, the compressed weights are dequantized back to INT8 and then processed using integer GEMM with INT8-quantized activations, enabling compatibility with INT8-only hardware while reducing memory usage. A similar idea was introduced in Q-Serve, which quantizes weights in two stages, first to symmetric INT8 grid and then to asymmetric INT4 [31]. We consider exploring such extensions an interesting direction for future work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly reflect the paper's contributions and scope by explicitly highlighting our key contributions: (1) deriving the information-theoretically optimal bit allocation strategy, (2) introducing *Q-Palette*, a comprehensive suite of fractional-bit quantizers (including trellis-coded, vector, and scalar quantizers) with efficient CUDA kernel implementations, and (3) proposing a novel *fusion-aware mixed-scheme quantization* framework. All these claims are thoroughly supported through theoretical analyses, detailed methodology, kernel implementation descriptions, and extensive empirical evaluations presented in the main paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in Appendix H.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We state the theoretical result in Section 3.1. The assumptions and complete proofs corresponding to our theoretical results are provided in Appendices A and B.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all the necessary details to reproduce the main experimental results in Appendix G (for experimental settings), Appendix D (for kernels), Appendix F (for Figure 1), and Appendix E (for details on MSQ).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We include the implemented CUDA kernels and the codebase required to reproduce the experimental results as supplemental material in the submission, and we have released the code on GitHub.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All essential details regarding experimental settings are clearly documented in Appendices D to G.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: We comprehensively evaluate our methods across multiple models (LLaMA 2-7B, 13B, LLaMA 3.1-8B, 3.2-1B, 3B) and various bitwidths using standard metrics (WikiText2 perplexity and zero-shot accuracy measured by `lm_eval`). However, we do not explicitly report error bars or statistical significance tests, as it is common practice in quantization literature to demonstrate robustness through comprehensive evaluations across multiple models and bitwidths rather than conducting isolated statistical tests.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information regarding the computing resources used is provided in Appendix D for CUDA kernel evaluations and Appendix G for the main experimental settings.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, our research complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Potential societal impacts are discussed in the Impact statement section of the main paper.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work focuses on quantization methods and its compute kernels. It does not involve releasing models, data, or other artifacts with high risk of misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We credit the libraries used in our experiments (Appendix G) and acknowledge the original kernel implementation upon which our kernel implementation builds (Appendix D).

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide our CUDA kernel implementations and the codebase for reproducing results with a `README.md` file, as a zip file in the supplemental material at submission, and we have released the code on GitHub.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Our work focuses on quantization methods and its compute kernels. It does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: It does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLMs were only used for editing purposes and did not affect the core methodology, scientific rigor, or originality of the research.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.