SFMP: Fine-Grained, Hardware-Friendly and Solver-Free Mixed-Precision Quantization for Large Language Models

Xin Nie[1] **Haicheng Zhang**[1] **Liang Dong**[1] **Beining Feng**[1] **Jinhong Weng**[1] **Guiling Sun**[1]

[1] College of Electronic Information and Optical Engineering, Nankai University

Correspondence: sungl@nankai.edu.cn

Abstract

Mixed-precision quantization is a promising approach for compressing large language models under tight memory budgets. However, existing mixed-precision methods typically suffer from one of two limitations: they either rely on expensive discrete optimization to determine precision allocation, or introduce hardware inefficiencies due to irregular memory layouts. We propose **SFMP**, a solver-free and hardware-friendly mixed-precision quantization framework for large language models. SFMP defines a theoretically simple objective for mixed-precision quantization and is built upon three novel ideas: 1) *Block-wise mixed-precision*, enabling fine-grained precision within weight matrices while remaining hardware-friendly; 2) *Row-column weight reordering*, which improves structural alignment via row and column reordering, incurring only a small activation reordering overhead during inference; 3) *Unified GEMM kernel*, which supports block-wise mixed-precision GEMM at arbitrary average bit-widths. Extensive experiments demonstrate that SFMP outperforms state-of-the-art layer-wise mixed-precision methods under the same memory constraints, while significantly reducing quantization cost and improving inference efficiency. Code is available at https://github.com/Nkniexin/SFMP.

1 Introduction

Weight quantization is an efficient approach to compressing large language models (LLMs). It requires no modifications to the model architecture and directly maps high-precision continuous weights to a discrete space, reducing the average bits of model parameters, which enables deployment of LLMs in memory-constrained edge scenarios (Zhang et al., 2024; Hosseinzadeh and Khamfroush, 2025; Husom et al., 2025). Existing methods (Frantar et al., 2023; Xiao et al., 2023; Lin et al., 2024; Kim et al., 2024; Liu et al., 2025b) achieve near-lossless compression at 8-bit precision and only incur 1–3% accuracy loss at 4-bit.

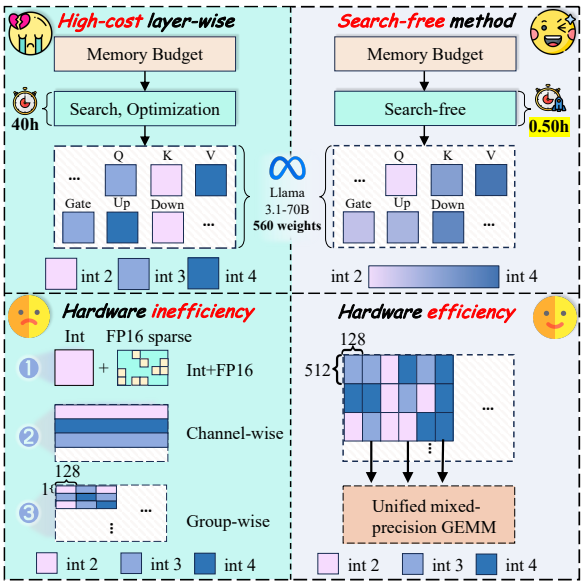


Figure 1: Comparison of SFMP with other mixed-precision quantization methods.

For ultra-large models, such as LLaMA3.1-70B, assigning a uniform integer bit-width to all linear layers limits the choices of model size and cannot adapt to diverse memory budgets. To address this limitation, layer-wise mixed-precision quantization (Lee et al., 2025; Cheng et al., 2025; Liu et al., 2025a; Dong et al., 2020) assigns different integer bit-widths to each linear layer, enabling flexible compression under given memory constraints. However, from an optimization perspective, layer-wise mixed-precision quantization constitutes a constrained integer programming problem (ILP), which is known to be NP-hard (Hochba, 1997). For example, for LLaMA3.1-70B (560 weight matrices) with candidate bit-widths $\{2, 3, 4\}$, the search space is 3^{560}. Existing methods typically transform the problem to fit off-the-shelf ILP solvers (Bellman, 1966; Wolsey, 2020) or heuristic algorithms (Deb et al., 2002; Kirkpatrick et al., 1983) to obtain a relatively good solution in a short time. Even

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 2233–2259
November 6-10, 2026 ©2026 Association for Computational Linguistics

so, as shown in Fig. 1, state-of-the-art layer-wise mixed-precision methods AMQ (Lee et al., 2025) still require 44 hours to search for the bit allocation of LLaMA3.1-70B. This raises a question: **"Under a given memory budget, can we design a strategy to obtain a near-optimal bit allocation without any search or solver-based optimization?"**

Beyond layer-wise mixed-precision quantization, many methods introduce finer-grained strategies, such as channel-wise (Chen et al., 2024b; Wang et al., 2024), group-wise (Huang et al., 2025; Hooper et al., 2025), or even element-wise schemes (Kim et al., 2024; Zhao and Yuan, 2025; Li et al., 2023) in a weight matrix. Although such strategies can further improve model accuracy, they induce irregular memory access patterns and incur substantial overhead in weight packing and unpacking, which significantly degrades inference performance. Some approaches (Kim et al., 2024; Chen et al., 2024b) attempt to mitigate this inference speed degradation with custom compute kernels (Li et al., 2024; Qin et al., 2020; Liu et al., 2025c). However, the resulting speedup is limited. For example, our empirical study in Fig. 10 shows that the group-wise mixed-precision method SliM-LLM (Huang et al., 2025) suffers 30–50% lower inference throughput than GPTQ (Frantar et al., 2023) at the same average bit. Additionally, combining these fine-grained schemes with layer-wise mixed-precision further complicates the discrete optimization problem. This raises another important question: **"Can we design a quantization format that achieves fine-grained mixed-precision while remaining hardware-friendly?"**

In this paper, to address these limitations, we propose **SFMP**, a **Solver-free Mixed-precision** framework. SFMP eliminates the need to solve complex integer programming problems under memory constraints, reducing the time required for compressing LLMs. For example, SFMP completes bit allocation for LLaMA3.1-70B in just 30 minutes. Moreover, SFMP is built upon three key ideas: 1) **Block-wise mixed-precision**: a format achieves fine-grained mixed-precision while remaining hardware-friendly; 2) **Row-column weight reordering**: weights are reordered along rows and columns to align with the block-wise format, incurring only a small activation reordering overhead during inference; 3) **Unified GEMM kernel**: for weight matrices composed of heterogeneous-precision blocks, we propose a unified kernel for memory layout and mixed-precision GEMM execution. Fig. 1 shows the overview of SFMP.

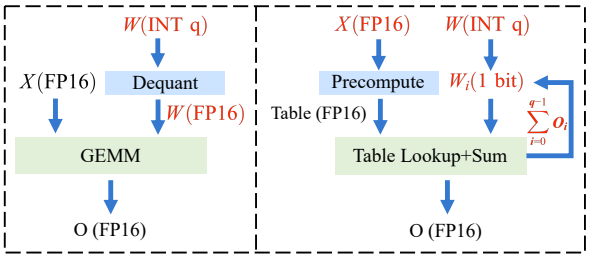


Figure 2: Comparison of two GEMM computation paradigms: Left) dequant-based GEMM; Right) one-bit LUT-based GEMM.

2 Related Works

Salience-Aware Mixed-Precision Quantization. Weight salience is widely used to guide mixed-precision quantization. GPTQ (Frantar et al., 2023) reorders the weight matrix column-wise according to the diagonal entries of the Hessian, prioritizing columns associated with larger diagonal values during quantization. SqueezeLLM (Kim et al., 2024) computes the global Fisher Information Matrix (Ly et al., 2017) and retains a small set of salient weights in high precision. BiLLM (Huang et al., 2024) observes that salience often concentrates along specific rows or columns and reduces quantization error through column-wise partitioning. Similarly, Slim-LLM (Huang et al., 2025) exploits row-wise salience by introducing group-wise mixed-precision quantization, achieving improved accuracy over fixed-precision schemes under the same average bit-width.

One-bit LUT-Based GEMM. Some prior works (Wei et al., 2025; Park et al., 2025b, 2024; You et al., 2024; Ganji et al., 2023; Park et al., 2025a) introduce a dequantization-free compute paradigm for FP-INT GEMM. As shown in Fig. 2, a q-bit quantized weight matrix is decomposed into q one-bit matrices. For an activation vector of group size g, the dot products between the activation and all 2^g possible binary patterns are precomputed and stored in a lookup table (LUT). Consequently, the GEMV operation between the activation vector and the one-bit weight matrix can be replaced by LUT accesses and accumulation. This paradigm eliminates the overhead of weight unpacking at runtime, particularly for ultra low-bit quantization. Moreover, it provides a unified computation kernel: The GEMM computation between activation and a weight matrix of arbitrary integer bit-width can be expressed as a linear combination of one-bit

GEMMs. Owing to its LUT-dominated execution, this paradigm has been demonstrated to be energy-efficient (Jeon et al., 2020; Cicek et al., 2022; Kim et al., 2025), making it especially suitable for edge devices. A detailed description of the computation flow is provided in Appendix B.

3 SFMP

3.1 Preliminaries

As pointed out in prior works (Kim et al., 2024; Zhao and Yuan, 2025), the impact of quantized weights on the model output can be estimated via Taylor expansion. Assuming the model has converged, the loss variation $\Delta\mathcal{L}$ induced by a perturbation from W to W' can be approximated by a second-order expansion:

$$\Delta\mathcal{L} \approx (W - W')^\top H (W - W'), \tag{1}$$

where H denotes the Hessian of the global loss with respect to weights. Since computing the full Hessian is intractable for large-scale models, we approximate it using the Fisher Information Matrix (Ly et al., 2017), $H \approx \mathcal{F} = \mathbb{E}[gg^\top]$, where g denotes the gradient of the loss with respect to the weights, and the expectation is taken over a small calibration dataset. Following prior work (Kim et al., 2024), we adopt a diagonal approximation by ignoring cross-weight interactions. Therefore, the quantization-induced perturbation of the loss can be written as:

$$\Delta\mathcal{L} \approx \sum_{i=1}^{N} \mathcal{F}_{ii}(W_i - W_i')^2, \tag{2}$$

where N denotes the total number of scalar weights in the model. Detailed derivations are provided in Appendix C.

3.2 Objective of SFMP

From Eq. 2, we formulate the mixed-precision bit allocation problem as follows. Given a candidate set of integer bit-widths $\mathcal{B} = \{b_1, b_2, \ldots, b_q\}$ with $b_i \in \mathbb{Z}_{>0}$ and $b_1 < b_2 < \cdots < b_q$, we treat each weight element as the minimal quantization unit. For a target average bit-width $b \in \mathbb{R}_{>0}$, the goal is to find coefficients $\{\alpha_1, \alpha_2, \ldots, \alpha_q\}$ such that

$$\begin{aligned} \min \quad & \Delta\mathcal{L} \approx \sum_{i=1}^{N} \mathcal{F}_{ii}(W_i - W_i')^2, \\ \text{s.t.} \quad & \sum_{k=1}^{q} \alpha_k = 1, \sum_{k=1}^{q} \alpha_k b_k = b. \end{aligned} \tag{3}$$

To make the objective tractable, we consider the expected loss increase under stochastic quantization (Agustsson and Theis, 2020). Therefore, the objective becomes

$$\mathbb{E}[\Delta\mathcal{L}] \approx \sum_{i=1}^{N} \mathcal{F}_{ii}\mathbb{E}[(W_i - W_i')^2]. \tag{4}$$

Specifically, we assume that each weight is quantized independently under a uniform quantizer with quantization step δ, and the dynamic range $(W_{\max} - W_{\min})$ is approximately constant across weights (Appendix G.6). Under this assumption, the quantization error can be modeled as a uniform random variable $x \sim \mathcal{U}(-\delta/2, \delta/2)$, leading to

$$\mathbb{E}[(W_i - W_i')^2] = \frac{1}{\delta}\int_{-\delta/2}^{\delta/2} x^2 dx = \frac{\delta^2}{12}, \tag{5}$$

where $\delta = \frac{W_{\max} - W_{\min}}{2^{b_i} - 1}$.

Substituting the above relation into the Fisher-weighted objective, where the expected quantization error is determined by the bit-width b_i, yields the following bit allocation objective:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} \frac{\mathcal{F}_{ii}}{(2^{b_i} - 1)^2} \\ \text{s.t.} \quad & \sum_{k=1}^{q} \alpha_k = 1, \sum_{k=1}^{q} \alpha_k b_k = b. \end{aligned} \tag{6}$$

Here, we present *sorting-based* solutions for different sizes of the candidate bit-width set $\mathcal{B}$, avoiding iterative optimization.

Case 1: $|\mathcal{B}| = 2$**.** When $\mathcal{B} = \{b_1, b_2\}$, the coefficients α_1 and α_2 are uniquely determined by the constraints. The problem reduces to a special case of the 0-1 knapsack problem. Since assigning higher bit-widths to weights with larger $\mathcal{F}_{ii}$ always yields lower loss, the optimal solution is obtained by sorting $\mathcal{F}_{ii}$ in ascending order, assigning the smallest α_1 portion to b_1, and the remaining α_2 to b_2.

Case 2: $|\mathcal{B}| = 3$**.** When $\mathcal{B} = \{b_1, b_2, b_3\}$, we adopt a simple one-dimensional grid search over α_1, as summarized in Alg. 1. Once α_1 is fixed, α_2 and α_3 are uniquely determined, reducing the problem to the **Case 1**. In practice, a grid step size Δ of 0.01 achieves a good trade-off between efficiency and performance. A further analysis of Δ is provided in section 4.2

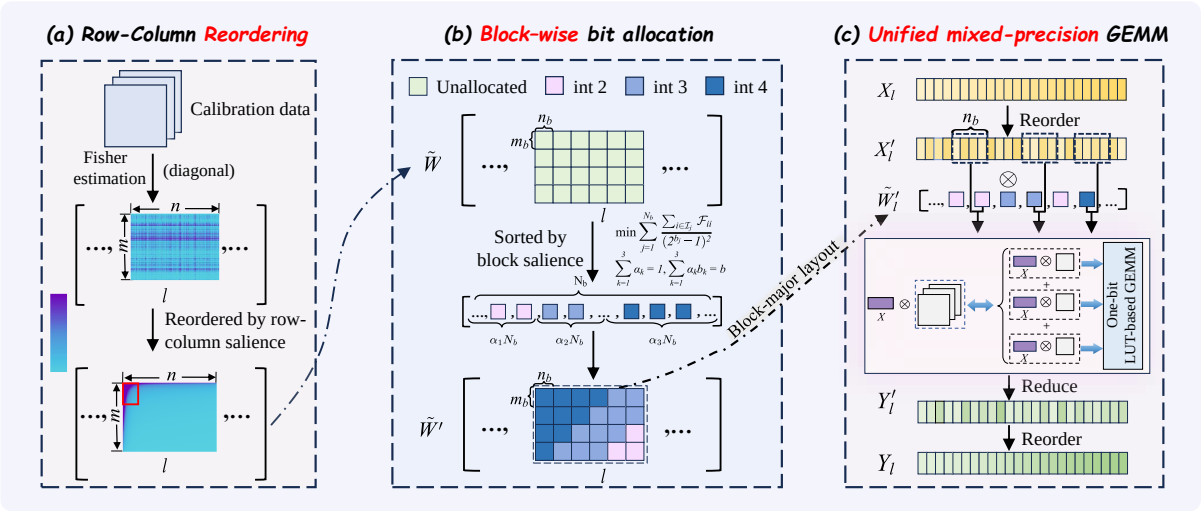


Figure 3: Pipeline of SFMP. a) Row-column reordering to aggregate salient weights; b) Block-wise mixed-precision allocation; c) Activation reordering followed by unified mixed-precision GEMM for matrix multiplication.

Algorithm 1 Element-wise Mixed-Precision Allocation ($|\mathcal{B}| = 3$)

Input: $\{\mathcal{F}_{ii}\}_{i=1}^{N}$, target bit-width b, candidate bits $\{b_1, b_2, b_3\}$, step size Δ
Output: bit assignment $\{b_i\}_{i=1}^{N}$
Sort indices of $\{\mathcal{F}_{ii}\}$ in ascending order
for $\alpha_1 = 0$ to 1 with step Δ **do**

$$\alpha_2 = \frac{b - b_3 - (b_1 - b_3)\alpha_1}{b_2 - b_3}$$

$$\alpha_3 = 1 - \alpha_1 - \alpha_2$$

$$n_k = \lfloor \alpha_k N \rfloor, \quad k \in \{1, 2, 3\}$$

$$b_i = \begin{cases} b_1, & i \le n_1 \\ b_2, & n_1 < i \le n_1 + n_2 \\ b_3, & \text{otherwise} \end{cases}$$

$$\mathcal{L} = \sum_{i=1}^{N} \frac{\mathcal{F}_{ii}}{(2^{b_i} - 1)^2}$$

end for
Return assignment with minimum $\mathcal{L}$

Case 3: $|\mathcal{B}| > 3$**.** In principle, the **Case 2** can be extended to multi-dimensional grid search. However, the search complexity grows exponentially with $|\mathcal{B}|$, while the performance gain is marginal. As shown in Appendix G.4, increasing the number of candidate bit-widths beyond three brings negligible improvement compared to the $|\mathcal{B}| = 3$ setting.

3.3 Block-Wise Mixed-Precision

Although the element-wise bit allocation strategy in Alg. 1 is theoretically accurate and simple to implement, it is inefficient for hardware execution. Weights assigned the same bit-width are scattered in a highly irregular spatial pattern within the weight matrix, hindering structured quantization schemes such as group quantization.

To balance quantization performance and hardware efficiency, we adopt a coarser yet structured block-wise bit allocation strategy, where bit-widths are assigned at the block level rather than to individual weights. Specifically, as shown in Fig. 3(b), each weight matrix is partitioned into non-overlapping two-dimensional blocks of size $m_b \times n_b$. Leveraging the additive structure of the objective in Eq. 6, we reformulate it as:

$$\min \sum_{j=1}^{N_b} \frac{\sum_{i \in \mathcal{I}_j} \mathcal{F}_{ii}}{(2^{b_j} - 1)^2} \tag{7}$$

where $\mathcal{I}_j$ denotes the index set of weights in the j-th block, and N_b is the total number of blocks. Treating each block as the minimal unit for precision allocation, the strategy proposed in Section 3.2 can be directly applied. The resulting block-wise scheme exhibits regular and hardware-friendly bit patterns. In practice, the block dimensions (m_b, n_b) are chosen to balance fine-grained granularity and hardware characteristics. For example, on GPUs, we adopt block sizes such as $(256, 128)$ or $(512, 128)$ to match common GEMM tiling strategies and warp-level parallelism in CUDA.

3.4 Row-Column Weight Reordering

As shown in Fig. 3(a), it can be observed that the spatial distribution of the diagonal values of the Fisher information matrix exhibits strong structure along rows or columns. This distributional property has been exploited by prior works (Huang et al., 2024, 2025) to guide quantization, for example, by

reordering weights column-wise. In our work, our block-wise bit allocation is spatially misaligned with this distribution. To address this mismatch, we further propose a bidirectional reordering strategy that reorganizes the weight matrix based on both row-wise and column-wise aggregated Fisher diagonal values.

Given a weight matrix $W_l \in \mathbb{R}^{m\times n}$, we define row- and column-wise salience by aggregating the diagonal entries of the Fisher information matrix:

$$s_{l,\text{row}} = \sum_{i\in\mathcal{R}_l} \mathcal{F}_{ii}, \qquad s_{l,\text{col}} = \sum_{i\in\mathcal{C}_l} \mathcal{F}_{ii}, \quad (8)$$

where $\mathcal{R}_l$ and $\mathcal{C}_l$ denote the index sets of weights in each row and column of W_l, respectively.

Row and column permutations are obtained by sorting these aggregated Fisher values in descending order: $p_{l,\text{row}} = \text{argsort}(s_{l,\text{row}}), p_{l,\text{col}} = \text{argsort}(s_{l,\text{col}})$. The corresponding permutation matrices are denoted by $P_{l,\text{row}}$ and $P_{l,\text{col}}$. The reordered weight matrix is given by:

$$\tilde{W}_l = P_{l,\text{row}} W_l P_{l,\text{col}}. \quad (9)$$

As shown in Fig. 3(a), the bidirectional reordering strategy spatially aggregate high-salience weights, improving the alignment between weight salience and block-wise bit allocation. Notably, the reordering is performed offline and incurs no runtime overhead. During inference, as shown in Fig. 3(c), the same permutation can be equivalently applied to the activation, with negligible cost compared to GEMM computation.

3.5 Unified Mixed-Precision GEMM Kernel

For our proposed block-wise mixed-precision format, adopting dequant-based operators introduces two major challenges: 1) conventional row-major or column-major storage layouts complicate weight packing and unpacking, as the block structure is not explicitly represented in memory. 2) mixed-precision formats require additional control-flow branching and multiple kernel variants in compute kernels (e.g., CUDA kernels), increasing implementation complexity.

As shown in Fig. 3(c), to address challenge 1), we propose a block-major representation, where the quantized weight matrix is partitioned into blocks and organized in a block-major layout, enabling contiguous memory access within each block. To address challenge 2), we employ a unified GEMM kernel that processes all blocks regardless of their bit-widths. Specifically, each block is decomposed into one-bit components and computed via one-bit LUT-based GEMM, thereby eliminating explicit weight dequantization and precision-specific execution paths (see Appendix F for detailed CUDA implementation).

A potential concern is that heterogeneous bit-widths may introduce load imbalance, with high-precision blocks dominating the overall latency. However, this is avoided in our design. Our kernel typically tiles the computation into thread blocks of size $[M_{\text{tile}}, K_{\text{tile}}]$, e.g., $[256, 128]$. For a 4096×4096 matrix (e.g., the q_{proj} in LLaMA3.1-8B), this results in 512 thread blocks. From a hardware perspective, modern GPUs execute thread blocks via dynamic scheduling over Streaming Multiprocessors (SMs). Each thread block runs to completion on a single SM, while idle SMs continuously fetch new blocks from a global work queue. In our setting, the number of thread blocks is much larger than the number of SMs (e.g., 82 on RTX 3090 and 108 on A100), so each SM processes multiple blocks sequentially. In a multi-SM parallel setting, higher-precision blocks mainly affect the tail of overall execution, which constitutes only a small fraction of the total runtime. The overall latency is determined by the **aggregate workload**, rather than the highest-precision blocks.

4 Experiments

We evaluate our method on several state-of-the-art pretrained models, including LLaMA3.1 8B and 70B (Grattafiori et al., 2024), Qwen3 8B, 14B and 32B (Yang et al., 2025). The diagonal values of the Fisher Information Matrix are estimated using 1k samples from C4 (Raffel et al., 2020). The block shape (m_b, n_b) is (512, 128), where group quantization is applied along the n_b dimension within each block. The candidate bit-width set is $\{2, 3, 4\}$. To ensure fair comparison, following AMQ (Lee et al., 2025), we avoid introducing complex tricks and just adopt AWQ (Lin et al., 2024) as our quantization method. We compare our method against fixed-precision methods such as GPTQ and AWQ, element-wise mixed-precision method SqueezeLLM (Kim et al., 2024), group-wise mixed-precision method SliM-LLM (Huang et al., 2025), any-size methods such as BitStack (Wang et al., 2025) and AMQ. Our method is orthogonal to most quantization tricks, in Appendix I, we further combine our approach with

Quantization-Aware Training (QAT) method (Chen et al., 2024a). All experiments are conducted on A100-80GB GPU.

We evaluate our method from multiple perspectives. For language modeling, we report perplexity on C4 and WikiText2 (Merity et al., 2017). For zero-shot evaluation, we use the LM Evaluation Harness (Gao et al., 2021) to evaluate six tasks, including ARC-Challenge, ARC-Easy (Clark et al., 2018), PIQA (Bisk et al., 2019), HellaSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019), and WinoGrande (Sakaguchi et al., 2021). We further evaluate 5-shot performance on MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021). For inference performance, considering edge deployment scenarios, we report both kernel-level latency and end-to-end inference speed (tokens / s) when generating 128 tokens with batch size 1.

4.1 Main Results

SFMP vs. Any-Size Methods. Table 1 reports the model perplexity and zero-shot task accuracy under different memory budgets for models quantized with SFMP, AMQ, and BitStack. We report results under different **BPW** (bits per weight) settings. Across multiple model scales, SFMP consistently outperforms AMQ. The advantage of SFMP becomes more pronounced at extremely low precision (e.g., BPW=2.5), where it achieves the best average zero-shot accuracy among all methods. At BPW=3.5, SFMP retains 98.90% of the average zero-shot performance of the BF16 LLaMA3.1-70B model. Moreover, Table 2 shows that on the 5-shot MMLU and GSM8K benchmarks, SFMP consistently outperforms AMQ across all model sizes. These results demonstrate that SFMP remains robust on challenging tasks and show that our strategy is more stable and effective than layer-wise mixed-precision methods, even without any search or optimization.

SFMP vs. Fixed-Precision Methods. We compare SFMP with GPTQ, AWQ and SliM-LLM, which apply a uniform bit-width across all weight matrices. Table 3 reports perplexity and average zero-shot accuracy on the BPW ranging from 2.25 to 4. Across all settings, SFMP consistently outperforms fixed-precision methods. Moreover, SFMP consistently outperforms the group-wise mixed-precision method SliM-LLM and provides greater flexibility in precision allocation, as SliM-LLM enforces fixed average bit-width across all weight matrices.

Model	Mem. (MB)	BPW	Method	Wiki(↓)	C4(↓)	Avg.(↑)
8B	15,317	16	BF16	6.15	8.89	75.01
	4,085	2.5	BitStack	23.28	38.23	58.19
			AMQ	17.85	24.01	58.65
			SFMP	**13.68**	**17.77**	**65.97**
	4,501	3.0	BitStack	12.55	20.47	64.40
			AMQ	9.38	13.05	68.78
			SFMP	**8.65**	**12.04**	**69.92**
	4,917	3.5	BitStack	9.47	15.29	68.59
			AMQ	7.39	10.54	72.56
			SFMP	**7.30**	**10.38**	**73.43**
	5,333	4.0	BitStack	8.39	13.47	70.95
			AMQ	6.86	9.79	73.46
			SFMP	**6.84**	**9.74**	**74.15**
70B	134,571	16	BF16	2.81	7.11	80.96
	24,411	2.5	BitStack	7.55	12.92	74.51
			AMQ	7.62	12.14	74.33
			SFMP	**7.24**	**10.07**	**74.60**
	28,491	3.0	BitStack	6.38	11.21	76.30
			AMQ	5.84	9.47	77.80
			SFMP	**5.31**	**8.36**	**78.07**
	32,571	3.5	BitStack	5.44	9.52	78.24
			AMQ	4.26	8.20	79.11
			SFMP	**4.00**	**7.33**	**80.07**
	36,651	4.0	BitStack	4.98	8.92	79.17
			AMQ	3.49	7.61	80.14
			SFMP	**3.37**	**7.01**	**80.47**

Table 1: Evaluation of Llama 3.1 8B/70B models compressed by SFMP, BitStack and AMQ at the BPW of 2.5, 3.0, 3.5 and 4.0, using a groupsize of 128, showing WikiText-2 and C4 dataset perplexity (PPL) alongside zero-shot tasks average accuracy. **BPW** denotes *"bits per weight"*. Quantization scales and zero-points are stored in BF16. Detailed zero-shot accurcy is provided in Table 18.

Inference Performance. We evaluate the inference performance of SFMP across a wide range of hardware platforms. As shown in Fig. 4, fixed-precision uniform quantization framework GPTQ-Model [1] becomes slower as the BPW decreases. This counterintuitive behavior is caused by the increasing weight unpacking and dequantization overhead at low bit-width. In contrast, SFMP exhibits an increase in inference speed as the BPW reduces. This advantage stems from its one-bit LUT-based GEMM formulation, where the computational latency of the kernel scales approximately linearly with the BPW (see Fig. 7). Moreover, the decomposition-based compression method BitStack suffers from repeated weight reconstruction during inference, leading to substantially worse performance, even compared to BF16.

4.2 Analysis and Ablation Study

More analysis and ablation study can be found in appendix G.

[1] https://github.com/ModelCloud/GPTQModel

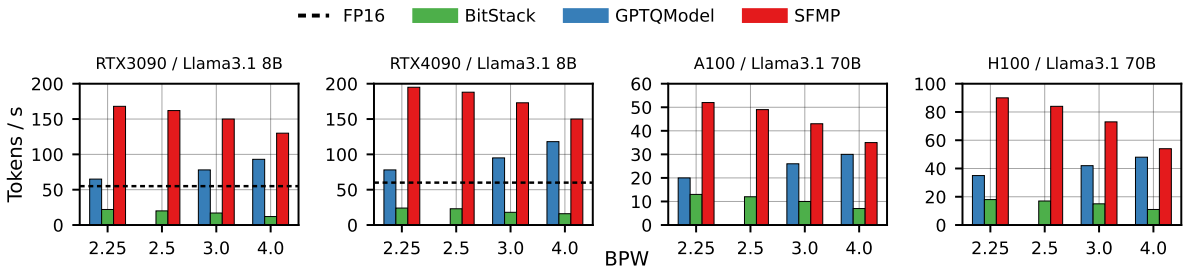


Figure 4: End-to-end throughput (tokens/s) of generating a sequence length of 128 with batchsize of 1. BF16 inference of LLaMA3.1 70B is not feasible on single A100 and H100 due to memory constraints. GPTQModel uses Marlin (Frantar et al., 2025) for BPW=4, and Triton (Tillet et al., 2019) for BPW=2 and 3. At BPW=4, we provide comparisons with TensorRT-LLM (NVIDIA, 2026) and ExLlama (Turboderp, 2023) in Table 14.

Model	Mem. (MB)	BPW	Method	MMLU	GSM8K
Qwen3 8B	15,623	16	BF16	74.88	87.19
	4,445	2.5	AMQ	50.78	15.09
			SFMP	**57.33**	**49.20**
	4,859	3.0	AMQ	65.41	73.66
			SFMP	**66.03**	**74.34**
	5,273	3.5	AMQ	71.71	83.32
			SFMP	**72.11**	**84.46**
	5,687	4.0	AMQ	73.44	85.75
			SFMP	**73.60**	**87.17**
Qwen3 14B	28,169	16	BF16	78.78	88.19
	6,906	2.5	AMQ	56.18	48.76
			SFMP	**62.55**	**59.14**
	7,694	3.0	AMQ	71.89	79.16
			SFMP	**73.74**	**85.22**
	8,481	3.5	AMQ	75.87	84.48
			SFMP	**77.00**	**86.10**
	9,269	4.0	AMQ	76.89	86.58
			SFMP	**78.14**	**87.41**
Qwen3 32B	62,490	16	BF16	81.28	85.05
	12,270	2.5	AMQ	64.37	59.38
			SFMP	**73.62**	**68.08**
	14,130	3.0	AMQ	73.89	67.11
			SFMP	**77.68**	**72.33**
	15,990	3.5	AMQ	77.12	77.96
			SFMP	**79.30**	**79.30**
	17,850	4.0	AMQ	80.19	79.83
			SFMP	**81.03**	**81.04**

Table 2: 5-shot MMLU, GSM8K task results over Qwen3 family. PPL and zero-shot accuracy can be found in Table 20.

Model	Mem. (MB)	BPW	Method	Wiki(↓)	C4(↓)	Avg.(↑)
8B	15,317	16	BF16	6.15	8.89	75.01
	3,877	2.25	GPTQ_{w2g128}	232	165	38.55
			AWQ_{w2g128}	1.57e6	1.86e6	35.80
			SliM-LLM_{g128}	193	142	40.67
	3,961	2.35	$\textbf{SFMP}_{g128}$	**24.57**	**28.92**	**60.80**
	4,501	3.0	GPTQ_{w3}	22.13	25.05	55.83
			AWQ_{w3}	16.06	19.79	64.61
			SqueezeLLM_{w3}	13.43	15.64	65.78
			$\textbf{SFMP}_{g128}$	**8.65**	**12.04**	**69.92**
	4,709	3.25	GPTQ_{w3g128}	8.28	11.49	69.22
			AWQ_{w3g128}	8.23	11.58	70.72
			SliM-LLM_{g128}	8.17	11.25	70.31
			$\text{SqueezeLLM}_{0.45\%}$	7.95	11.39	70.97
			$\textbf{SFMP}_{g128}$	**7.78**	**10.97**	**72.47**
	5,333	4.0	GPTQ_{w4}	7.5	10.38	71.46
			AWQ_{w4}	7.23	10.26	73.60
			SqueezeLLM_{w4}	7.17	10.11	73.17
			$\textbf{SFMP}_{g128}$	**6.84**	**9.74**	**74.15**
70B	134,571	16	BF16	2.81	7.11	80.96
	22,371	2.25	GPTQ_{w2g128}	113.22	131.9	40.02
			AWQ_{w2g128}	1.8e6	1.5e6	40.65
			SliM-LLM_{g128}	68.84	88.36	46.51
	23,187	2.35	$\textbf{SFMP}_{g128}$	**8.17**	**11.42**	**72.65**
	28,491	3.0	GPTQ_{w3}	11.27	12.19	66.27
			AWQ_{w3}	10.86	11.74	68.84
			SqueezeLLM_{w3}	10.17	10.62	69.18
			$\textbf{SFMP}_{g128}$	**5.31**	**8.36**	**78.07**
	30,531	3.25	GPTQ_{w3g128}	5.17	8.76	72.82
			AWQ_{w3g128}	4.78	8.57	75.18
			SliM-LLM_{g128}	4.74	8.52	77.41
			$\text{SqueezeLLM}_{0.45\%}$	4.71	8.48	75.16
			$\textbf{SFMP}_{g128}$	**4.33**	**7.56**	**79.38**
	36,651	4.0	GPTQ_{w4}	4.58	8.42	74.88
			AWQ_{w4}	4.18	8.29	75.95
			SqueezeLLM_{w4}	4.19	8.28	77.23
			$\textbf{SFMP}_{g128}$	**3.37**	**7.01**	**80.47**

Table 3: Evaluation of Llama3.1 8B/70B models on WikiText-2, C4 perplexity (PPL), and zero-shot tasks. Memory overhead from extra quantization parameters in GPTQ and AWQ at w3, w4 is omitted as it is negligible. SliM-LLM only supports group-wise quantization. Detailed zero-shot accuracy is provided in Table 19.

Extra Metadata for blocks and reorder. SFMP requires storing additional metadata to index block-wise precision and activation reordering vectors. For a general weight matrix of size $[M, N]$, using a block size of $[512, 128]$, the matrix is partitioned into $\frac{M}{512} \times \frac{N}{128}$ blocks. Each block requires one int8 value to store its bit-width. In addition, we store two activation reordering vectors of sizes $[1, M]$ and $[1, N]$, both in fp16. Therefore, the average metadata overhead (in BPW) for a L-layers model can be written as:

$$\frac{\sum_{l=1}^{L} \left(\frac{M_l}{512} \frac{N_l}{128} \text{ sizeof(int8)} + (M_l + N_l) \text{ sizeof(fp16)} \right)}{\sum_{l=1}^{L} M_l N_l}$$

In practice, taking LLaMA 3.1 8B as an example. The overhead is **0.006BPW**, which is negligible.

Impact of row-column reordering. We analyze the contribution of row-column reordering through an ablation study with four settings: no reorder, column reorder, row reorder, and combined

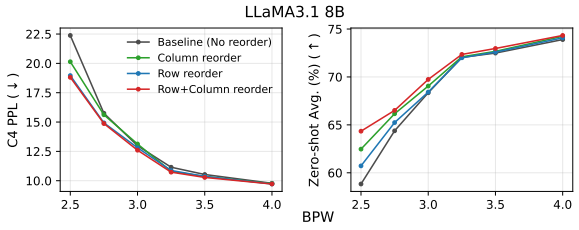


Figure 5: Impact of row and column reordering across different average bits on model perplexity (↓) and zero-shot accuracy (↑).

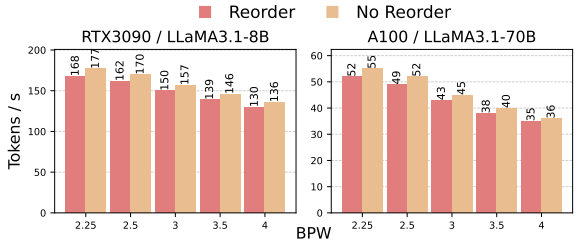


Figure 6: Throughput (tokens / s) comparison for end-to-end generation of 128 tokens with and without reordering.

row–column reorder. As shown in Fig. 5, two key observations can be drawn: 1) Column reordering usually outperforms row reordering. This may be because it is better aligned with the activation-aware principle of AWQ, thereby more effectively protecting important input channels during quantization. 2) The performance gains from the row and column reordering gradually diminish as the BPW increases. At BPW = 4, all four configurations achieve nearly identical accuracy. This convergence is due to AWQ already achieving near-lossless compression at 4-bit precision, which leaves little room for further improvement from reordering. Furthermore, Fig. 6 shows the impact of reordering on end-to-end inference speed. Reordering incurs a modest slowdown of at most 5%, with a smaller impact on larger models, as the increasing cost of GEMM amortizes the fixed overhead introduced by reordering.

Kernel evaluation. Fig. 7 compares GEMV latency under three settings: FP16 (cuBLAS), the uniform quantization kernel from GPTQModel, and our unified kernel. For the uniform quantization kernel, we adopt the backend selected by GPTQModel's automatic tuning. Specifically, at BPW=4, GPTQModel employs the state-of-the-art W4A16 kernel, Marlin (Frantar et al., 2025). At BPW = 2 and 3, existing works provide limited kernel support and no widely adopted high-performance implementations, so GPTQModel uses a Triton-based implementation (Tillet et al., 2019). We select two

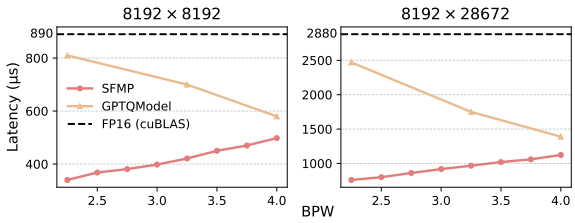


Figure 7: Latency comparison of our unified mixed-precision kernel , uniform quantization kernel from GPTQModel and cuBLAS FP16 kernel on A100.

representative GEMV operations from LLaMA3.1-70B: q_proj and down_proj. Across all shapes and bits, our kernel consistently achieves lower latency than both cuBLAS and GPTQModel. Notably, the latency of our kernel decreases approximately linearly with reducing BPW. This trend is attributed to eliminating weight dequantization overhead and leveraging LUT-based computation. In contrast, the latency of GPTQModel's kernel increases with lower BPW due to the growing overhead of weight unpacking.

Search cost. Table 6 compares the algorithmic cost of searching mixed-precision configurations for BitStack, AMQ, SliM-LLM and SFMP in terms of memory and time, evaluated on A100-80GB GPUs. BitStack incurs substantial overhead due to weight decomposition and block-wise sorting. AMQ reduces the search cost via proxy-based predictors and pruning, reducing the time to 44 hours for LLaMA3.1-70B. SliM-LLM can be executed with fewer GPUs, but configuring LLaMA3.1-70B still takes 8 hours. In contrast, SFMP directly assigns bit-widths by solving a special knapsack problem. Its main cost is estimating the diagonal values of the Fisher Information Matrix using a small calibration set, resulting in minimal algorithmic overhead.

Analysis of Step Size Δ. We conduct an ablation study on the one-dimensional grid search step size Δ to evaluate its impact on model performance, with results summarized in table 4. We observe that a step size $\Delta = 0.01$ already achieves strong performance. Further reducing the step size below 0.01 yields only marginal improvements. Therefore, we adopt $\Delta = 0.01$ in our method.

Comparison with LUT-based GEMM Family. We further compare SFMP with a representative LUT-based GEMM baseline, AnyBCQ (Park et al., 2025a) on A100. As both approaches are based on LUT-based computation, this comparison provides a more direct assessment of kernel efficiency

Model	BPW	Δ			
		0.005	**0.01**	0.02	0.04
LLaMA3.1 8B	2.35	(24.55, 60.91)	(24.57, 60.80)	(24.68, 60.51)	(24.82, 60.51)
	2.5	(13.68, 65.92)	(13.68, 65.97)	(13.75, 65.23)	(13.94, 64.89)
	3	(8.64, 69.97)	(8.65, 69.92)	(8.67, 69.75)	(8.74, 69.26)
	3.25	(7.78, 72.56)	(7.78, 72.47)	(7.80, 72.10)	(7.85, 71.78)
	3.5	(7.30, 73.56)	(7.30, 73.43)	(7.31, 73.34)	(7.33, 73.12)
	4	(6.84, 74.22)	(6.84, 74.15)	(6.84, 74.08)	(6.84, 73.82)

Table 4: Ablation on grid search step size Δ. Each entry reports (WikiText2 perplexity (↓), Zero-shot accuracy (%) (↑)). Result shows that $\Delta = 0.01$ achieves strong performance, while smaller step sizes bring negligible gains.

within the same optimization family. As shown in table 5, SFMP achieves comparable kernel latency to AnyBCQ across different bit-widths and matrix sizes, with the performance gap consistently within 3%. This small latency gap is mainly attributed to the additional runtime computation in SFMP required to determine the actual memory offsets of blocks in the flattened weight layout. The experiment indicates that SFMP preserves the efficiency of LUT-based GEMM implementations while introducing a more flexible block-wise quantization scheme.

Bits	(M, K)	AnyBCQ (ms)	SFMP (ms)
2	(8192,8192)	332	340
	(28672,8192)	755	760
3	(8192,8192)	410	421
	(28672,8192)	942	967
4	(8192,8192)	478	495
	(28672,8192)	1076	1102

Table 5: Kernel latency comparison (ms) with the LUT-based GEMM baseline (AnyBCQ).

Model	8B		70B	
Parameter	#GPU	Cost (h)	#GPU	Cost (h)
SliM-LLM	1	2	1	8
BitStack	1	12	4	>300
AMQ	1	7	4	44
SFMP	1	0.15	4	0.50

Table 6: The search time on Llama 3.1 family of SFMP, BitStack, AMQ, SliM-LLM.

Bit allocation visualization. Fig. 8 shows the bit allocation result on LLaMA3.1-8B. We provide detailed average bit-widths for each linear layer in Table 17. It can be observed that the Value projection in self-attention consistently retains the higher bit-widths, followed by the Gate, Up, and Down

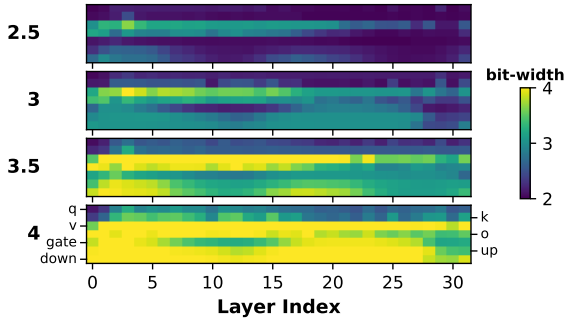


Figure 8: Visualization of bit allocation over linear layers with different average bits at Llama3.1 8B. The numbers on the left indicate the average bit.

layers, with Query and Key projections assigned the lower bit-widths. This patter is consistent with prior findings from AMQ (Lee et al., 2025), validating the effectiveness of SFMP's bit allocation scheme.

5 Conclusion

SFMP formulates a fine-grained mixed-precision quantization problem based on a Taylor expansion of the model's final output, simplifying the optimization objective and avoiding complex iterative search or black-box optimization. To maintain hardware friendliness, SFMP adopts a structured block-wise pattern, slightly sacrificing accuracy in exchange for regular memory layouts and efficient execution, and introduces a unified computation kernel. Overall, SFMP provides a practical solution for deploying large language models in resource-constrained environments.

Limitations

Despite the effectiveness of the proposed method, several limitations remain and point to promising directions for future work. First, our current implementation and evaluation focus on GPU-based inference. Supporting additional hardware platforms such as CPUs, NPUs, and TPUs, would significantly broaden the applicability of our method. Second, current work focuses on weight-only quantization. Extending the optimization objective to include activation quantization would further improve inference efficiency, particularly in compute-bound scenarios. Third, as discussed in the Appendix G.1, the group size plays a critical role in determining model accuracy under a fixed memory budget. However, existing mixed-precision quantization methods typically treat the group size as a fixed hyperparameter (e.g., 128), determined heuristically, and is independent of the bit allocation strategy. A promising future direction is to incorporate the group size into the mixed-precision optimization process and allow flexible, adaptive group sizes, which may further improve model performance.

Ethical Considerations

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Use of AI Assistance

AI assistants were used solely for academic writing support, including language polishing, sentence refinement, and grammatical revision of the manuscript. They were not involved in the conception of research ideas, algorithm development, experimental design, or result analysis. All core contributions, technical content, and conclusions were independently developed by the authors.

References

Eirikur Agustsson and Lucas Theis. 2020. Universally quantized neural compression. *Advances in neural information processing systems*, 33:12367–12376.

Richard Bellman. 1966. Dynamic programming. *science*, 153(3731):34–37.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language. *arXiv e-prints*, arXiv:1911.11641.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2024a. Efficientqat: Efficient quantization-aware training for large language models. *arXiv preprint arXiv:2407.11062*.

Zihan Chen, Bike Xie, Jundong Li, and Cong Shen. 2024b. Channel-wise mixed-precision quantization for large language models. *arXiv preprint arXiv:2410.13056*.

Wenhua Cheng, Weiwei Zhang, Heng Guo, and Haihao Shen. 2025. Signroundv2: Closing the performance gap in extremely low-bit post-training quantization for llms. *arXiv preprint arXiv:2512.04746*.

Nihat Mert Cicek, Xipeng Shen, and Ozcan Ozturk. 2022. Energy efficient boosting of gemm accelerators for dnn via reuse. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(5):1–26.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv e-prints*, arXiv:2110.14168.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. 2025. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 239–251.

Darshan C Ganji, Saad Ashfaq, Ehsan Saboori, Sudhakar Sah, Saptarshi Mitra, Mohammadhossein

Askarihemmat, Alexander Hoffman, Ahmed Hassanien, and Mathieu Leonardon. 2023. Deepgemm: Accelerated ultra low-precision inference on cpu architectures using lookup tables. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4656–4664.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 181 others. 2024. The Llama 3 Herd of Models. *arXiv e-prints*, arXiv:2407.21783.

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

Dorit S Hochba. 1997. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52.

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

Minoo Hosseinzadeh and Hana Khamfroush. 2025. Dilemma: Joint llm quantization and distributed llm inference over edge computing systems. *arXiv preprint arXiv:2503.01704*.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024. Billm: pushing the limit of post-training quantization for llms. In *Proceedings of the 41st International Conference on Machine Learning*.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and XIAOJUAN QI. 2025. Slim-LLM: Salience-driven mixed-precision quantization for large language models. In *Forty-second International Conference on Machine Learning*.

Erik Johannes Husom, Arda Goknil, Merve Astekin, Lwin Khin Shar, Andre KÃ¥ sen, Sagar Sen, Benedikt Andreas Mithassel, and Ahmet Soylu. 2025. Sustainable llm inference for edge ai: Evaluating quantized llms for energy efficiency, output accuracy, and inference latency. *ACM Transactions on Internet of Things*, 6(4):1–35.

Wonsuk Jang and Thierry Tambe. 2025. Blockdialect: Block-wise fine-grained mixed format quantization for energy-efficient LLM inference. In *Forty-second International Conference on Machine Learning*.

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

Hyeoncheol Kim, Taehoon Kim, Taehyeong Park, Donghyeon Kim, Yongseung Yu, Hanjun Kim, and Yongjun Park. 2025. Accelerating llms using an efficient gemm library and target-aware optimizations on real-world pim devices. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization*, pages 225–240.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. 2024. Squeezellm: dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science*, (4598):671–680.

Sangjun Lee, Seung-taek Woo, Jun-gyu Jin, Changhun Lee, and Eunhyeok Park. 2025. Amq: Enabling automl for mixed-precision weight-only quantization of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35520–35538.

Jinhao Li, Jiaming Xu, Shiyao Li, Shan Huang, Jun Liu, Yaoxiu Lian, and Guohao Dai. 2024. Fast and efficient 2-bit llm inference on gpu: 2/4/16-bit in a weight matrix with asynchronous dequantization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9.

Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. 2023. Llm-mq: Mixed-precision quantization for efficient llm deployment. In *The Efficient Natural Language and Speech Processing Workshop with NeurIPS*, volume 9, page 3.

Xinlin Li, Timothy Chou, Josh Fromm, Zichang Liu, Yunjie Pan, and Christina Fragouli. 2026. Scalebits: Scalable bitwidth search for hardware-aligned mixed-precision llms. *arXiv preprint arXiv:2602.17698*.

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

Fangxin Liu, Zongwu Wang, Jinhong Xia, Junping Zhao, Shouren Zhao, Jinjin Li, Jian Liu, Li Jiang, and Haibing Guan. 2025a. FlexQuant: A flexible and efficient dynamic precision switching framework for LLM quantization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 4152–4161.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen

Blankevoort. 2025b. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, Lin Xiao, Yuandong Tian, Bilge Soran, Raghuraman Krishnamoorthi, Tijmen Blankevoort, and Vikas Chandra. 2025c. Paretoq: Improving scaling laws in extremely low-bit LLM quantization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers. 2017. A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

NVIDIA. 2026. Tensorrt-llm. https://github.com/NVIDIA/TensorRT-LLM. Accessed: 2026-04-16.

Gunho Park, Jeongin Bae, Beomseok Kwon, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. 2025a. Anybcq: Hardware efficient flexible binary-coded quantization for multi-precision llms. *arXiv preprint arXiv:2510.10467*.

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

Gunho Park, Baeseong park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. In *The Twelfth International Conference on Learning Representations*.

Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70. IEEE.

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

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

Chen Tang, Kai Ouyang, Zhi Wang, Yifei Zhu, Wen Ji, Yaowei Wang, and Wenwu Zhu. 2022. Mixed-precision neural network quantization via learned layer-wise importance. In *European conference on computer vision*, pages 259–275. Springer.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19.

Turboderp. 2023. Exllama. https://github.com/turboderp/exllama. Accessed: 2026-04-16.

Jinguang Wang, Yuexi Yin, Haifeng Sun, Qi Qi, Jingyu Wang, Zirui Zhuang, Tingting Yang, and Jianxin Liao. 2024. Outliertune: Efficient channel-wise quantization for large language models. *arXiv preprint arXiv:2406.18832*.

Xinghao Wang, Pengyu Wang, Bo Wang, Dong Zhang, Yunhua Zhou, and Xipeng Qiu. 2025. Bitstack: Any-size compression of large language models in variable memory environments. In *The Thirteenth International Conference on Learning Representations*.

Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. 2025. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. In *Proceedings of the Twentieth European Conference on Computer Systems*, page 278–292.

Laurence A Wolsey. 2020. *Integer programming*. John Wiley & Sons.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 Technical Report. *arXiv e-prints*, arXiv:2505.09388.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Xinyuan Zhang, Jiang Liu, Zehui Xiong, Yudong Huang, Gaochang Xie, and Ran Zhang. 2024. Edge intelligence optimization for large language model inference with batching and quantization. In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE.

Pengxiang Zhao and Xiaoming Yuan. 2025. GANQ: GPU-adaptive non-uniform quantization for large language models. In *Forty-second International Conference on Machine Learning*.

Appendix

Appendix Overview

Appendix A: Additional Related Works
Appendix B: Details about One-Bit Lut-Based GEMM
Appendix C: Fisher-Information-Based Analysis of Quantized Weight
Appendix D: Empirical Study on Inference Speed of Group-wise Mixed-Precision Methods
Appendix E: Empirical Analysis of Fisher Diagonal Value Distribution
Appendix F: Detailed CUDA Implementation
Appendix G: Additional Ablation Analysis
Appendix H: Discussion of Related Concurrent Work
Appendix I: SFMP with Quantization-Aware Training
Appendix J: Autoregressive Decoding Comparison Between SFMP and AMQ
Appendix K: More Results of Bit Allocation Visualizations

A Additional Related Works

A.1 Structured and Unstructured Quantization Format

Structured quantization formats are generally more favorable for hardware execution. For example, assigning a uniform integer precision to an entire linear layer enables regular memory access patterns and allows weights to be dequantized in a uniform manner, without introducing conditional branches or complex control flow. In large-scale models where such weight matrices appear extensively, this structured design is particularly advantageous for hardware acceleration (Frantar et al., 2023; Lin et al., 2024; Chen et al., 2024a; Lee et al., 2025; Jang and Tambe, 2025). In contrast, unstructured quantization formats typically offer finer granularity and greater flexibility. They allow the quantization precision to be adaptively adjusted according to the characteristics of individual weights or channels, and thus can achieve higher model accuracy under the same memory budget compared to structured quantization (Huang et al., 2025; Jang and Tambe, 2025; Li et al., 2023; Kim et al., 2024; Zhao and Yuan, 2025). However, this increased flexibility often comes at the cost of irregular memory access patterns and more complex dequantization procedures. As a result, unstructured quantization is generally less efficient in terms of inference latency

and hardware utilization than structured quantization, especially on general-purpose accelerators.

A.2 Layer-Wise Mixed-Precision

Layer-wise mixed-precision quantization assigns different bit-widths to individual linear layers and typically formulates bit allocation as an integer programming or multi-objective optimization problem under a memory budget. However, this problem is NP-complete. For large-scale models, the search space becomes prohibitively large: LLaMA3.1 8B contains 224 linear layers, leading to a search space of 2^{224} even with only two candidate bit-widths, while LLaMA3.1 70B expands this space to 2^{560}. To obtain acceptable solutions within a reasonable time, existing mixed-precision methods rely on heuristic strategies to reduce the search space. Most approaches (Cheng et al., 2025; You et al., 2024) adopt constrained formulations and solve the resulting integer programs using off-the-shelf solvers, whereas methods such as AMQ (Lee et al., 2025) cast bit allocation as a multi-objective optimization problem and employ genetic algorithms to approximate Pareto-optimal solutions.

B Details about One-Bit Lut-Based GEMM

Fig. 9 illustrates the detailed computational procedure of one-bit Lut-based GEMM. First, a q-bit quantized weight matrix $\mathrm{W}_{int} \in \mathbb{Z}^{m\times n}$ is decomposed into q one-bit matrices $\{W_0, W_1, ..., W_{q-1}\}$, where $W_i \in \{0,1\}^{m\times n}$, representing the respective bit planes of the original weights. For example, for integer values (9, 7, 6, 3) with binary representations (1001, 0111, 0110, 0011), the vector for the lowest bit is (1, 1, 0, 1), and the vector for the highest bit is (1, 0, 0, 0). This decomposition is performed offline, incurring no runtime overhead. During inference, for an activation vector of the group size g, the operator precomputes the dot products between this activation vector and all 2^g possible combinations of one-bit weights, storing the results in the LUT. Thus, the original matrix computation requiring high-precision multiply-accumulate operations is simplified into highly efficient table lookups followed by summation. As shown in the Eq. 10, the matrix multiplication between the activation X and the original quantized weight W_{int} can be transformed into a sum of multiple one-bit GEMM operations:

$$\begin{aligned} X \times W_{int} &= X \times \left(\sum_{i=0}^{q-1} 2^i W_i\right) \\ &= \sum_{i=0}^{q-1} 2^i\, X \times W_i, \\ & W_i \in \{0,1\}^{m\times n} d \end{aligned} \tag{10}$$

To reduce the table size and accelerate table lookup, a commonly used technique is mirror storage. Typically, dequantized weight $\hat{W}$ can be written in the following form:

$$\hat{\mathrm{W}} = \sum_{i=0}^{q-1} 2^i s W_i + z, \;\; W_i \in \{0,1\}^{m\times n}, \tag{11}$$

where $s \in \mathbb{R}$ denotes the scale and $z \in \mathbb{R}$ denotes the zero-point. we apply a simple linear transformation by setting $\hat{s} = \frac{1}{2}s$, $\hat{W}_i = 2W_i - 1$, $\hat{z} = z + \frac{1}{2}\sum_{i=0}^{q-1} 2^i s$. After that, the dequantized weight $\hat{W}$ can be rewritten as:

$$\hat{\mathrm{W}} = \sum_{i=0}^{q-1} 2^i \hat{s} \hat{W}_i + \hat{z}, \;\; \hat{W}_i \in \{-1,1\}^{m\times n}. \tag{12}$$

Under this transformation, for example, with an input activation combinations $[x_1, x_2, x_3, x_4]$, the

output of the dot product has 16 possible outcomes, ranging from $(-x_1 - x_2 - x_3 - x_4,\ ...,\ x_1 + x_2 + x_3 + x_4)$. When storing the lookup table, we only need to store half of the possible results, as the remaining half can be obtained by negating the stored values. This table compression method is lossless, fully preserving model inference accuracy while also reducing memory usage by half and accelerating table access.

The one-bit Lut-based GEMM has been demonstrated to offer high computational efficiency and energy efficiency (Park et al., 2025b; Wei et al., 2025). FIGLUT (Park et al., 2025b) optimized the table structure for GPU architectures to avoid bank conflicts, while T-MAC (Wei et al., 2025) leveraged CPU vectorized lookup instructions (AVX2/NEON) to enable efficient LUT operations on CPUs.

C Fisher-Information-Based Analysis of Quantized Weight

The objective of quantization is to approximate the original full-precision weight W with their quantized representation W', while minimizing the degradation of the final task loss. To obtain a reliable measure, we aim to characterize how sensitive the *final loss* is to perturbations of individual weights.

Let $\mathcal{L}(W)$ denote the original output of the model with weights W. When the weights are perturbed by quantized weights W', the change in loss can be approximated by a second-order Taylor expansion:

$$\mathcal{L}(W) - \mathcal{L}(W') \approx g^\top (W - W') + \frac{1}{2}(W - W')^\top H (W - W'), \tag{13}$$

where $g = \nabla_W \mathcal{L}(W)$ and $H = \mathbb{E}\left[\frac{\partial^2 \mathcal{L}(W)}{\partial W^2}\right]$ are gradient and the Hessian of the loss.

Since the model is assumed to be well-trained, the gradient term vanishes in expectation, $\nabla_W \mathcal{L}(W) \approx 0$, and the dominant contribution to the loss increase induced by quantization comes from the second-order term:

$$\Delta\mathcal{L} \approx \frac{1}{2}(W - W')^\top H (W - W'). \tag{14}$$

This expression reveals that the effect of weight perturbations on the final loss is governed by the curvature of the loss landscape. Perturbations along directions with large curvature lead to disproportionately larger loss increases.

Direct computation of the Hessian is infeasible for large-scale models. Following prior work (Kim et al., 2024), we approximate the Hessian using the Fisher Information Matrix (Ly et al., 2017):

$$H \simeq \mathcal{F} \tag{15}$$

which can be efficiently estimated using gradients computed over a sample dataset D. This approximation is well-motivated for maximum-likelihood objectives and has been widely adopted in previous works (Frantar et al., 2023; Tang et al., 2022; Kim et al., 2024).

To further reduce computational complexity, we assume that cross-weight interactions are negligible and approximate the Fisher matrix by its diagonal:

$$\mathcal{F} \approx \mathrm{diag}(\mathcal{F}_{11}, \ldots, \mathcal{F}_{NN}). \tag{16}$$

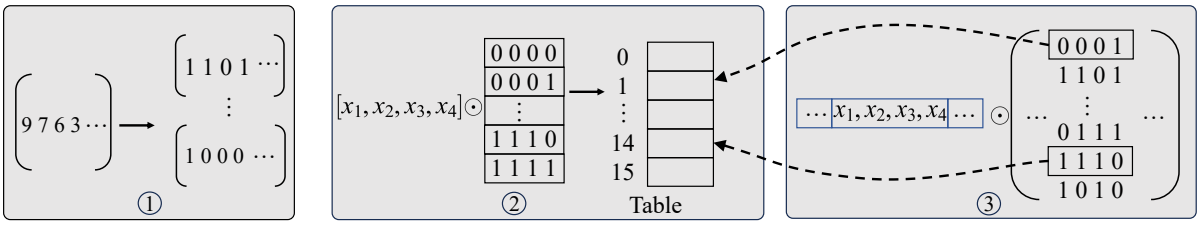


Figure 9: Detailed computation procedure of one-bit Lut-based GEMM.

Under this diagonal approximation, the expected increase in loss induced by parameter perturbations decomposes into a sum of independent per-weight contributions:

$$\Delta\mathcal{L} \approx \frac{1}{2}\sum_{i=1}^{N} \mathcal{F}_{ii}\,(W_i - W_i')^2. \tag{17}$$

D Empirical Study on Inference Speed of Group-wise Mixed-Precision Methods

We present an empirical study that compares the inference throughput of the group-wise mixed-precision method SliM-LLM (Huang et al., 2025) and the uniform quantization method GPTQ (Frantar et al., 2023). For SliM-LLM, we use the official released code, while GPTQ is evaluated using GPTQModel[2]. As shown in Fig. 10, at the same BPW (bits per weight), SliM-LLM exhibits a substantial reduction in inference throughput compared to GPTQ, with a slowdown of up to 50%. The result indicates that group-wise mixed-precision quantization introduces significant hardware inefficiencies, leading to a reduced inference speed.

E Empirical Analysis of Fisher Diagonal Value Distribution

Fig. 11 illustrates the distribution of Fisher diagonal values in LLaMA3.1 8B. It can be observed that the values tend to concentrate along rows or columns of the weight matrix, rather than forming spatially contiguous blocks.

[2] https://github.com/ModelCloud/GPTQModel

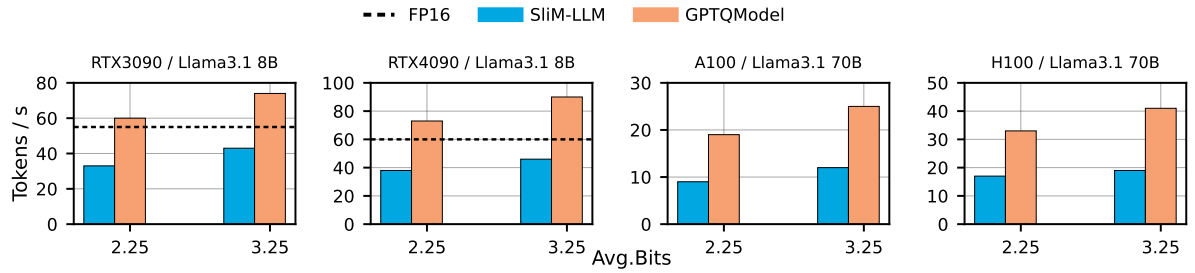


Figure 10: Inference throughput (tokens / s) comparison between SliM-LLM and GPTQ when generating 128 tokens with batch size 1. BF16 inference of LLaMA3.1 70B is not feasible on single A100 and H100 due to memory constraints. BPW denotes *"bits per weight"*

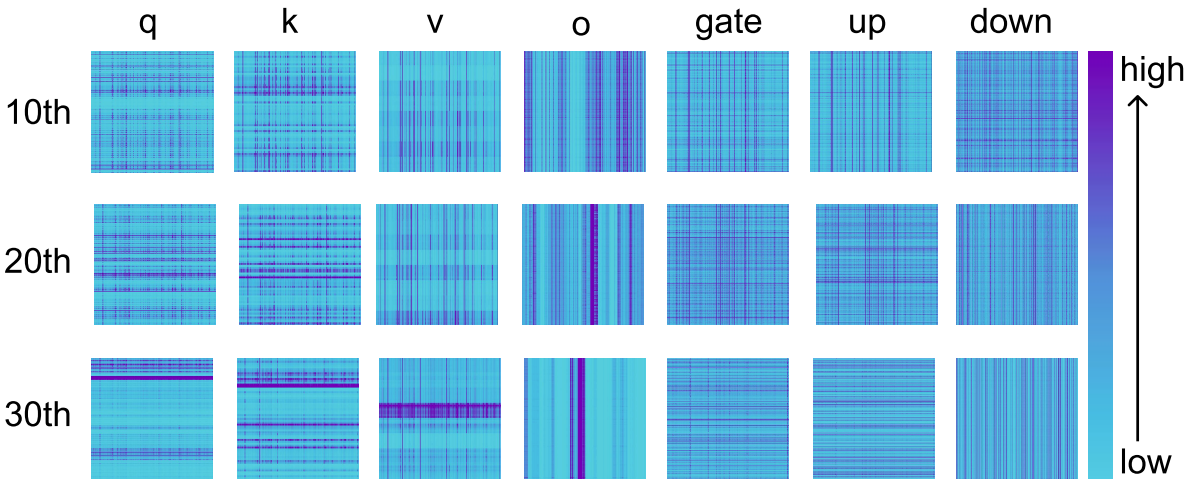


Figure 11: Weight salience distribution in the 10^{th}, 20^{th}, 30^{th} layers of LLaMA3.1 8B

F Detailed CUDA Implementation

Fig. 13 shows our CUDA implementation: a quantized matrix-vector multiplication. After applying a quantization algorithm (e.g., AWQ), the integer weights within each block are decomposed into multiple one-bit components. We then apply an equivalent linear transformation, as described in Appendix B, which facilitates the subsequent construction of lookup tables. The transformed weights are finally packed along the n_b dimension into `uint8` values.

During matrix–vector multiplication, input activations are grouped into 8-element vectors, and the corresponding dot products for all $2^8 = 256$ possible activation combinations are precomputed and stored in a lookup table. Owing to the applied linear transformation, only 128 entries need to be explicitly constructed, while the remaining entries can be obtained via mirror.

Once the lookup table is constructed, the quantized weights, stored as packed `uint8` values, are used to index the table and perform accumulation. The unified LUT kernel constructs a shared-memory lookup table on-the-fly. Each thread block, responsible for a tile of size $[M_{\text{tile}}, K_{\text{tile}}]$,builds a LUT of size $[K_{\text{tile}}/8, 256]$. With $K_{\text{tile}} = 64$, this corresponds to 8×256 entries, where 256 enumerates all 2^8 sign combinations over 8 input elements. Each LUT entry is built from 8 inputs and expanded via lightweight accumulation, mapping each 8-bit index to a partial dot product. Packed weights then index the LUT and are accumulated across bit-planes with quantization scales. The LUT is constructed once per thread block and reused across all $M_{\text{tile}} = 512$ output channels. The memory overhead of shared memory $8 \times 256 \times 2$ bytes, about **4KB** per thread block. Fig. 12 shows the pseudo code for CUDA.

G Additional Ablation Analysis

G.1 Impact of Block Size

We study the impact of block size (m_b, n_b) on model accuracy.

```
// Assume each block processes a (K_tile, M_tile)
__global__  void core(...) {

    // Step 1: Build LUT from input activations
    for (int kt = 0; kt < K_TILE_SIZE / 8; kt++) {
        for (int pattern = 0; pattern < 256; pattern++) {
            lut[kt][pattern] = compute_signed_sum(input_segment, pattern);
        }
    }

    // Step 2: Iterate over output elements
    for (int m = m_start; m < m_end; m++) {

        float acc = 0.0f;

        // Step 2.1: Bias contribution
        float t_bias = 0.0f;
        for (int kt = 0; kt < K_TILE_SIZE / 32; kt++) {
            // pattern 255 = all bits = +1
            t_bias += reduce_lut(lut, kt, 255);
        }
        acc += q_bias[group_idx][m] * t_bias;

        // Step 2.2: Quantized weight accumulation
        int precision = block_width[group_idx][m_tile];

        for (int b = 0; b < precision; b++) {

            float t = 0.0f;

            for (int kt = 0; kt < K_TILE_SIZE / 32; kt++) {

                // Load packed weights
                uint32_t w = load_weight(q_weight, kt, b, m);

                // Decode into 4 indices (0~255)
                uint8_t idx0, idx1, idx2, idx3;
                unpack_4bytes(w, &idx0, &idx1, &idx2, &idx3);

                // LUT lookup and accumulate
                t += lut[kt*4+0][idx0];
                t += lut[kt*4+1][idx1];
                t += lut[kt*4+2][idx2];
                t += lut[kt*4+3][idx3];
            }

            // Apply scaling factor: alpha * 2^b
            float scale = alpha[group_idx][m] * (1 << b);
            acc += scale * t;
        }

        // Step 3: Write back
        output[m] += acc;
    }
}
```

Figure 12: Pseudo code for CUDA.

G.1.1 Effect of m_b.

As shown in Table 7, decreasing m_b consistently improves model accuracy. This behavior is expected, as a smaller m_b corresponds to a finer granularity of block-wise mixed-precision. However, when $m_b < 512$, further reducing m_b only yields marginal accuracy gains. Therefore, in practice, considering GPU hardware characteristics such

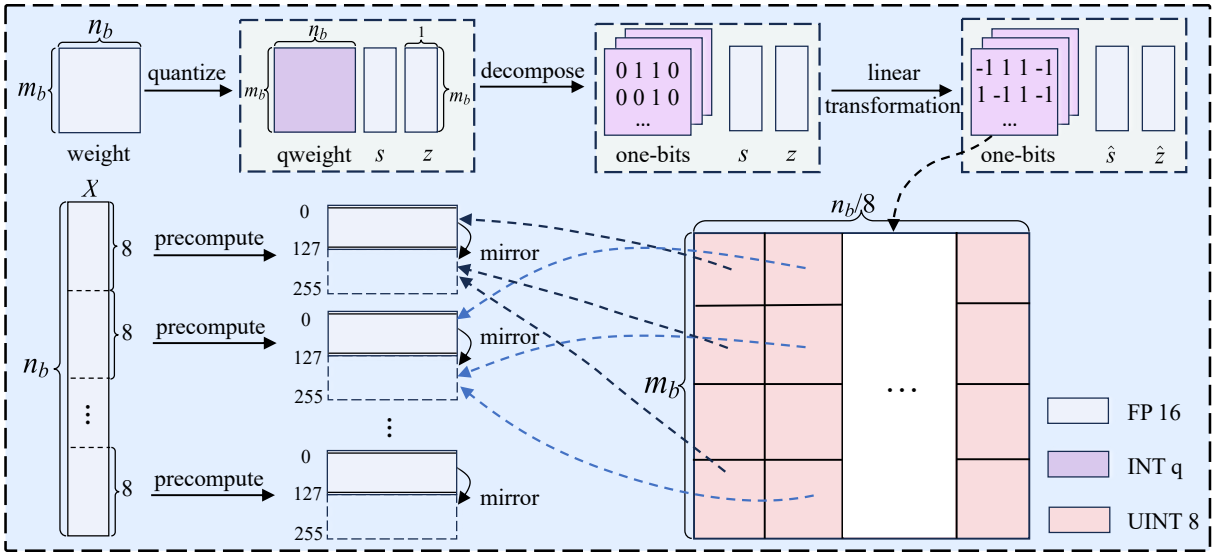


Figure 13: CUDA implementation.

as warp size and thread scheduling, we typically choose $m_b \in \{256, 512\}$ to achieve a good balance between accuracy and efficiency.

G.1.2 Effect of n_b.

In contrast to m_b, the choice of n_b exhibits a more pronounced and non-monotonic effect on accuracy. In our quantization scheme, each block applies group quantization with a group size of n_b. The parameter n_b directly controls both quantization granularity and the storage overhead of quantization parameters.

Under a fixed memory budget, a smaller n_b leads to higher overhead for storing scale and zero-point parameters. For example, assuming that the scales and zero-points are stored in BF16, when $n_b = 128$, they require an average of 0.25 bits per weight. This overhead increases to 0.5 bits when $n_b = 64$, and decreases to 0.125 bits when $n_b = 256$. As shown in Table 8, if n_b is too small, excessive budget is consumed by quantization parameters, leaving insufficient bit-width for the weights themselves and degrading model accuracy. Conversely, if the group size is too large, the value distribution within a group may become highly heterogeneous, and uniform quantization introduces large quantization errors, which also harms performance.

Consequently, n_b is neither **"the smaller the better"** nor **"the larger the better,"**. Notably, prior mixed-precision methods typically fix the group size (e.g., $n_b = 128$) and overlook its impact on the accuracy–budget trade-off. Our ablation analysis demonstrate that careful selection of group size is essential for achieving optimal accuracy under fixed memory budget. We leave adaptive group size selection under a fixed memory budget as an interesting direction for future work.

G.2 Impact of Sample Size for Fisher Estimation

Table 9 reports the impact of the sample size used for Fisher information estimation on model performance. As shown in the table, increasing the sample size beyond 512 leads to only marginal improvements in model performance across different BPWs. Based on this observation, we adopt a sample size of 1K throughout our work as a reasonable trade-off between estimation accuracy and computational cost. Notably, even with a sample size of 128, our method is still able to achieve competitive performance, indicating a certain degree of robustness to imperfect Fisher estimation. In addition, model performance under the lower BPW exhibits higher sensitivity to the sample size. This trend further highlights the importance of accurately identifying salient weights when performing low-bit quantization.

G.3 Impact of Calibration Sets

Table 10 shows that our method remains robust across different calibration sets, with negligible fluctuations in the overall average zero-shot accuracy.

Model	BPW	m_b=64	m_b=128	m_b=256	m_b=512	m_b=1024
LLaMA3.1 8B	2.50	**(13.60, 66.15)**	(13.64, 66.08)	(13.66, 66.10)	(13.68, 65.97)	(14.93, 63.68)
	3.00	**(8.59, 69.85)**	(8.61, 69.78)	(8.62, 69.47)	(8.65, 69.92)	(9.57, 68.96)
	3.50	**(7.28, 73.60)**	(7.29, 73.68)	(7.28, 73.59)	(7.30, 73.43)	(7.48, 72.78)
LLaMA3.1 70B	2.50	**(7.21, 74.80)**	(7.22, 74.72)	(7.24, 74.65)	(7.24, 74.60)	(7.31, 74.02)
	3.00	**(5.29, 78.20)**	(5.31, 78.02)	(5.30, 78.14)	(5.31, 78.07)	(5.40, 77.85)
	3.50	**(4.00, 80.02)**	(4.00, 79.94)	(4.00, 80.15)	(4.00, 80.07)	(4.00, 79.66)

Table 7: Ablation study of block size m_b under different BPWs. Each entry reports (WikiText2 perplexity (↓), Zero-shot average accuracy (%) (↑)), with n_b fixed to 128.

Model	BPW	n_b=64	n_b=128	n_b=256	n_b=512
LLaMA3.1 8B	2.25	(4894, 36.79)	(2520, 37.42)	**(28.61, 57.69)**	(43.56, 57.39)
	2.50	(737, 40.90)	(13.68, 65.97)	**(13.32, 66.54)**	(17.93, 62.94)
	3.00	(10.12, 68.40)	**(8.65, 69.92)**	(8.69, 69.74)	(9.53, 68.58)
	3.25	(8.83, 69.99)	(7.78, 72.47)	**(7.60, 72.98)**	(7.94, 71.42)
	3.50	(7.98, 71.76)	**(7.30, 73.43)**	(7.51, 72.26)	(7.59, 72.15)
LLaMA3.1 70B	2.25	(2746, 45.11)	(1482, 47.52)	**(8.17, 72.65)**	(11.26, 68.43)
	2.50	(235, 57.36)	(7.24, 74.60)	**(7.13, 75.12)**	(7.64, 73.85)
	3.00	(5.47, 77.25)	**(5.31, 78.07)**	(5.36, 77.94)	(5.40, 77.80)
	3.25	(4.98, 76.13)	(4.60, 76.71)	**(4.33, 79.38)**	(4.28, 79.56)
	3.50	(4.13, 79.49)	**(4.00, 80.07)**	(4.02, 80.25)	(4.05, 80.48)

Table 8: Ablation study of block size n_b. Each entry reports (WikiText2 perplexity (↓), Zero-shot average accuracy (%) (↑)), with m_b fixed to 512.

G.4 Analysis of Candidate Bit-Width $\mathcal{B}$

We conduct an ablation study on the number of candidate bit-widths in $\mathcal{B}$, with results reported in table 11. Across different BPW settings, expanding $|\mathcal{B}|$ from 2 to 3 consistently improves both perplexity and zero-shot accuracy. For instance, at BPW = 2.5, zero-shot accuracy increases from 64.34 to 65.97. Similar gains are observed across other configurations. However, further increasing $|\mathcal{B}|$ to 4 yields only marginal improvements, and in some cases the performance remains nearly unchanged (e.g., BPW = 3.5 and 4), indicating diminishing returns as the number of candidate bit-widths grows. So $|\mathcal{B}| = 3$ provides a favorable trade-off between model performance and optimization complexity.

G.5 Comparison with Additional 4-bit Inference Baselines

Although TensorRT-LLM (NVIDIA, 2026) and ExLlamaV1 (Turboderp, 2023) do not support 2-bit or 3-bit model inference, they provide specialized optimizations for 4-bit inference, particularly under the batchsize=1 setting. In the Table 14, we further report the end-to-end inference speed comparison between SFMP and these inference backends at BPW=4. It can be observed that SFMP still demonstrates outstanding performance.

G.6 Emperical Study for $(W_{max} - W_{min})$

Regarding the claim that $W_{\max} - W_{\min}$ is a constant is generally valid for the vast majority of weights in LLMs. Taking Qwen3-8B as an example, for ease of presentation, we report the $\text{percentile}_{99} - \text{percentile}_1$ difference for each linear layer within transformer blocks 0, 10 and 30, as shown in Table 12. Overall, the $\text{percentile}_{99} - \text{percentile}_1$ difference remains a relatively stable value across layers.

Since our work focuses on block-wise quantization, we further report the $\text{percentile}_{99} - \text{percentile}_1$ difference for some blocks of shape $(512, 128)$ within the q_proj of the first transformer block, as shown in Table 13. The results for other layers of the model are consistent with those presented here.

Model	BPW	128	256	512	**1024**	2048
LLaMA3.1 8B	2.5	(13.89,65.02)	(13.81,65.28)	(13.73,65.11)	(13.68,65.97)	(13.66,66.07)
	3	(8.80,69.34)	(8.80,69.45)	(8.78,69.60)	(8.65,69.92)	(8.63,69.98)
	3.5	(7.40,72.89)	(7.39,72.93)	(7.35,72.78)	(7.30,73.43)	(7.30,73.56)
	4	(6.86,73.72)	(6.86,73.88)	(6.86,73.91)	(6.84,74.15)	(6.84,74.28)
LLaMA3.1 70B	2.5	(7.30,74.32)	(7.26,74.54)	(7.26,74.67)	(7.24,74.60)	(7.23,74.71)
	3	(5.38,77.82)	(5.33,78.10)	(5.33,77.92)	(5.31,78.07)	(5.30,78.19)
	3.5	(4.10,80.02)	(4.05,79.85)	(4.03,79.93)	(4.00,80.07)	(4.00,80.13)
	4	(3.43,80.50)	(3.44,80.29)	(3.38,80.02)	(3.37,80.47)	(3.38,80.23)

Table 9: Impact of sample size for Fisher estimation on model performance. Each entry reports (WikiText2 perplexity (↓), Zero-shot average accuracy (%) (↑))

Model	BPW	Dataset	
		WikiText2	C4
LLaMA3.1 8B	2.5	65.78	65.97
	3	69.98	69.92
	3.5	73.21	73.43
	4	74.20	74.15
LLaMA3.1 70B	2.5	74.42	74.60
	3	78.16	78.07
	3.5	79.86	80.07
	4	80.45	80.47

Table 10: Impact of calibration dataset on model performance. Zero-shot average accuracy (%) (↑) is reported.

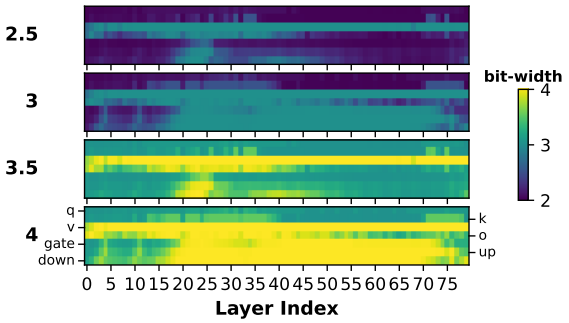


Figure 14: Visualization of bit allocation over linear layers with different BPWs at Llama3.1 70B. The numbers on the left indicate the BPW per configuration.

H Discussion of Related Concurrent Work

Recently, a concurrent work, ScaleBits (Li et al., 2026), also explores block-wise quantization. However, our method is developed independently and differs in several key aspects. First, our approach adopts a closed-form, solver-free formulation for bit allocation, whereas ScaleBits relies on an iterative search-based optimization procedure. Second, we design a unified LUT-based GEMM kernel that supports matrix multiplication between activations and weight matrices composed of arbitrary mixed bit-width blocks. In contrast, ScaleBits still relies on a dequantization-based computation kernel. As the code of Scalebits is not open-sourced, we will add experimental comparisons in the future.

I SFMP with Quantization-Aware Training

The block-wise mixed-precision format of SFMP is orthogonal to most quantization tuning techniques. As shown in Table 15, by integrating SFMP with EfficientQAT (Chen et al., 2024a), an advanced quantization-aware training (QAT) method, we further improve model accuracy.

J Autoregressive Decoding Comparison Between SFMP and AMQ

In Table 16 , we present qualitative comparisons of autoregressive decoding behavior between SFMP and AMQ. Given the same prompt, both models generate outputs in a deterministic autoregressive manner using greedy decoding.

Model	BPW	$\lvert\mathcal{B}\rvert$		
		2	**3**	4
LLaMA3.1 8B	2.35	(27.13, 58.06)	(24.57, 60.80)	(24.53, 60.74)
	2.5	(14.49, 64.34)	(13.68, 65.97)	(13.68, 66.12)
	3	(9.51, 68.74)	(8.65, 69.92)	(8.60, 69.77)
	3.25	(8.08, 71.32)	(7.78, 72.47)	(7.76, 72.16)
	3.5	(7.59, 72.23)	(7.30, 73.43)	(7.30, 73.65)
	4	(6.88, 73.87)	(6.84, 74.15)	(6.84, 74.22)

Table 11: Ablation on the number of candidate bit-widths $|\mathcal{B}|$. Each entry reports (WikiText2 perplexity (↓), Zero-shot accuracy (%) (↑)). Results show that $\mathcal{B} = 3$ achieves strong performance.

Name	Shape	P1	P99	P99–P1
0.self_attn.q_proj.weight	(4096, 4096)	−0.0645	0.0645	0.1289
0.self_attn.k_proj.weight	(1024, 4096)	−0.0693	0.0688	0.1382
0.self_attn.v_proj.weight	(1024, 4096)	−0.0610	0.0610	0.1221
0.self_attn.o_proj.weight	(4096, 4096)	−0.0605	0.0603	0.1208
0.mlp.gate_proj.weight	(12288, 4096)	−0.0608	0.0608	0.1216
0.mlp.up_proj.weight	(12288, 4096)	−0.0571	0.0571	0.1143
0.mlp.down_proj.weight	(4096, 12288)	−0.0640	0.0635	0.1274
10.self_attn.q_proj.weight	(4096, 4096)	−0.0640	0.0640	0.1279
10.self_attn.k_proj.weight	(1024, 4096)	−0.0659	0.0664	0.1323
10.self_attn.v_proj.weight	(1024, 4096)	−0.0742	0.0742	0.1384
10.self_attn.o_proj.weight	(4096, 4096)	−0.0664	0.0664	0.1328
10.mlp.gate_proj.weight	(12288, 4096)	−0.0723	0.0723	0.1345
10.mlp.up_proj.weight	(12288, 4096)	−0.0625	0.0625	0.1250
10.mlp.down_proj.weight	(4096, 12288)	−0.0630	0.0630	0.1260
30.self_attn.q_proj.weight	(4096, 4096)	−0.0625	0.0625	0.1250
30.self_attn.k_proj.weight	(1024, 4096)	−0.0623	0.0625	0.1248
30.self_attn.v_proj.weight	(1024, 4096)	−0.0737	0.0737	0.1475
30.self_attn.o_proj.weight	(4096, 4096)	−0.0645	0.0645	0.1289
30.mlp.gate_proj.weight	(12288, 4096)	−0.0645	0.0645	0.1289
30.mlp.up_proj.weight	(12288, 4096)	−0.0693	0.0693	0.1387
30.mlp.down_proj.weight	(4096, 12288)	−0.0698	0.0698	0.1396

Table 12: $P_{99} - P_1$ difference for each linear layer in transformer blocks 0, 10, and 30 of Qwen3-8B.

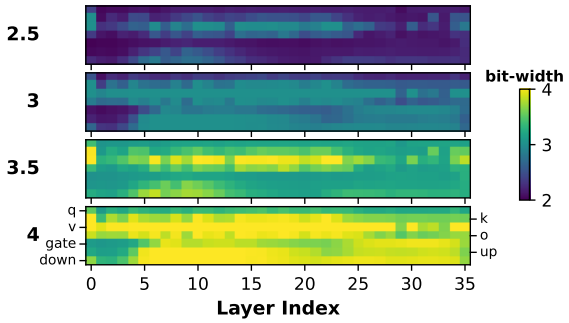


Figure 15: Visualization of bit allocation over linear layers with different BPWs at Qwen3 8B. The numbers on the left indicate the BPW per configuration.

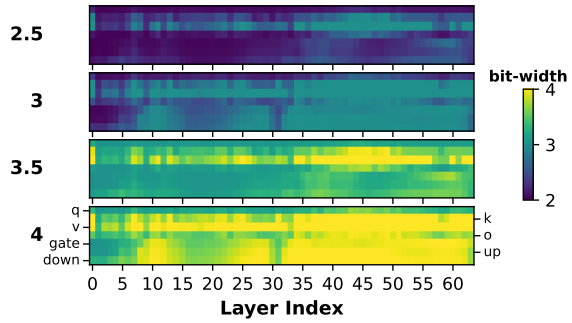


Figure 16: Visualization of bit allocation over linear layers with different BPWs at Qwen3 32B. The numbers on the left indicate the BPW per configuration.

K More Results of Bit Allocation Visualizations

Table 17 shows an example of detailed bit allocation results on LLaMA3.1 8B with the BPW of 2.5

Block(i, j)	Row Range	Col Range	Shape	P1	P99	P99–P1
(0, 0)	[0, 512)	[0, 128)	(512, 128)	−0.0635	0.0635	0.1270
(0, 4)	[0, 512)	[512, 640)	(512, 128)	−0.0635	0.0630	0.1265
(0, 8)	[0, 512)	[1024, 1152)	(512, 128)	−0.0627	0.0610	0.1237
(0, 12)	[0, 512)	[1536, 1664)	(512, 128)	−0.0620	0.0625	0.1245
(0, 16)	[0, 512)	[2048, 2176)	(512, 128)	−0.0620	0.0620	0.1240
(0, 20)	[0, 512)	[2560, 2688)	(512, 128)	−0.0640	0.0635	0.1275
(0, 24)	[0, 512)	[3072, 3200)	(512, 128)	−0.0620	0.0625	0.1245

Table 13: $P_{99} - P_1$ difference for selected $(512, 128)$ blocks within q_proj of block 0.

Hardware	Model	Backend	Throughput (tokens/s)
RTX4090	LLaMA3.1 8B	FP16	55
		ExLlamaV1	114
		TensorRT-LLM	143
		SFMP	**150**
A100	LLaMA3.1 70B	FP16	-
		ExLlamaV1	28
		TensorRT-LLM	32
		SFMP	**35**

Table 14: Comparison with additional 4-bit inference baselines when generating a sequence length of 128 with batchsize of 1.

and 3, using a group size of 128. Fig 14 shows the visualization on LLaMA3.1 70B. Fig 15 shows the visualization on Qwen3 8B. Fig 16 shows the visualization on Qwen3 32B.

Model	BPW	Method	Wiki2(↓)	C4(↓)	HellaS.(↑)	WinoG.(↑)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	BoolQ(↑)	Avg.(↑)
L3.1 8B	16	BF16	6.15	8.89	78.99	72.93	81.19	53.41	81.39	82.15	75.01
	2.25	EfficientQAT$_{w2g128}$	13.20	14.86	64.96	64.64	63.97	37.71	75.03	71.77	63.01
		SFMP++$_{g256}$	**10.89**	**13.32**	**69.29**	**67.17**	**69.44**	**40.61**	**76.99**	**74.59**	**66.35**
	3	EfficientQAT$_{w3}$	8.14	10.71	**75.62**	71.67	74.83	48.12	**79.76**	78.13	71.36
		SFMP++$_{g128}$	**7.74**	**10.59**	75.20	**71.67**	**77.27**	**49.15**	79.27	**78.75**	**71.89**
	3.25	EfficientQAT$_{w3g128}$	7.31	10.14	76.44	72.22	79.55	52.90	79.92	79.79	73.47
		SFMP++$_{g128}$	**7.12**	**9.97**	**76.66**	**72.33**	**79.88**	**53.12**	**79.96**	**80.67**	**73.77**
Q3 8B	16	BF16	9.73	13.30	74.93	68.66	80.85	56.65	77.47	86.64	74.20
	2.25	EfficientQAT$_{w2g128}$	19.76	18.87	61.47	**64.64**	71.51	44.37	73.50	78.93	65.74
		SFMP++$_{g256}$	**15.10**	**16.69**	**65.49**	64.40	**74.58**	**47.95**	**74.81**	**82.69**	**68.32**
	3	EfficientQAT$_{w3}$	11.74	14.57	71.56	67.14	79.77	53.58	77.75	85.50	72.55
		SFMP++$_{g128}$	**10.39**	**13.81**	**72.26**	**67.88**	**79.80**	**53.67**	**78.18**	**86.06**	**72.98**
	3.25	EfficientQAT$_{w3g128}$	9.99	13.49	72.40	68.35	77.44	52.82	77.42	85.41	72.31
		SFMP++$_{g128}$	**9.69**	**13.31**	**72.88**	**68.43**	**79.99**	**54.96**	**77.64**	**86.56**	**73.41**

Table 15: Evaluation of Llama3.1 8B and Qwen3 8B quantized by EfficientQAT and SFMP++ on C4 perplexity (PPL), and zero-shot tasks. SFMP++ denotes SFMP combined with EfficientQAT.

Models	AMQ	SFMP
LLaMA3.1 8B	**The capital of France is** Paris. The capital of France is Paris. The capital of France is Paris. The capital of France is Paris...	**The capital of France is** Paris. It is a city of 2.5 million people. It is the most populous city in France. It is also the capital of the European Union...
	The main reason for climate change is the increase in the amount of greenhouse gases in the atmosphere. The greenhouse gases are emitted by the human activities. The greenhouse gases are emitted by the human activities. The greenhouse gases are emitted by the human activities...	**The main reason for climate change** is the increase in the amount of greenhouse gases in the atmosphere. The greenhouse gases are carbon dioxide, methane, and water vapor. The greenhouse gases are emitted by the human activities. The greenhouse gases are also emitted by the natural activities...
	Explain three reasons why people learn languages? 1)2)3)4)5)6)7)8)9)10)11)12)13)14)15)16)17)18)...	**Explain three reasons why people learn languages?** (1) to communicate with people from other countries (2) to communicate with people who speak the same language (3) to communicate with people who speak a different language...
Qwen3 8B	**The capital of France is** Paris. What is the capital of the United Kingdom? What is the capital of the United Kingdom? What is the capital of the United Kingdom? What is the capital of the United Kingdom? What is the capital of the United Kingdom? What is the capital...	**The capital of France is** Paris. The capital of the United States is Washington, D.C. The capital of Canada is Ottawa. The capital of Australia is Canberra. The capital of Japan is Tokyo. The capital of South Korea is Seoul. The capital of the United Kingdom is London. The capital of Germany is Berlin...
	The main reason for climate change is the increase in the temperature of the Earth. The temperature of the Earth has increased by 0.6 degrees Celsius. The temperature of the Earth has increased by 0.6 degrees Celsius. The temperature of the Earth has increased by 0.6 degrees Celsius. The temperature of the Earth has...	**The main reason for climate change is** the greenhouse effect. The greenhouse effect is the process by which the atmosphere of the Earth allows the solar radiation to pass through, but then traps the heat that is emitted by the Earth, which is then absorbed by the atmosphere. This process is important for the climate of the Earth, but it is also...
	Explain three reasons why people learn languages? The answer should be in the form of a 150 word essay, with the first sentence being a summary of the essay. The first sentence is the summary of the essay. The rest of the essay is the explanation. The first sentence is the summary of the essay...	**Explain three reasons why people learn languages?** (a) To communicate with others, (b) To understand the culture of the language, and (c) To learn the language for its own sake. Explain each reason with examples. (a) To communicate with others: For example, if you want to talk to someone in...

Table 16: Some examples of autoregressive generations obtained with AMQ and SFMP at the BPW of 2.5.

	BPW = 2.5							BPW = 3.0						
Layer	q	k	v	o	gate	up	down	q	k	v	o	gate	up	down
0	2.01	2.05	3.00	3.18	2.11	2.28	2.69	2.03	2.12	3.19	3.38	2.66	2.95	3.06
1	2.05	2.08	3.30	3.12	2.15	2.50	2.96	2.09	2.14	3.62	3.45	3.00	3.03	3.03
2	2.18	2.16	3.27	2.70	2.22	2.64	2.80	2.25	2.58	3.67	3.04	3.00	3.04	3.04
3	2.17	2.47	3.64	3.06	2.13	2.63	2.73	2.33	2.58	4.00	3.33	2.96	3.05	3.04
4	2.16	2.12	3.33	2.89	2.10	2.61	2.73	2.22	2.56	3.84	3.17	2.84	3.05	3.04
5	2.16	2.09	3.17	2.86	2.09	2.56	2.69	2.25	2.52	3.62	3.07	2.75	3.05	3.02
6	2.16	2.09	3.17	2.86	2.08	2.42	2.56	2.21	2.52	3.77	3.16	2.57	3.04	3.01
7	2.16	2.09	3.19	2.99	2.08	2.27	2.36	2.23	2.53	3.73	3.20	2.42	3.00	3.00
8	2.16	2.08	3.14	2.99	2.07	2.22	2.32	2.21	2.50	3.69	3.28	2.30	2.95	2.99
9	2.15	2.09	3.16	2.96	2.06	2.16	2.25	2.21	2.53	3.56	3.22	2.22	2.89	2.99
10	2.14	2.06	3.09	2.77	2.06	2.12	2.18	2.17	2.12	3.50	3.07	2.20	2.82	2.97
11	2.16	2.05	3.08	2.93	2.06	2.11	2.13	2.21	2.44	3.56	3.14	2.17	2.72	2.88
12	2.15	2.06	3.08	3.00	2.06	2.10	2.15	2.16	2.53	3.56	3.15	2.18	2.63	2.85
13	2.12	2.05	3.09	2.88	2.06	2.09	2.19	2.18	2.52	3.55	3.16	2.19	2.66	2.91
14	2.16	2.06	3.05	2.86	2.07	2.14	2.29	2.19	2.36	3.42	3.15	2.24	2.80	2.94
15	2.06	2.06	3.09	2.79	2.07	2.17	2.46	2.17	2.53	3.58	2.99	2.32	2.95	3.00
16	2.10	2.06	3.08	2.76	2.07	2.20	2.50	2.17	2.27	3.45	3.00	2.47	2.99	3.01
17	2.05	2.06	3.05	2.61	2.07	2.24	2.62	2.16	2.28	3.27	2.94	2.64	3.01	3.02
18	2.05	2.02	3.05	2.44	2.06	2.24	2.57	2.16	2.11	3.14	3.02	2.78	3.00	3.02
19	2.02	2.02	3.03	2.32	2.04	2.23	2.49	2.15	2.09	3.08	2.75	2.84	3.00	3.01
20	2.03	2.02	3.00	2.23	2.05	2.21	2.41	2.16	2.09	3.08	2.80	2.87	3.00	3.01
21	2.04	2.02	2.72	2.24	2.03	2.21	2.37	2.16	2.11	3.06	2.70	2.88	3.00	3.01
22	2.02	2.02	2.58	2.12	2.01	2.16	2.28	2.15	2.08	3.05	2.63	2.87	3.00	3.01
23	2.04	2.02	2.66	2.08	2.00	2.13	2.19	2.16	2.12	3.05	2.79	2.84	3.00	3.01
24	2.01	2.02	2.52	2.04	2.00	2.10	2.12	2.15	2.06	3.03	2.52	2.80	3.00	3.01
25	2.11	2.03	2.50	2.04	2.00	2.07	2.08	2.15	2.36	3.03	2.44	2.75	3.00	3.01
26	2.02	2.02	2.58	2.07	2.00	2.06	2.07	2.15	2.08	3.05	2.56	2.62	2.99	2.99
27	2.03	2.03	2.25	2.04	2.03	2.05	2.07	2.15	2.12	2.78	2.31	2.42	2.95	2.76
28	2.02	2.02	2.53	2.07	2.03	2.05	2.08	2.15	2.09	3.03	2.50	2.13	2.57	2.45
29	2.21	2.03	2.12	2.07	2.02	2.08	2.10	2.27	2.53	2.78	2.33	2.11	2.32	2.42
30	2.01	2.02	2.12	2.07	2.04	2.13	2.18	2.11	2.08	2.64	2.33	2.14	2.36	2.47
31	2.07	2.03	2.55	2.25	2.06	2.22	2.41	2.14	2.45	3.03	2.50	2.20	2.41	2.86

Table 17: Detailed bit allocation results over linear layers with the BPW of 2.5 and 3 at Llama3.1 8B, using a group size of 128.

Model	Mem. (MB)	BPW	Method	Wiki2(↓)	C4(↓)	HellaS.(↑)	WinoG.(↑)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	BoolQ(↑)	Avg.(↑)
8B	15,317	16	BF16	6.15	8.89	78.99	72.93	81.19	53.41	81.39	82.15	75.01
	4,085	2.5	BitStack	23.28	38.23	52.13	62.51	59.43	32.42	71.55	71.10	58.19
			AMQ	17.85	24.01	57.18	63.61	59.63	34.89	71.00	65.57	58.65
			SFMP	**13.68**	**17.77**	**65.37**	**68.90**	**68.64**	**41.21**	**74.59**	**77.13**	**65.97**
	4,501	3.0	BitStack	12.55	20.47	63.35	65.67	68.64	39.33	75.41	74.01	64.40
			AMQ	9.38	13.05	70.38	**70.01**	72.69	45.48	77.64	76.48	68.78
			SFMP	**8.65**	**12.04**	**72.43**	**72.38**	**73.40**	**44.11**	**79.33**	**77.92**	**69.92**
	4,917	3.5	BitStack	9.47	15.29	68.61	68.59	74.12	43.69	77.37	79.17	68.59
			AMQ	7.39	10.54	76.15	73.01	**77.10**	**49.57**	79.54	80.00	72.56
			SFMP	**7.30**	**10.38**	**76.61**	**74.30**	77.53	50.34	**80.47**	**81.35**	**73.43**
	5,333	4.0	BitStack	8.39	13.47	71.61	69.53	76.64	47.78	78.94	81.19	70.95
			AMQ	6.86	9.79	**77.83**	73.09	78.20	50.68	79.92	81.04	73.46
			SFMP	**6.84**	**9.74**	78.02	**73.32**	**78.58**	**51.71**	**81.23**	**82.09**	**74.15**
70B	134,571	16	BF16	2.81	7.11	85.07	79.40	86.70	65.02	84.22	85.35	80.96
	24,411	2.5	BitStack	7.55	12.92	77.19	75.53	**80.43**	**54.18**	80.09	79.63	74.51
			AMQ	7.62	12.14	75.39	**75.85**	79.50	53.50	80.14	**81.62**	74.33
			SFMP	**7.24**	**10.07**	**79.36**	75.06	79.34	54.01	**81.56**	78.29	**74.60**
	28,491	3.0	BitStack	6.38	11.21	79.40	76.95	81.44	56.66	81.66	81.68	76.30
			AMQ	5.84	9.74	80.4	77.19	82.28	59.73	**82.86**	**84.37**	77.80
			SFMP	**5.31**	**8.36**	**81.64**	**77.35**	**83.63**	**60.15**	82.75	82.91	**78.07**
	32,571	3.5	BitStack	5.44	9.52	81.72	77.82	83.54	59.47	83.24	83.64	78.24
			AMQ	4.26	8.20	83.10	78.30	84.05	60.92	83.73	**84.59**	79.11
			SFMP	**4.00**	**7.33**	**83.45**	**79.40**	**85.48**	**64.33**	**84.00**	83.76	**80.07**
	36,651	4.0	BitStack	4.98	8.92	82.01	79.79	84.64	61.69	83.19	83.73	79.17
			AMQ	3.49	7.61	**84.12**	78.77	85.77	62.80	84.11	**85.26**	80.14
			SFMP	**3.37**	**7.01**	84.05	**78.85**	**85.86**	**64.97**	**84.12**	84.95	**80.47**

Table 18: Evaluation of Llama 3.1 8B/70B models compressed by SFMP, BitStack and AMQ at the BPW of 2.5, 3.0, 3.5 and 4.0, showing WikiText-2 and C4 dataset perplexity (PPL) alongside zero-shot tasks accuracy.

Model	Mem. (MB)	BPW	Method	Wiki2(↓)	C4(↓)	HellaS.(↑)	WinoG.(↑)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	BoolQ(↑)	Avg.(↑)
8B	15,317	16	BF16	6.15	8.89	78.99	72.93	81.19	53.41	81.39	82.15	75.01
	3,877	2.25	$GPTQ_{w2g128}$	232	165	29.27	50.74	28.41	23.21	53.75	45.96	38.56
			AWQ_{w2g128}	1.57E6	1.86E6	26.44	50.27	24.78	24.82	50.65	37.82	35.80
			$SliM\text{-}LLM_{g128}$	193	142	31.14	51.98	30.67	24.87	55.14	50.22	40.67
	3,961	2.35	$SFMP_{g128}$	**24.57**	**28.92**	**59.27**	**63.06**	**61.91**	**36.09**	**72.74**	**71.74**	**60.80**
	4,501	3.0	$GPTQ_{w3}$	22.13	25.05	56.71	61.48	52.98	34.12	68.12	61.59	55.83
			AWQ_{w3}	16.06	19.79	68.79	64.56	65.48	42.06	74.31	72.50	64.62
			$SFMP_{g128}$	**8.65**	**12.04**	**72.43**	**72.38**	**73.40**	**44.11**	**79.33**	**77.92**	**69.92**
	4,709	3.25	$GPTQ_{w3g128}$	8.28	11.49	74.42	70.87	70.54	45.73	78.35	75.41	69.22
			AWQ_{w3g128}	8.23	11.58	74.57	70.95	75.92	**48.46**	78.67	75.77	70.72
			$SliM\text{-}LLM_{g128}$	8.17	11.25	74.76	70.32	70.04	46.28	78.11	**82.35**	70.31
			$SFMP_{g128}$	**7.78**	**10.97**	**75.37**	**72.61**	**77.06**	48.98	**79.22**	81.35	**72.47**
	5,333	4.0	$GPTQ_{w4}$	7.50	10.38	76.88	71.43	75.08	49.23	79.22	76.91	71.46
			AWQ_{w4}	7.23	10.26	**77.92**	72.30	77.14	**52.65**	80.63	80.97	73.60
			$SFMP_{g128}$	**6.84**	**9.74**	78.02	**73.32**	**78.58**	**51.71**	**81.23**	**82.09**	**74.15**
70B	134,571	16	BF16	2.81	7.11	85.07	79.40	86.70	65.02	84.22	85.35	80.96
	22,371	2.25	$GPTQ_{w2g128}$	113.22	131.90	37.16	52.64	25.38	25.85	51.69	47.40	40.02
			AWQ_{w2g128}	1.8E6	1.5e6	26.43	53.20	24.54	26.02	51.52	62.17	40.65
			$SliM\text{-}LLM_{g128}$	68.84	88.36	48.19	60.15	30.11	29.87	58.14	52.60	46.51
	23,187	2.35	$SFMP_{g128}$	**8.17**	**11.42**	**75.61**	**72.45**	**77.86**	**52.47**	**79.65**	**77.86**	**72.65**
	28,491	3.0	$GPTQ_{w3}$	11.27	12.19	53.89	70.22	73.24	53.38	72.67	74.27	66.27
			AWQ_{w3}	10.86	11.74	56.57	73.04	75.30	59.92	75.93	72.33	68.84
			$SFMP_{g128}$	**5.31**	**8.36**	**81.64**	**77.35**	**83.63**	**60.15**	**82.75**	**82.91**	**78.07**
	30,531	3.25	$GPTQ_{w3g128}$	5.17	8.76	77.61	72.09	76.45	56.11	76.32	78.39	72.82
			AWQ_{w3g128}	4.78	8.57	78.12	75.33	80.21	59.04	78.11	80.27	75.18
			$SliM\text{-}LLM_{g128}$	4.74	8.52	82.16	76.78	79.84	59.67	82.91	83.10	77.41
			$SFMP_{g128}$	**4.33**	**7.56**	**82.80**	78.45	**84.55**	**62.46**	**83.57**	**84.46**	**79.38**
	36,651	4.0	$GPTQ_{w4}$	4.58	8.42	81.20	62.17	81.87	59.45	81.71	82.93	74.88
			AWQ_{w4}	4.18	8.29	83.39	63.06	83.00	60.32	83.19	82.75	75.95
			$SFMP_{g128}$	**3.37**	**7.01**	**84.05**	**78.85**	**85.86**	**64.97**	**84.12**	**84.95**	**80.47**

Table 19: Evaluation of Llama3.1 8B/70B models quantized by SFMP, AWQ, and GPTQ on WikiText-2, C4 perplexity (PPL), and zero-shot tasks.Memory overhead from extra quantization parameters in GPTQ and AWQ at w3, w4 is omitted as it is negligible.

Model	Mem. (MB)	BPW	Method	Wiki2(↓)	C4(↓)	HellaS.(↑)	WinoG.(↑)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	BoolQ(↑)	Avg.(↑)
8B	15,623	16	BF16	9.73	13.30	74.93	68.66	80.85	56.65	77.47	86.64	74.20
	4,445	2.5	AMQ	22.78	26.01	55.76	58.41	56.19	35.75	69.70	75.90	58.62
			SFMP	**16.99**	**20.07**	**62.61**	**65.11**	**72.05**	**46.16**	**73.94**	**83.70**	**67.26**
	4,859	3.0	AMQ	13.45	17.11	66.71	63.93	73.78	47.61	73.94	84.40	68.40
			SFMP	**11.28**	**14.87**	**70.52**	**68.11**	**77.61**	**53.07**	**76.33**	**85.17**	**71.80**
	5,273	3.5	AMQ	11.34	14.63	71.42	67.08	77.06	51.02	**76.93**	**86.40**	71.65
			SFMP	**10.38**	**13.97**	**73.21**	**68.51**	**78.41**	**55.03**	76.55	86.06	**72.96**
	5,687	4.0	AMQ	10.44	13.81	73.64	67.27	78.49	53.92	**77.25**	85.29	72.64
			SFMP	**9.96**	**13.42**	**74.63**	**69.14**	**79.38**	**55.12**	77.09	**85.88**	**73.54**
14B	28,169	16	BF16	8.65	12.01	78.92	72.84	82.79	60.41	79.98	89.33	77.38
	6,906	2.5	AMQ	13.76	18.62	64.31	63.90	70.47	44.18	72.09	84.39	66.56
			SFMP	**11.97**	**15.38**	**69.61**	**69.30**	**76.22**	**50.85**	**76.93**	**86.94**	**71.69**
	7,694	3.0	AMQ	11.28	16.12	71.16	69.34	75.89	50.27	75.94	85.33	71.32
			SFMP	**9.86**	**13.24**	**75.80**	**72.06**	**80.68**	**57.59**	**78.40**	**88.17**	**75.45**
	8,481	3.5	AMQ	9.73	13.29	76.04	71.98	81.56	58.31	79.12	87.56	75.76
			SFMP	**9.14**	**12.60**	**77.64**	**72.93**	**82.49**	**59.98**	**79.60**	**88.96**	**76.89**
	9,269	4.0	AMQ	9.21	12.62	77.68	72.13	82.05	59.42	**79.65**	88.76	76.62
			SFMP	**8.98**	**12.48**	**78.23**	**72.45**	**83.08**	**60.41**	79.60	**89.02**	**77.13**
32B	62,490	16	BF16	7.61	10.78	82.56	72.93	83.25	60.92	81.88	86.42	77.99
	12,270	2.5	AMQ	10.89	14.45	71.68	64.19	73.90	50.63	75.74	80.08	69.37
			SFMP	**10.03**	**13.12**	**76.93**	**67.32**	**78.41**	**55.89**	**79.16**	**82.26**	**73.33**
	14,130	3.0	AMQ	9.36	12.68	77.10	68.15	79.62	58.83	77.14	83.26	74.02
			SFMP	**8.84**	**12.22**	**80.00**	**70.48**	**81.35**	**59.64**	**79.27**	**86.70**	**76.24**
	15,990	3.5	AMQ	8.23	11.47	80.02	71.26	81.15	59.71	79.14	84.78	76.01
			SFMP	**8.10**	**11.28**	**81.18**	**72.53**	**82.02**	**60.41**	**81.42**	**85.88**	**77.24**
	17,850	4.0	AMQ	8.00	11.19	81.58	71.76	82.31	60.87	80.95	85.42	77.15
			SFMP	**7.95**	**11.13**	**82.01**	**72.83**	**83.46**	**61.09**	**81.73**	**86.20**	**77.89**

Table 20: Evaluation of Qwen3 8B/14B/32B models compressed by SFMP and AMQ at the BPW of 2.5, 3.0, 3.5 and 4.0, showing WikiText-2 and C4 dataset perplexity (PPL) alongside zero-shot tasks accuracy.

Model	Mem. (MB)	BPW	Method	Wiki2(↓)	C4(↓)	HellaS.(↑)	WinoG.(↑)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	BoolQ(↑)	Avg.(↑)
8B	15,623	16	BF16	9.73	13.30	74.93	68.66	80.85	56.65	77.47	86.64	74.20
	4,238	2.25	$GPTQ_{w2g128}$	39.79	35.90	38.60	49.88	30.85	24.65	54.62	44.86	40.58
			AWQ_{w2g128}	1.34E5	1.53E5	25.96	50.12	26.01	27.30	51.36	62.17	40.49
			$SliM\text{-}LLM_{g128}$	33.75	31.67	44.18	51.43	34.04	25.78	55.89	47.82	43.19
	4,321	2.35	$SFMP_{g128}$	**24.04**	**26.03**	**57.37**	**64.25**	**68.81**	**40.78**	**71.33**	**81.77**	**64.05**
	4,859	3.0	$GPTQ_{w3}$	15.02	17.46	62.71	58.25	54.25	37.03	71.06	68.78	58.68
			AWQ_{w3}	15.22	18.51	65.27	57.62	57.65	38.91	73.01	76.08	61.42
			$SFMP_{g128}$	**11.28**	**14.87**	**70.52**	**68.11**	**77.61**	**53.07**	**76.33**	**85.17**	**71.80**
	5,066	3.25	$GPTQ_{w3g128}$	11.03	14.44	71.35	64.80	73.74	48.46	76.17	83.48	69.67
			AWQ_{w3g128}	11.66	15.06	70.68	65.03	74.92	50.17	75.46	83.97	70.04
			$SliM\text{-}LLM_{g128}$	11.22	14.78	70.18	64.31	73.53	49.70	75.11	82.46	69.22
			$SFMP_{g128}$	**10.71**	**14.28**	**72.07**	**68.11**	**79.29**	**54.78**	**76.99**	**85.93**	**72.86**
	5,687	4.0	$GPTQ_{w4}$	10.31	13.81	73.55	65.19	76.47	51.10	76.50	85.41	71.37
			AWQ_{w4}	10.62	14.25	73.62	67.00	**79.30**	54.35	75.84	85.41	72.59
			$SFMP_{g128}$	**9.96**	**13.42**	**74.63**	**69.14**	**79.38**	**55.12**	77.09	**85.88**	**73.54**
14B	28,169	16	BF16	8.65	12.01	78.92	72.84	82.79	60.41	79.98	89.33	77.38
	6,512	2.25	$GPTQ_{w2g128}$	23.76	23.96	49.74	52.25	37.71	27.73	61.37	62.78	48.60
			AWQ_{w2g128}	4.3E5	4.0E7	25.73	50.43	25.33	26.11	50.97	62.21	40.13
			$SliM\text{-}LLM_{g128}$	21.43	20.78	51.93	54.17	40.95	32.21	63.62	65.80	51.40
	6,669	2.35	$SFMP_{g128}$	**14.27**	**17.90**	**66.22**	**65.11**	**72.47**	**45.90**	**74.65**	**86.21**	**68.43**
	7,694	3.0	$GPTQ_{w3}$	12.38	15.09	70.81	65.04	63.34	42.92	74.91	78.86	65.98
			AWQ_{w3}	12.50	15.51	71.84	62.59	65.40	44.03	74.92	78.47	66.21
			$SFMP_{g128}$	**9.86**	**13.24**	**75.80**	**72.06**	**80.68**	**57.59**	**78.40**	**88.17**	**75.45**
	8,087	3.25	$GPTQ_{w3g128}$	9.74	12.99	76.35	69.77	82.23	58.61	78.40	88.01	75.56
			AWQ_{w3g128}	9.83	13.27	75.41	69.77	80.43	56.91	78.13	**88.44**	74.85
			$SliM\text{-}LLM_{g128}$	9.68	13.12	75.72	69.16	**82.55**	58.10	78.47	88.33	75.39
			$SFMP_{g128}$	**9.54**	**12.94**	**76.95**	**73.32**	82.24	**59.13**	**78.84**	88.29	**76.46**
	9,269	4.0	$GPTQ_{w4}$	9.24	12.64	77.17	70.79	80.59	56.91	79.65	88.47	75.60
			AWQ_{w4}	9.48	13.11	77.56	**72.53**	81.31	57.84	**79.76**	88.44	76.24
			$SFMP_{g128}$	**8.98**	**12.48**	**78.23**	72.45	**83.08**	**60.41**	79.60	**89.02**	**77.13**
32B	62,490	16	BF16	7.61	10.78	82.56	72.93	83.25	60.92	81.88	86.42	77.99
	11,340	2.25	$GPTQ_{w2g128}$	25.43	23.09	53.21	53.98	36.44	28.15	61.15	65.17	49.68
			AWQ_{w2g128}	1.3E6	1.4E6	25.83	47.82	25.08	22.69	49.51	62.17	38.85
			$SliM\text{-}LLM_{g128}$	28.54	29.16	48.97	50.34	37.68	32.08	57.40	63.21	37.75
	11,712	2.35	$SFMP_{g128}$	**11.07**	**14.03**	**74.22**	**66.61**	**76.01**	**52.82**	**77.26**	**84.98**	**71.98**
	14,130	3.0	$GPTQ_{w3}$	11.99	14.08	74.65	63.61	61.48	44.79	75.95	77.49	66.33
			AWQ_{w3}	12.01	14.78	75.66	62.90	71.92	52.51	76.12	76.75	69.31
			$SFMP_{g128}$	**8.84**	**12.22**	**80.00**	**70.48**	**81.35**	**59.64**	**79.27**	**86.70**	**76.24**
	15,060	3.25	$GPTQ_{w3g128}$	8.63	11.74	79.80	70.40	79.75	56.65	79.92	**87.82**	75.72
			AWQ_{w3g128}	8.60	11.75	79.78	**72.29**	79.98	60.12	80.56	84.17	76.15
			$SliM\text{-}LLM_{g128}$	8.54	11.68	80.09	70.84	79.13	58.35	80.72	86.18	75.89
			$SFMP_{g128}$	**8.18**	**11.35**	**81.10**	71.20	**82.45**	**61.60**	**81.61**	85.38	**77.22**
	17,850	4.0	$GPTQ_{w4}$	8.33	11.34	81.05	71.58	80.17	58.44	80.08	**87.88**	76.53
			AWQ_{w4}	8.26	11.40	**82.02**	70.48	81.87	60.56	81.06	80.61	76.10
			$SFMP_{g128}$	**7.95**	**11.13**	82.01	**72.83**	**83.46**	**61.09**	**81.73**	85.20	**77.89**

Table 21: Evaluation of Qwen3 8B/14B/32B models quantized by SFMP, AWQ, SliM-LLM and GPTQ on WikiText-2, C4 perplexity (PPL), and zero-shot tasks. Memory overhead from extra quantization parameters in GPTQ and AWQ at w3, w4 is omitted as it is negligible.