AMQ: Enabling AutoML for Mixed-precision Weight-Only Quantization of Large Language Models

Sangjun Lee[1]* **Seung-taek Woo**[1]* **Jungyu Jin**[1] **Changhun Lee**[2] **Eunhyeok Park**[1]
[1] Graduate School of Artificial Intelligence
[2] Department of Convergence IT Engineering
Pohang University of Science and Technology (POSTECH)
{leesangjun, wst9909, jgjin0317, changhun.lee, eh.park}@postech.ac.kr

Abstract

To enable broader deployment of Large Language Models (LLMs), it is essential to identify the best-performing model under strict memory constraints. We present AMQ, Automated Mixed-Precision Weight-Only Quantization, a framework that assigns layer-wise quantization bit-widths to optimally balance model quality and memory usage. However, the combinatorial search space, with over 10^{100} possible configurations, makes conventional black-box optimization infeasible. AMQ overcomes this challenge through four key innovations: (1) **search space pruning** using prior knowledge to exclude unpromising configurations, (2) **quantization proxy** to bypass costly format conversions during search, (3) **quality predictor** to minimize evaluation overhead, and (4) **iterative search-and-update** strategy for fast and stable convergence. By integrating these components, AMQ efficiently explores the quality-efficiency landscape, reaching the Pareto frontier and yielding LLMs that are both compact and high-performing. Our code is available at https://github.com/dlwns147/amq.

1 Introduction

Weight-only quantization is a powerful optimization technique for large language models (LLMs), significantly reducing memory usage and alleviating performance bottlenecks by lowering the bit-width of model weights. Recent advances (Frantar et al., 2022; Lin et al., 2024; Chee et al., 2024) have demonstrated that even 4-bit quantization can preserve model accuracy. However, pushing below 4 bits often leads to substantial accuracy degradation due to increased quantization error. This raises a critical question: **"Given a fixed memory budget, how can we compress an LLM to achieve the best possible performance?"**

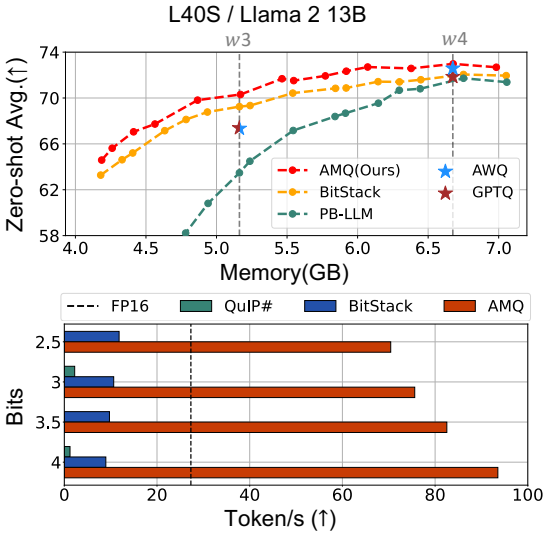

Figure 1: (Top): Trade-off between memory usage and average zero-shot accuracy on ARC-Easy, ARC-Challenge, PIQA, HellaSwag, WinoGrande, and BoolQ. (Bottom): Inference speed. See Section 4 and Appendix C for more details.

Building on this motivation, our study investigates the Pareto frontier of weight-only quantization to maximize model quality under a fixed memory budget. A key observation is that quantization sensitivity varies widely across models, modules, and layers, indicating that fine-grained bit-width allocation can open up a new possible solution, enhancing the quality-efficiency trade-off. However, this flexibility comes at a cost: it dramatically enlarges the configuration space, posing a key yet largely unexplored challenge, **how to efficiently identify the optimal configuration within such a vast search space**.

This problem can be formulated as a discrete combinatorial optimization task, which is known to be NP-complete. A common strategy is to apply black-box optimization methods such as simulated annealing or genetic algorithms (Kirkpatrick et al., 1983; Golberg, 1989). However, these methods are impractical in our context. Due to the massive scale of LLMs, even converting between different bit-

*These authors contributed equally.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 35532–35550
November 4-9, 2025

width formats can take several hours. Moreover, evaluating model quality on a full validation set is computationally expensive, often requiring thousands to millions of iterations per configuration. Consequently, despite aggressive evaluation pruning, black-box approaches remain prohibitively costly and unsuitable for real-world deployment.

To address these limitations, we propose **AMQ**, Automated Mixed-Precision Weight-Only Quantization, a novel framework that automatically identifies the optimal quantization configuration for any given model, maximizing accuracy and execution efficiency within a fixed memory budget, without requiring expert intervention. AMQ is built upon four key innovations: (1) **search space pruning** based on prior knowledge to eliminate unpromising configurations, (2) **quantization proxy** to avoid costly bit-width format conversions during the search, (3) **quality predictor** to reduce evaluation overhead, and (4) **iterative search-and-update** strategy that enables fast and stable convergence. Together, these techniques make black-box optimization tractable, allowing AMQ to discover near-optimal configurations for fine-grained LLMs quantization. Extensive experiments show that AMQ consistently outperforms existing methods in accuracy under the same memory constraints, while also delivering the highest throughput, as illustrated in Figure 1.

2 Related Works

2.1 Weight-only Quantization

Weight-only quantization for LLMs reduces model memory usage, addressing hardware constraints and improving inference speed. GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024) successfully minimized performance degradation while reducing model weights to 3-bit or 4-bit precision. However, these approaches rely on fixed-precision quantization, making them less adaptable to diverse memory environments. The advanced mixed-precision quantization (Huang et al., 2024a; Shang et al., 2023; Lee et al., 2024b; Huang et al., 2024b; Kim et al., 2023; Lee et al., 2024a), which assigns different bit precisions to specific groups or channels within linear layers based on weight distribution, has further improved performance. However, these methods require irregular data access due to mixed precision, making it challenging to achieve actual speedup in real-world deployment.

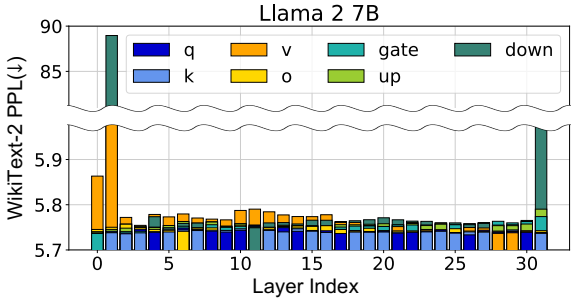

Figure 2: Quantization sensitivity of Llama 2 7B per linear layer with HQQ. Result is the perplexity on the WikiText-2 test set.

2.2 Model Compression with Varying Targets

Recent research has focused on model compression techniques that allow LLMs to be flexibly adjusted to accommodate various memory restrictions. LLM-MQ (Li et al.) and CMPQ (Chen et al., 2024) perform mixed-precision quantization at the linear or channel level to match variable target bits, while sharing the common problem of irregular pattern from mixed precision. BitStack (Wang et al., 2024) stores weights using low-rank decomposition and reconstructs them by loading residual blocks within memory constraints. It is an alternate approach supporting various compression targets, but the weight reconstruction process slows down inference notably.

2.3 AutoML for Neural Architecture Search

Neural architecture search (NAS) is a prominent branch of research within the field of AutoML, offering various solutions to combinatorial optimization problems, which align closely with our own research. To identify neural network architectures within a discrete space that optimize accuracy and performance, numerous studies have explored black-box optimization methods and differentiable approaches (Zoph, 2016; Pham et al., 2018; Real et al., 2019; Tan and Le, 2019). Among these, we drew significant inspiration from research based on neural predictors (Wen et al., 2020; Wan et al., 2020; White et al., 2021). In particular, the quality predictor serves as a key common element that substantially reduces search costs. However, our unique contribution lies in additional ideas that consider the characteristics of quantization, which is also highly important to make AMQ feasible.

3 Method

The primary objective of this work is to make the search for optimal quantization configurations computationally feasible. To effectively navigate the

trade-off between memory and accuracy, we aim to construct a Pareto frontier over various bit-width combinations. NSGA-II (Deb et al., 2002), a genetic algorithm-based method, is well-suited for this task, as it efficiently explores Pareto-optimal solutions across multiple objectives. If search over the configuration space is tractable, one can simply select the most accurate model within the given memory budget from the resulting Pareto frontier. However, in practice, the search process requires hundreds of thousands of quantization/evaluation runs, resulting in prohibitive costs. This section identifies the key sources of this overhead and introduces four core ideas designed to mitigate it.

3.1 Search Space Design

A key step in making the search efficient is to carefully define the configuration space. While finer-grained quantization (e.g., per-weight or per-channel) may offer more optimal configurations, it dramatically increases the number of meaningless candidates and raises the risk of suboptimal convergence. In addition, such granularity often breaks hardware-friendly memory access patterns, leading to significant slowdowns during inference.

To strike a balance between flexibility and efficiency, we adopt **layer-wise bit-width** as the core unit of our search space. This design aligns well with hardware execution and offers sufficient expressiveness for mixed-precision strategies. As shown in Figure 5, our empirical analysis shows that applying different bit-widths within a single linear layer (Huang et al., 2024b) introduces irregular memory access, causing considerable inference latency with only marginal gains in quality. Instead, assigning a single bit-width, chosen from 2, 3, or 4 bits, to each linear layer, using grouped quantization (Lin et al., 2024), provides a good trade-off: it preserves high-quality representations while maintaining a rich search space. For example, Llama 2 7B (Touvron et al., 2023) has 224 linear layers, yielding a search space size of $3^{224} \approx 10^{106}$ possibilities. For the group size of 128, the range of our search space is $[2.25, 4.25]$.

3.2 Space Pruning via Prior Knowledge

However, the vast configuration space contains many low-quality candidates that are unlikely to contribute to an optimal solution. Our analysis reveals substantial variation in quantization sensitivity across linear layers. As shown in Figure 2, individually quantizing a single layer to 2-bit, while keeping all others at 4-bit, results in widely varying degrees of perplexity degradation. This suggests that some layers are highly sensitive and should remain at higher precision (e.g., 4-bit), while others are robust enough to tolerate lower bit-widths.

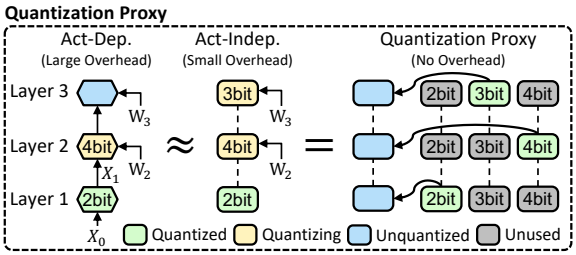

Figure 3: Illustration of activation-dependent vs. activation-independent quantization and Quantization Proxy.

Based on this observation, we propose a simple yet effective strategy to refine the search space by pruning configurations unlikely to yield high-quality results. However, overly aggressive pruning risks eliminating promising candidates. To mitigate this, we adopt a conservative filtering criterion. We first measure per-layer sensitivity using a small calibration set, following the aforementioned procedure. Layers with sensitivity exceeding twice the median are treated as outliers and fixed to 4-bit precision; the remaining layers define the active search space. This model-aware pruning approach substantially reduces search complexity while preserving flexibility. For Llama 2 7B, 13B, and 70B, only 4, 3, and 5 layers are excluded, corresponding to just 1.8%, 1.1%, and 0.9% of all linear layers, respectively. Despite its simplicity, this modification leads to a notable gain, as demonstrated in Section 5.2.

3.3 Quantization Proxy

One of the key challenges in layer-wise mixed-precision search is the high computational cost of generating quantized model representations. For example, the widely used AWQ (Lin et al., 2024) takes approximately 1.5 hours on a single A100 GPU to convert the FP16 weights of a 70B model into 4-bit precision. Additionally, AWQ employs an activation-dependent quantization scheme in which each layer's quantization depends on the activation results of preceding layers. Consequently, every distinct bit-width configuration requires a separate, end-to-end conversion process.

While activation-dependent quantization techniques achieve state-of-the-art model quality, their computational cost makes them impractical for di-

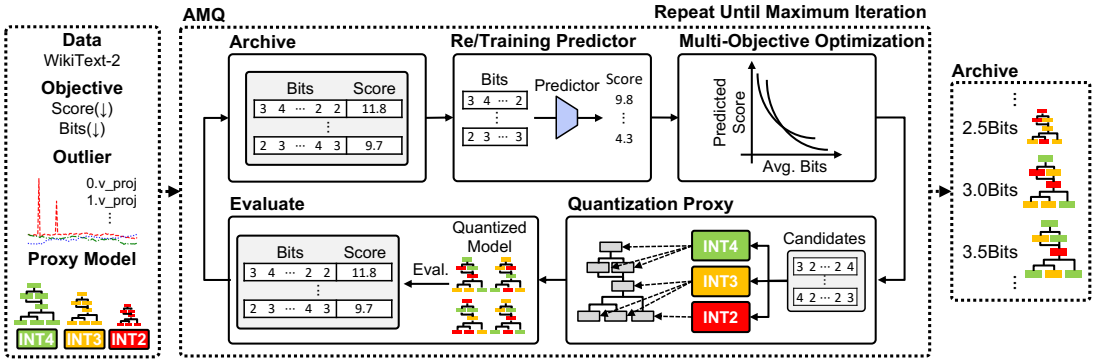

Figure 4: The overview of our AMQ pipeline.

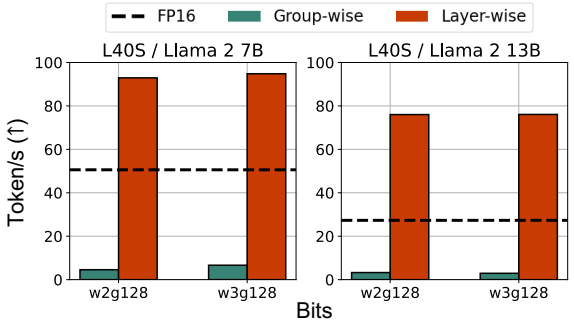

Figure 5: Llama 2 7/13B inference speed of FP16, group-wise mixed-precision quantization (Huang et al., 2024b) and layer-wise quantization on single NVIDIA L40S GPU.

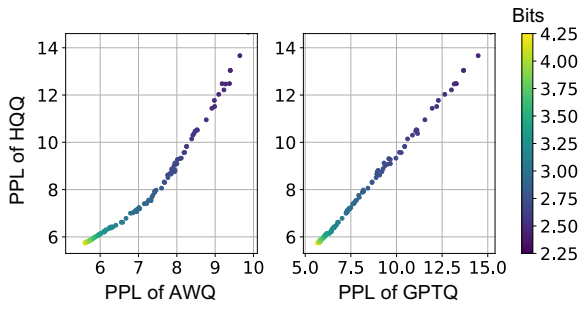

Figure 6: WikiText-2 perplexity (PPL) of HQQ, asymmetric clipping AWQ, and GPTQ on Llama 2 7B, evaluated over a randomly sampled 20% of Pareto frontiers identified by AMQ.

rect use within a search process. To address this limitation, we propose a **quantization proxy**, a lightweight approximation that assembles a quantized model by selecting precomputed versions of each linear layer based on the desired bit-width configuration, as illustrated in Figure 3. The motivation behind this approach is formalized in the following theorem:

Theorem. *Let X be the bit-width search space with size function $S : X \to \mathbb{R}_{>0}$ and injective model-quality scores $Q_1, Q_2 : X \to \mathbb{R}$, where Q_1 corresponds to proxy quantization and Q_2 to activation-dependent quantization. If*

$$Q_1(x) < Q_1(y) \;\Rightarrow\; Q_2(x) < Q_2(y)$$

for all $x, y \in X$, then the Pareto frontier for (Q_1, S) coincides with that for (Q_2, S).

The proof of the theorem is provided in Appendix A. To apply this proxy-based approach, we evaluate several existing quantization methods and identify HQQ (Badri and Shaji, 2023), an activation-independent technique that quantizes weights without requiring activation data, as a strong candidate. As shown in Figure 6, the Pareto frontier obtained using HQQ closely aligns with those derived from activation-dependent methods such as GPTQ and asymmetric clipping AWQ (Gong et al., 2024). This implies that we can search for optimal configurations using HQQ, and then transfer the resulting bit-width assignments to GPTQ or AWQ for final deployment. To leverage the activation-independent nature of HQQ, we precompute each linear layer at 2-, 3-, and 4-bit precision. Given any target configuration, a quantized model can be efficiently assembled by selecting the corresponding precomputed layers. This approach significantly reduces computational overhead while preserving the quality of solutions discovered during the search.

3.4 Quality Predictor

To efficiently guide the search, we assess model quality using logit similarity, based on the intuition that a quantized model retains quality if its output distribution remains close to that of the original. Specifically, we measure the Jensen–Shannon divergence (JSD) between the logits of the quantized model and that of the original model.

Although computing JSD requires only a forward pass, evaluating each quantized model over the full dataset remains expensive, especially when scaling to thousands of configurations. To mitigate this overhead, we introduce a surrogate quality predictor that estimates model performance using

a small set of JSD-labeled samples. We adopt a Radial Basis Function (RBF) model (Baker et al., 2017), which predicts the expected quality of unseen configurations during the search. Leveraging this model, we can rapidly evaluate hundreds of thousands of configurations. However, since the predictions are approximate, the resulting Pareto frontier may contain errors. These inaccuracies are mitigated through the iterative search-and-update process described below, ensuring that model quality is preserved despite the use of the predictor.

3.5 Iterative Search-and-update

Building on the proposed quantization proxy and quality predictor, we apply NSGA-II to search for optimal bit-width configurations. Figure 4 illustrates the complete AMQ pipeline, and the corresponding pseudo-code is provided in Algorithm 1.

The process begins with search space pruning to eliminate unpromising candidates. We then perform random sampling, evaluating hundreds of configurations to initialize the archive, a set of observed samples, which is used to train the initial quality predictor. Next, NSGA-II explores the Pareto front, optimizing two objectives: the model quality (estimated by the quality predictor) and the average bit-width (as a proxy for memory usage).

The candidate solutions found by NSGA-II are then evaluated to obtain their true quality scores, which are used to update the archive. The quality predictor is retrained on this updated archive, and the search process is repeated for a fixed number of iterations. Finally, the best configuration that satisfies the memory constraint is selected.

This iterative pipeline ensures that the predictor is continuously refined using high-quality samples discovered during the search. As a result, both performance estimation and solution quality improve over time, enabling efficient exploration of the quantization search space.

4 Experiments

To validate the superiority of AMQ, we conducted a series of analyses and experiments using the Llama 2 model family (Touvron et al., 2023), ranging in size from 7B to 70B. Also, we further provide the experimental results on Llama 3.1 8B, 70B (Grattafiori et al., 2024), Qwen2.5-7B, 14B (Yang et al., 2024) and Mistral 7B v0.3 (Jiang et al., 2023) in Appendix H. We employed 128 samples from the WikiText-2 train set (Merity et al., 2016) as the calibration set to measure the sensitivity and evaluate the models identified during the search.

To assess the effectiveness of the bit configurations discovered by AMQ, we utilized AWQ (Lin et al., 2024), a well-established weight-only quantization method, following its original settings except for asymmetric clipping (Gong et al., 2024). AMQ utilized a group size of 128 for search and performance evaluation. We further compared AMQ with PB-LLM (Shang et al., 2023) and BitStack (Wang et al., 2024) under varying memory constraints, and also with fixed-precision quantization methods, GPTQ and AWQ.

We evaluate the optimized models by reporting language-modeling perplexity (PPL) on the WikiText-2 test set and the C4 validation set (Raffel et al., 2020). For zero-shot performance, we use the LM-Evaluation-Harness (Gao et al., 2024) on six benchmarks: ARC-Challenge, ARC-Easy (Clark et al., 2018), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019), and WinoGrande (Sakaguchi et al., 2021). To probe more demanding tasks, we further evaluate 5-shot MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021). We also report inference throughput as the median tokens per second when generating 128 tokens with batch size 1. Additional details are provided in Appendix C.

4.1 AMQ vs. Any-Size Compression

Table 1 shows the perplexity and zero-shot task performance of the optimized models under varying memory budgets and models using AMQ, BitStack, and PB-LLM. AMQ consistently outperforms BitStack across multiple bit-width settings and model scales. At extremely low precision (e.g., 2.5 bits), AMQ achieves the best average zero-shot accuracy among all methods. Notably, for the 70B model at 3.5 bits, AMQ retains up to 99.18% of the FP16 model's average zero-shot performance. Even with 0.5 fewer bits, AMQ matches or exceeds the performance of the 3.5 bits baselines over different model scales, demonstrating its robustness. As shown in Figure 1, AMQ consistently outperforms baselines across all bit levels. Moreover, Table 2 shows that AMQ consistently outperforms BitStack on 5-shot MMLU and GSM8K across models, demonstrating superior performance on challenging tasks. This highlights the efficiency and generalization capability of our search strategy.

Model	Mem. (MB)	Avg. Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
7B	12,853	16	FP16	5.47	7.26	74.58	46.25	79.11	76.00	69.22	77.77	70.49
	2,431	2.5	PB-LLM	24.53	32.05	37.50	23.04	58.00	34.25	51.85	61.77	44.40
			BitStack	**8.92**	**12.09**	55.56	33.62	72.31	61.85	**63.77**	**72.35**	59.91
			AMQ	9.24	12.37	**58.88**	**36.86**	**73.50**	**65.01**	62.75	66.39	**60.56**
	2,817	3.0	PB-LLM	11.60	14.81	53.20	29.10	70.02	53.82	61.72	71.31	56.53
			BitStack	7.46	10.13	62.16	37.63	74.81	66.96	66.38	**73.82**	63.63
			AMQ	**6.83**	**9.03**	**68.22**	**41.72**	**76.55**	**71.27**	**67.32**	68.44	**65.59**
	3,203	3.5	PB-LLM	7.90	10.40	62.75	36.60	74.92	65.43	67.80	**77.25**	64.12
			BitStack	6.72	9.04	64.06	40.44	76.17	69.61	67.88	75.11	65.54
			AMQ	**5.95**	**7.90**	**71.55**	**44.20**	**77.86**	**73.92**	**69.06**	73.88	**68.41**
13B	24,826	16	FP16	4.88	6.73	77.53	49.15	80.52	79.37	72.30	80.55	73.23
	4,408	2.5	PB-LLM	32.70	42.50	42.05	24.15	61.48	35.95	53.28	62.39	46.55
			BitStack	7.46	10.13	**67.89**	38.23	75.73	66.89	67.25	75.26	65.21
			AMQ	**6.88**	**9.46**	67.38	**40.19**	**77.09**	**71.11**	**69.38**	**77.16**	**67.05**
	5,164	3.0	PB-LLM	9.57	13.30	63.55	37.71	75.46	62.77	68.27	73.09	63.48
			BitStack	6.33	8.73	**74.37**	44.37	77.26	71.93	69.46	78.10	69.25
			AMQ	**5.68**	**7.80**	72.18	**45.39**	**78.35**	**76.02**	**70.32**	**79.57**	**70.31**
	5,920	3.5	PB-LLM	6.79	9.44	68.48	41.81	77.86	71.35	71.98	**80.58**	68.68
			BitStack	5.76	7.93	75.67	45.56	78.62	73.61	71.19	**80.58**	70.87
			AMQ	**5.20**	**7.13**	**75.84**	**48.89**	**80.25**	**77.67**	**72.22**	79.14	**72.34**
70B	131,591	16	FP16	3.32	5.71	81.02	57.25	82.70	83.78	77.98	83.79	77.75
	21,403	2.5	BitStack	4.91	7.43	76.73	51.11	79.82	77.25	**75.93**	76.70	72.92
			AMQ	**4.90**	**7.25**	**77.02**	**51.28**	**80.03**	**77.84**	75.61	**80.73**	**73.75**
	25,483	3.0	BitStack	4.34	6.69	**78.83**	54.78	81.56	79.81	**76.72**	80.95	75.44
			AMQ	**4.01**	**6.29**	78.54	**55.80**	**81.77**	**81.31**	75.45	**83.46**	**76.05**
	29,563	3.5	BitStack	3.95	6.30	79.71	56.23	81.99	81.06	**77.51**	82.48	76.50
			AMQ	**3.62**	**5.93**	**79.80**	**57.68**	**82.21**	**82.80**	77.19	**82.97**	**77.11**

Table 1: Evaluation of Llama 2 7B/13B/70B models compressed by AMQ, BitStack and PB-LLM at average bit widths of 2.5, 3.0, and 3.5, showing WikiText-2 and C4 dataset perplexity (PPL) alongside zero-shot tasks accuracy. PB-LLM is excluded due to lack of 70B model support.

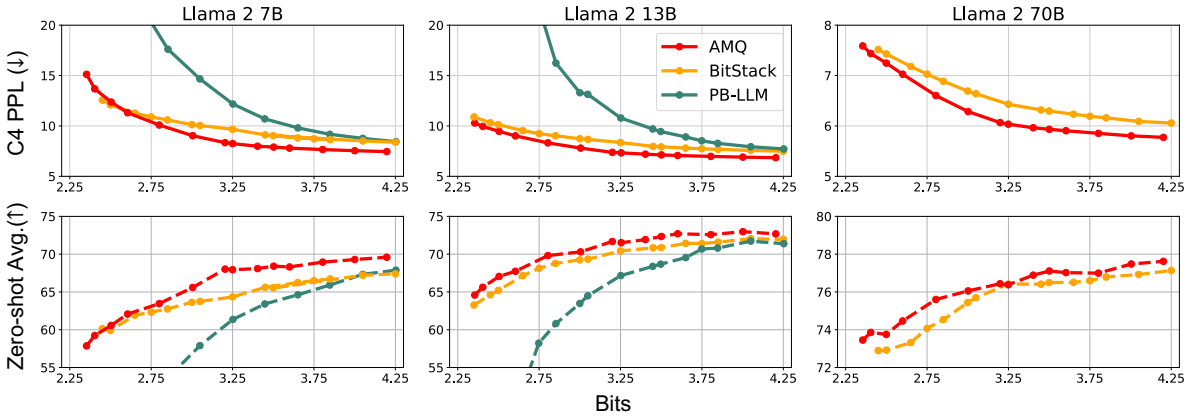

Figure 7: Trade-off between the average zero-shot accuracy and average bit-width over AMQ, BitStack and PB-LLM.

4.2 Inference Acceleration

As illustrated in Figure 8, weight decomposition-based compression methods (BitStack) suffer from the overhead of weight reconstruction during inference. In contrast, AMQ customizes kernels for each linear layer based on its bit configuration, resulting in up to 2.67× speedup compared to FP16 on the L40S GPU for Llama 2 7B. For Llama 2 13B, AMQ achieves an even higher speedup of 3.16×. Moreover, AMQ delivers high-speed inference while maintaining a small memory footprint, making it particularly effective in memory-constrained environments such as the RTX 3090.

4.3 AMQ vs. Fixed-Precision Quantization

We compared the performance of AMQ with existing quantization methods, GPTQ and AWQ, which using iso-precision over all layers. Table 3 presents the perplexity and average zero-shot accuracy across bit-widths ranging from 2.25 to 4 bits. AMQ consistently matches or surpasses uniform quantization, validating the effectiveness of its discovered bit configurations. For Llama 2 7B, AMQ maintains stable accuracy at 2.35 bits, while GPTQ and AWQ degrade sharply at 2.25 bits. It remains robust at 3 bits and competitive even at 4 bits across model sizes. On Llama 2 13B, AMQ

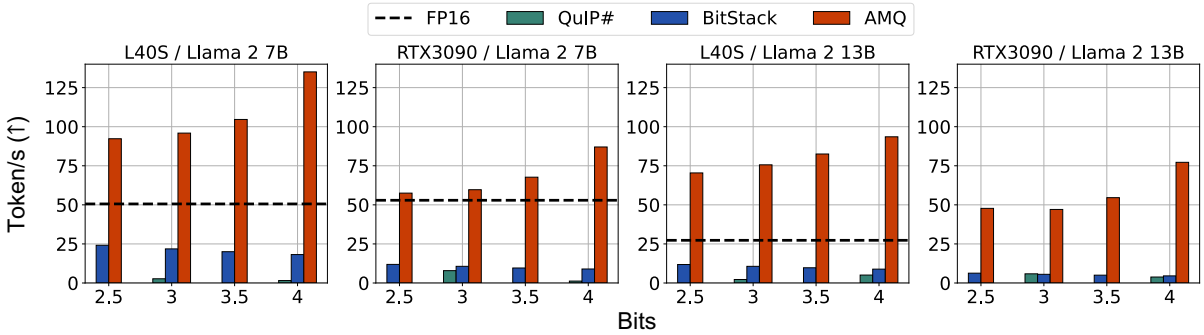

Figure 8: Inference speeds at various average bit-widths. Llama 2 13B (FP16) exceeds the VRAM capacity of a single RTX 3090.

Model	Mem. (MB)	Avg. Bits	Method	MMLU	GSM8K
Mistral 7B v0.3	13,825	16	FP16	62.40	36.32
	2,593	2.5	BitStack	34.60	3.18
			AMQ	**45.41**	**3.49**
	3,009	3	BitStack	44.87	10.70
			AMQ	**53.57**	**16.00**
	3,425	3.5	BitStack	52.70	17.29
			AMQ	**59.12**	**27.22**
	3,841	4	BitStack	54.20	22.44
			AMQ	**61.12**	**29.64**
Llama 3.1 8B	15,317	16	FP16	65.20	50.57
	4,085	2.5	BitStack	30.63	1.59
			AMQ	**32.10**	**2.35**
	4,501	3	BitStack	44.03	4.55
			AMQ	**52.78**	**19.86**
	4,917	3.5	BitStack	52.35	11.68
			AMQ	**60.68**	**31.92**
	5,333	4	BitStack	57.18	20.85
			AMQ	**62.68**	**40.41**
Qwen2.5 14B	28,171	16	FP16	79.85	87.72
	6,909	2.5	BitStack	60.24	34.95
			AMQ	**63.65**	**36.69**
	7,697	3	BitStack	67.66	65.73
			AMQ	**71.61**	**73.39**
	8,484	3.5	BitStack	72.55	72.02
			AMQ	**76.93**	**80.52**
	9,272	4	BitStack	74.43	79.30
			AMQ	**78.69**	**83.24**

Table 2: 5-shot MMLU, GSM8K task results over Mistral 7B v0.3, Llama 3.1 8B and Qwen2.5 14B.

achieves 96.01% of FP16 average zero-shot accuracy with only 3 bits, demonstrating strong memory efficiency without sacrificing performance.

5 Analysis

5.1 Search and Compression Cost

We compare the overall algorithmic costs of AWQ, OmniQuant (Shao et al., 2024), and BitStack with AMQ in Table 4. The comparison includes both search time, required for exploring memory-performance trade-offs, and compression time, needed to generate optimized models, using the Llama 2 family on NVIDIA A100-80GB GPUs. AWQ and OmniQuant incur no search overhead but are constrained to fixed-bit quantization, limiting flexibility. OmniQuant also requires fine-tuning, increasing its compression cost. BitStack performs only one compression run by weight decomposition but suffers from significantly higher search

Model	Mem. (MB)	Avg. Bits	Method	Wiki(↓)	C4(↓)	Avg.(↑)
7B	12,853	16	FP16	5.47	7.26	70.49
	2,238	2.25	GPTQ_{w2g128}	61.77	44.10	43.19
			AWQ_{w2g128}	2.22e5	1.68e5	36.12
	2,315	2.35	AMQ	**11.49**	**15.12**	**57.86**
	2,817	3.0	GPTQ_{w3}	9.27	11.81	60.70
			AWQ_{w3}	15.45	17.44	54.67
			AMQ	**6.83**	**9.03**	**65.59**
	3,010	3.25	GPTQ_{w3g128}	6.45	8.53	67.22
			AWQ_{w3g128}	6.25	8.30	67.63
			AMQ	**6.20**	**8.25**	**67.94**
	3,589	4.0	GPTQ_{w4}	6.09	7.86	68.55
			AWQ_{w4}	5.83	7.72	69.10
			AMQ	**5.68**	**7.54**	**69.29**
13B	24,826	16	FP16	4.88	6.73	73.23
	4,029	2.25	GPTQ_{w2g128}	27.78	23.39	45.64
			AWQ_{w2g128}	1.22e5	9.55e4	40.59
	4,181	2.35	AMQ	**7.60**	**10.29**	**64.59**
	5,164	3.0	GPTQ_{w3}	6.75	8.96	67.39
			AWQ_{w3}	6.45	9.07	67.34
			AMQ	**5.68**	**7.80**	**70.31**
	5,542	3.25	GPTQ_{w3g128}	5.48	7.49	70.89
			AWQ_{w3g128}	**5.32**	**7.31**	**72.11**
			AMQ	5.36	7.33	71.52
	6,676	4.0	GPTQ_{w4}	5.19	7.06	71.85
			AWQ_{w4}	5.06	6.96	72.59
			AMQ	**5.03**	**6.91**	**72.98**
70B	131,563	16	FP16	3.32	5.71	77.75
	19,363	2.25	GPTQ_{w2g128}	8.33	10.71	59.85
			AWQ_{w2g128}	7.25e4	6.56e4	40.54
	20,179	2.35	AMQ	**5.17**	**7.59**	**73.45**
	25,483	3.0	GPTQ_{w3}	4.88	7.11	73.31
			AWQ_{w3}	4.36	6.63	75.10
			AMQ	**4.01**	**6.29**	**76.05**
	27,523	3.25	GPTQ_{w3g128}	3.88	6.11	**76.64**
			AWQ_{w3g128}	3.74	6.04	76.58
			AMQ	**3.73**	**6.03**	76.38
	33,643	4.0	GPTQ_{w4}	3.59	5.90	77.07
			AWQ_{w4}	3.48	5.84	77.41
			AMQ	**3.46**	**5.80**	**77.48**

Table 3: Evaluation of Llama 2 7B/13B/70B models quantized by AMQ, AWQ, and GPTQ on WikiText-2, C4 perplexity (PPL), and zero-shot tasks. For 2.25-bit settings, our method matches asymmetric clipping in AWQ; thus, we report results with an additional 0.1 bits. Memory overhead from extra quantization parameters in GPTQ and AWQ at w3, w4 is omitted as it is negligible. Detailed zero-shot accuracy is provided in Table 14.

Model	7B				13B				70B			
Type	Search		Compression		Search		Compression		Search		Compression	
Parameter	#GPU	Cost (h)	#GPU	Cost (h)	#GPU	Cost (h)	#GPU	Cost (h)	#GPU	Cost (h)	#GPU	Cost (h)
AWQ	-	-	1	0.15	-	-	1	0.28	-	-	1	1.5
OmniQuant	-	-	1	1.24	-	-	1	2.2	-	-	1	10
BitStack	1	10	1	2.25	1	>72	1	4.25	2	>300	1	23
AMQ	1	5	1	0.15	1	8	1	0.28	2	44	1	1.5

Table 4: The search and compression time on Llama 2 family of AWQ, OmniQuant, BitStack, AMQ.

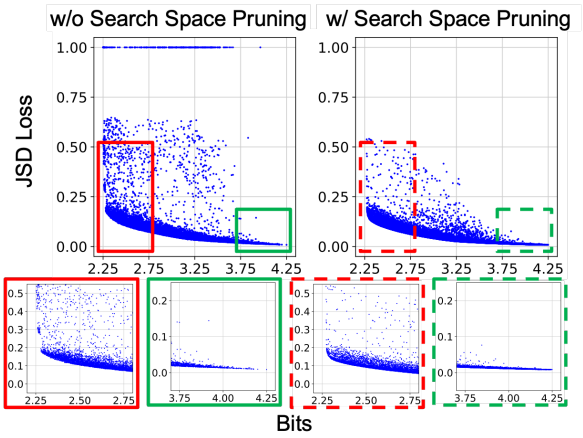

Figure 9: Total search samples on Llama 2 70B with vs. without Search Space Pruning.

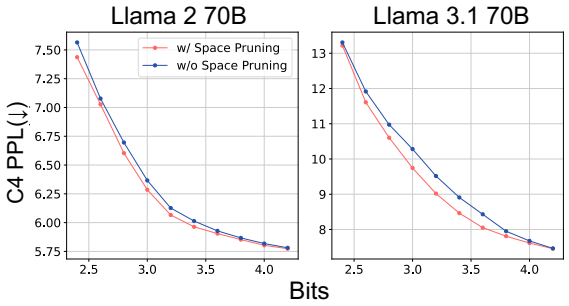

Figure 10: C4 perplexity of Llama 2 and 3.1 70B, with and without search space pruning.

and compression time due to block evaluation and sorting. AMQ achieves practical search costs by avoiding type conversion overhead through a quantization proxy and accelerating convergence via a quality predictor and search space pruning. For instance, on Llama 2 7B, AMQ evaluates only 10,250 candidates directly, while predicting performance for over 800,000 samples, despite the vast search space size (approx. 10^{100}).

5.2 Effect of Search Space Pruning

Impact on Vast Space. We assess the effect of pruning on large search space by comparing results with and without it. As shown in Figure 10, pruning markedly improves search quality for Llama 2 70B and Llama 3.1 70B. Without pruning, the search fails to explore configurations near 4.25 bits at Llama 2 70B, as those regions remain entirely unvisited, illustrated in Figure 9. This indicates that outlier linear layers destabilize training and inject

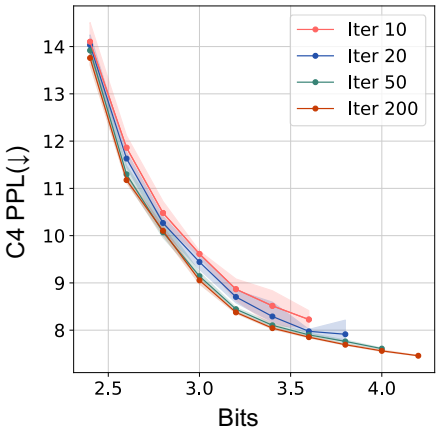

Figure 11: C4 perplexity variation of Pareto frontiers across iterations on Llama 2 7B with six random seeds. Data points are plotted only when all seeds discover a sample for the corresponding bit-width at a given iteration.

noise into quality predictions, steering the search toward suboptimal points, especially for large scale models.

Ablation on Threshold and Calibration Set. We investigate how the selection of outlier layers is affected by the calibration set and threshold. Note that we set the sensitivity threshold conservatively to prevent excluding too many candidates and limiting the expressiveness of the search space. Table 5 shows that sensitive layers consistently occur in early V linear layers of self-attention and early/late Down linear layers of MLP, regardless of calibration set. The excluded fraction remains small (0.45–2.14%) with negligible impact on C4 perplexity, and AMQ converges stably as long as the threshold is not overly strict. We thus adopt twice the median threshold, which yields consistently robust performance.

5.3 Robustness over Random Seed

Figure 11 illustrates the perplexity variations of the Pareto frontiers on the C4 validation set across different bit-widths at iterations 10, 20, 50, and 200 (final) during AMQ search on the Llama 2 7B model with six random seeds. The results highlight the robustness of the search process, which consistently converges to an optimal Pareto frontier while

Model	Dataset	Threshold (×median)	Outlier Layer	# Outlier Layer	2.5-bit	3-bit	3.5-bit	4-bit
Mistral 7B v0.3	WikiText-2 C4	1.5	V: [3], Down: [1, 31]	3 (1.34%)	**12.61**	10.05	9.04	**8.73**
	WikiText-2	2	Down: [1]	1 (0.45%)	12.67	10.08	**9.03**	8.74
	C4		Down: [1, 31]	2 (0.89%)	12.87	**9.97**	**9.03**	**8.73**
	WikiText-2 C4	3	Down: [1]	1 (0.45%)	12.67	10.08	**9.03**	8.74
	WikiText-2 C4	5						
Llama 2 13B	WikiText-2 C4	1.5	V: [0, 1, 2], Down: [0, 3, 39]	6 (2.14%)	9.43	7.78	**7.13**	**6.91**
	WikiText-2	2	V: [0], Down: [0, 3]	3 (1.07%)	9.39	7.77	**7.13**	**6.91**
	C4		V: [0, 1], Down: [0, 3]	4 (1.43%)	**9.38**	**7.75**	**7.13**	**6.91**
	WikiText-2 C4	3	V: [0], Down: [0, 3]	3 (1.07%)	9.39	7.77	**7.13**	**6.91**
	WikiText-2	5	Down: [0, 3]	2 (0.71%)	9.45	7.78	7.15	**6.91**
	C4		V: [0], Down: [0, 3]	3 (1.07%)	9.46	7.80	**7.13**	**6.91**

Table 5: C4 Perplexity and selected outlier linear layers over different calibration sets and sensitivity thresholds for Search Space Pruning. The default is WikiText-2 and 2 × median. Layer indices start at 0. We employ 32 samples from the C4 calibration set to approximate the token count of 128 samples from WikiText-2.

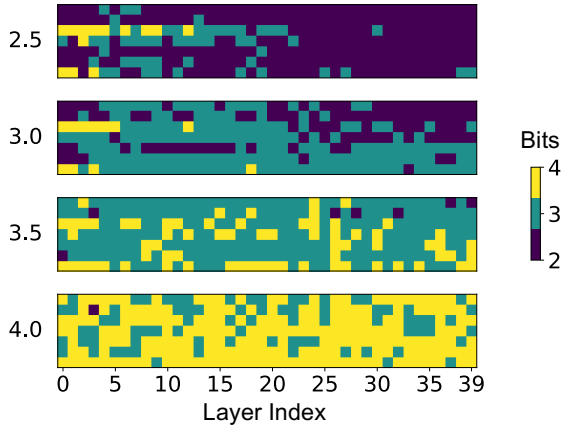

Figure 12: Visualization of bit allocation over linear layers with different average bits at Llama 2 13B. Each row in each box represents Q, K, V, O, Gate, Up, and Down. The numbers on the left indicate the average bits per configuration.

progressively reducing variation at each iteration, regardless of the initial random seed.

5.4 Visualization of Bit Allocation

Figure 12 shows bit allocation across linear layers for models quantized to average bit-widths of 2.5, 3.0, 3.5, and 4.0. As bit-width decreases, Query and Key layers in self-attention are prioritized for lower bit-widths, followed by the Gate layer in the MLP. Notably, the Value layer in self-attention consistently retains a higher bit-width, underscoring its critical role in preserving model performance.

6 Conclusion

In this paper, we propose AMQ, an automated mixed-precision weight-only quantization framework designed to achieve optimal model quality under memory constraints. Our approach precisely defines the search space, and by selectively excluding low-bit-sensitive outlier layers, we effectively prune the initial search space, enhancing both convergence speed and search quality. Additionally, we leverage a quantization proxy to generate quantized models rapidly. Finally, we introduce a quality predictor that estimates the performance of unseen bit-width combinations, significantly reducing the evaluation overhead during the search process. Our experimental results demonstrate that AMQ efficiently allocates bit-widths to each linear layer, even at lower precision levels, outperforming existing baselines while effectively addressing real-world constraints.

Limitations

This study presents an efficient approach to exploring the search space using a quality score predictor, a genetic algorithm, and search space shrinking to address the NP-complete problem of optimal bit configuration in memory-constrained environments. However, alternative solvers for NP-complete problems may yield better configurations, which will be investigated in future work.

The current method focuses on weight-only quantization, addressing primary memory constraints in LLMs. Future research will extend this to tackle computation-bound challenges through activation quantization and explore rotation-based techniques to further improve performance and efficiency.

Acknowledgements

This work was supported by IITP and NRF grant funded by the Korea government(MSIT) (No. RS-2023-00213611, RS-2024-00457882, RS-2024-00396013) and Samsung Advanced Institute of Technology.

References

AutoGPTQ. 2025. Accessed 16-02-2025. [link].

Hicham Badri and Appu Shaji. 2023. Half-quadratic quantization of large machine learning models.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

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

Rodrigo de Carvalho, Rodney Rezende Saldanha, BN Gomes, Adriano Chaves Lisboa, and AX Martins. 2012. A multi-objective evolutionary algorithm based on decomposition for optimal design of yagi-uda antennas. *IEEE Transactions on Magnetics*, 48(2):803–806.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2024. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36.

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

Xiangxiang Chu, Bo Zhang, and Ruijun Xu. 2020. Multi-objective reinforced evolution in mobile neural architecture search. In *european conference on computer vision*, pages 99–113. Springer.

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. *arXiv preprint arXiv:1905.10044*.

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, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint 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.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

David E Golberg. 1989. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102):36.

Ruihao Gong, Yang Yong, Shiqiao Gu, Yushi Huang, Chengtao Lv, Yunchen Zhang, Dacheng Tao, and Xianglong Liu. 2024. Llmc: Benchmarking large language model quantization with a versatile compression toolkit. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 132–152.

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

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

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Xianglong Liu, Luca Benini, Michele Magno, and Xiaojuan Qi. 2024b. Slim-llm: Salience-driven mixed-precision quantization for large language models.

Himanshu Jain and Kalyanmoy Deb. 2013. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. *IEEE Transactions on evolutionary computation*, 18(4):602–622.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao,

Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

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

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science*, 220(4598):671–680.

Changhun Lee, Jun-gyu Jin, Younghyun Cho, and Eunhyeok Park. 2024a. Qeft: Quantization for efficient fine-tuning of llms. *arXiv preprint arXiv:2410.08661*.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024b. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13355–13364.

Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. Llm-mq: Mixed-precision quantization for efficient llm deployment.

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.

Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. 2020. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, 73:102989.

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the genetic and evolutionary computation conference*, pages 419–427.

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

NVIDIA. 2025a. Accessed 16-02-2025. [link].

NVIDIA. 2025b. Accessed 16-02-2025. [link].

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR.

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.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

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.

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. 2023. Pb-llm: Partially binarized large language models. *arXiv preprint arXiv:2310.00034*.

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

Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. 2020. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12965–12974.

Xinghao Wang, Pengyu Wang, Bo Wang, Dong Zhang, Yunhua Zhou, and Xipeng Qiu. 2024. Bitstack: Fine-grained size control for compressed large language models in variable memory environments. *arXiv preprint arXiv:2410.23918*.

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer.

Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. 2021. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems*, 34:28454–28469.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

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

B Zoph. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

A Proof of Motivation in Section 3.3

Since Q_1 and Q_2 are injective, they induce strict total orders on X. The condition implies order equivalence for all $x, y \in X$:

$$Q_1(x) < Q_1(y) \iff Q_2(x) < Q_2(y).$$

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ denote the Pareto-frontiers for (Q_1, S) and (Q_2, S), respectively. Suppose $a \in \mathcal{F}_1$ but $a \notin \mathcal{F}_2$. Then there exists $b \in X$ such that $Q_2(b) > Q_2(a)$ and $S(b) \leq S(a)$, which by order equivalence implies $Q_1(b) > Q_1(a)$, contradicting $a \in \mathcal{F}_1$. Hence $\mathcal{F}_1 \subseteq \mathcal{F}_2$. The same argument with Q_1 and Q_2 swapped yields $\mathcal{F}_2 \subseteq \mathcal{F}_1$. Therefore, $\mathcal{F}_1 = \mathcal{F}_2$.

B Detailed Algorithm

Algorithm 1 Auto Weight Quantization Search

Require: $\mathcal{S}$: Search space, Q_2, Q_3, Q_4: Proxy models for 2-bit, 3-bit, and 4-bit quantization, $\mathcal{D}$: Calibration dataset, N: # of initial samples, I: # of iterations, B: Target bits

$\mathcal{S} \leftarrow \text{SpaceShrink}(\mathcal{S}, \mathcal{D})$
$\mathcal{A} \leftarrow \emptyset$ ▷ Initialize archive
for $i = 1$ to N **do** ▷ Initial sampling
  $\alpha \leftarrow \text{RandomSample}(\mathcal{S})$
  $Q_\alpha \leftarrow \text{Assemble}(\alpha, Q_2, Q_3, Q_4)$
  $\text{Score} \leftarrow \text{Evaluate}(Q_\alpha, \mathcal{D})$
  $\mathcal{A} \leftarrow \mathcal{A} \cup (\alpha, \text{Score})$
end for
for $j = 1$ to I **do** ▷ Iterative search
  $\mathcal{P} \leftarrow \text{Re/TrainPredictor}(\mathcal{A})$
  $\text{front} \leftarrow \text{ParetoSort}(\mathcal{A})$
  $\bar{\alpha} \leftarrow \text{NSGA-II}(\text{front}, \mathcal{P})$
  for $\alpha \in \bar{\alpha}$ **do** ▷ Verify candidates
    $Q_\alpha \leftarrow \text{Assemble}(\alpha, Q_2, Q_3, Q_4)$
    $\text{Score} \leftarrow \text{Evaluate}(Q_\alpha, \mathcal{D})$
    $\mathcal{A} \leftarrow \mathcal{A} \cup (\alpha, \text{Score})$
  end for
end for
$\alpha^* \leftarrow \text{SelectOptimal}(\mathcal{A}, B)$
return α^*

C Detailed Experimental Setting

All experiments were run on up to two NVIDIA A100-80GB GPUs. For AMQ, for each target bit precision we evaluated the candidate whose average bit-width lay within ±0.005 of the target and achieved the lowest score. During Search Space Pruning, we quantified each linear layer's sensitivity using the JSD score and the same calibration set used for model-quality evaluation within Iterative search-and-update process. For PB-LLM, we set the group size to 128, counting only weight memory and excluding any additional indexing overhead. For BitStack, we used the official pre-trained weights from their implementation.

In all inference speed experiments including FP16, QuIP#, and BitStack, we leveraged the multi-head attention and layer normalization kernels with FasterTransformer (NVIDIA, 2025a). For the 4-bit linear layers, AMQ employed TensorRT-LLM (NVIDIA, 2025b)-based kernels, while for the 2-bit and 3-bit linear layers, AMQ utilized AutoGPTQ (AutoGPTQ, 2025) kernels.

The hyperparameters used for searching the Llama 2 family in our experiments are detailed in Table 6.

Hyper-parameter	Model		
	7B	13B	70B
Search Iteration	200	200	250
NSGA-II Candidate	50	50	50
Pretraining Data	250	300	600
NSGA-II Pop	200	200	200
NSGA-II Iteration	20	20	20
Cross-over Probability	0.9	0.9	0.9
Mutation Probability	0.1	0.1	0.1
Subset Pop	100	100	100

Table 6: Hyper-parameters of our algorithm used in Llama 2 search space.

D Robustness over NSGA-II Hyperparameter

Average Bits	Crossover Prob.	Wiki(↓)	C4(↓)
2.5	0.5	9.26	**12.32**
	0.7	**9.19**	**12.32**
	0.9	9.24	12.37
3	0.5	**6.78**	**9.03**
	0.7	6.84	9.07
	0.9	6.83	**9.03**
3.5	0.5	5.93	7.89
	0.7	**5.92**	**7.88**
	0.9	5.95	7.90
4	0.5	**5.67**	**7.54**
	0.7	5.68	**7.54**
	0.9	5.68	**7.54**

Table 7: Evaluation of different crossover probabilities over Llama 2 7B. Our default option is 0.9.

We assess the robustness of the search process with respect to variations in NSGA-II hyperparameters, specifically the crossover and mutation prob-

Average Bits	Mutation Prob.	Wiki(↓)	C4(↓)
2.5	0.01	**9.18**	12.23
	0.05	9.25	**12.22**
	0.1	9.24	12.37
	0.2	9.23	12.26
	0.3	9.26	12.31
3	0.01	6.90	9.07
	0.05	**6.80**	9.03
	0.1	6.83	9.03
	0.2	6.84	**8.98**
	0.3	6.83	9.06
3.5	0.01	**5.93**	**7.88**
	0.05	5.98	7.91
	0.1	5.95	7.90
	0.2	5.94	7.89
	0.3	5.95	7.90
4	0.01	5.68	**7.54**
	0.05	5.70	7.56
	0.1	5.68	**7.54**
	0.2	5.69	**7.54**
	0.3	**5.67**	**7.54**

Table 8: Evaluation of different mutation probabilities over Llama 2 7B. Our default option is 0.1.

abilities. Table 7 and Table 8 present the results for the Llama 2 7B model under different settings. The results demonstrate that AMQ consistently maintains strong performance across a wide range of NSGA-II configurations, highlighting the robustness of the method. As no single setting shows clear superiority, we arbitrarily select one for our experiments.

E Choice of Search Method, Quantization Proxy, Predictor, and Iteration

Memory (MB)	Avg. Bits	Predictor	Wiki(↓)	C4(↓)
2,431	2.5	MLP	**9.24**	**12.24**
		RBF	**9.24**	12.37
2,817	3	MLP	**6.83**	9.07
		RBF	**6.83**	**9.03**
3,203	3.5	MLP	**5.93**	**7.89**
		RBF	5.95	7.90
3,589	4	MLP	**5.68**	7.55
		RBF	**5.68**	**7.54**

Table 9: Evaluation of MLP/RBF predictor over Llama 2 7B.

Search Method. NSGA-II is widely used in space exploration studies (Loni et al., 2020; Lu et al., 2019; Chu et al., 2020) as a standard multi-objective genetic algorithm. While other approaches such as NSGA-III (Jain and Deb, 2013) and MOEAD (Carvalho et al., 2012) exist, they often extend NSGA-II or require additional hyperparameters, such as reference directions. We therefore adopt NSGA-II for its simplicity and effectiveness.

Model	Iteration	Time (h)	2.5-bit	3-bit	3.5-bit	4-bit
7B	100	2	12.34	9.10	7.90	7.55
	200	5	12.37	9.03	7.90	7.54
	300	11	12.32	9.06	7.89	7.53
	400	21	12.32	9.06	7.90	7.53
13B	100	3	9.43	7.83	7.16	6.92
	200	8	9.39	7.77	7.13	6.91
	300	16	9.39	7.79	7.14	6.90
	400	29	9.39	7.76	7.13	6.90

Table 10: Search cost and C4 validation set perplexity over different numbers of search iterations of Llama 2 7B/13B. The default iteration for Llama 2 7B/13B is 200.

Model	# Layer	Iteration	Spearman's Rho	Kendall's Tau
Llama 3.1 8B	224	10	0.998	0.984
		50	1.000	1.000
		100	1.000	1.000
		200	1.000	1.000
Qwen2.5 14B	336	10	0.999	0.991
		50	1.000	1.000
		100	1.000	1.000
		200	1.000	1.000
Llama 3.1 70B	560	10	0.993	0.976
		50	1.000	0.998
		100	1.000	1.000
		250	1.000	1.000

Table 11: Rank correlation of predictions and true values at different iterations.

Although single-objective optimization methods, such as genetic algorithms, reinforcement learning, and policy gradients, may be suitable when optimizing for accuracy at a fixed average bit-width, our goal is to identify the Pareto frontier. This makes NSGA-II the most appropriate choice for our setting.

Quantization Proxy. For the quantization proxy, we considered data-independent methods like RTN. However, they either yield lower performance or offer no clear advantage in correlation compared to HQQ. Thus, we adopt HQQ for its stronger alignment with actual performance metrics.

Predictor. We also explored alternative predictor models, including multilayer perceptrons (MLPs), classification and regression trees (CART), and Gaussian processes. Due to slower training or negligible performance improvements over radial basis function (RBF) models, we selected RBF predictors for their efficiency. Table 9 shows a comparison of AMQ using MLP and RBF predictors on the Llama 2 7B model, revealing minimal performance differences between the two.

Table 11 reports the rank correlation between RBF predictor outputs and ground truth during the search on Llama 3.1 8B/70B and Qwen2.5 14B. Since search space pruning removes outlier layers, no outlier samples are used. The RBF predictor achieves strong accuracy, with high Kendall's Tau and Spearman's Rho, which we attribute to the low

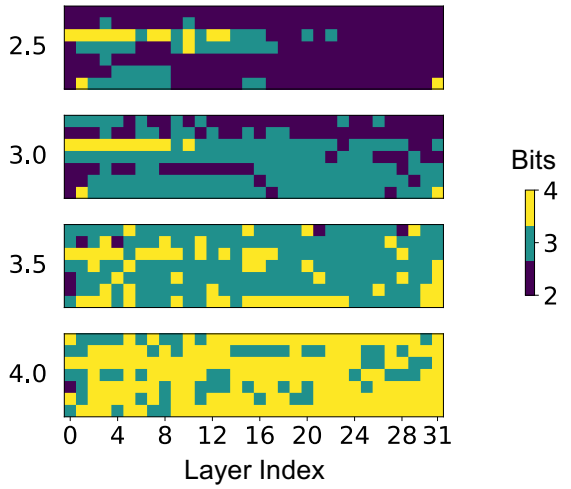

Figure 13: Visualization of bit allocation over linear layers with different average bits at Llama 2 7B. Each row in each box represents Q, K, V, O, Gate, Up, and Down. The numbers on the left indicate the average bits per configuration.

noise in the high-dimensional inputs. The iterative search-and-update strategy further enhances performance near the Pareto frontier by updating multiple candidate points at once, enabling more stable and reliable modeling.

Iteration. Table 10 reports C4 perplexity and search time for Llama 2 7B/13B under varying iterations. Our search-and-update procedure considers only Pareto-superior candidates, ensuring stable convergence. However, increasing the iteration limit substantially raises costs, primarily due to predictor training and search, while offering negligible performance improvements. Accordingly, we set the iteration limit to 200 for models up to 30B and 250 for larger models. Nevertheless, when resources or time are limited, using only 100 iterations still produces competitive models, demonstrating that AMQ is a practical rather than time-consuming method.

F Additional Visualization of Bit Allocation

Figure 13 and Figure 14 visualize the bit configuration on various bit-widths searched from AMQ with Llama 2 7B/70B.

G Comparison with Other Discrete Structure Search Methods

Given the vast search space of all possible bit-width assignments for Llama 2 7B ($3^{224} \approx 10^{106}$), exhaustive grid search is infeasible within a practical time. For this reason, we propose two lightweight search methods.

Method	Cost (h)	
	7B	13B
One-shot search	0.1	0.3
Greedy search	10	43
AMQ	7	16

Table 12: Cost of one-shot search and greedy search over Llama 2 7B/13B on single NVIDIA RTX6000-ADA.

- **One-shot search.** Layers are first ranked by JSD sensitivity, then the most sensitive layers are assigned 4 bits and the least sensitive layers 2 bits so as to match a target average bit-width in one pass.
- **Greedy search.** Starting from all layers at 4 bits, we iteratively quantize one layer to 2 bits, measure the impact on JSD loss, and permanently fix the layer causing the smallest quality drop. This repeats until the target average bit-width is reached.

Table 12 and Table 13 demonstrate that AMQ outperformed both one-shot and greedy search methods, resulting in significantly lower perplexity and higher zero-shot task performance. These results highlight the effectiveness of AMQ's finer-grained approach, which surpasses heuristic methods in optimizing the trade-off between memory and quality.

H Additional Experiments

Table 15, Table 16, Table 17, Table 18, Table 19, and Table 20 are the evaluations of Llama 3.1 8/70B, Qwen2.5 7/14B, Mistral 7B v0.3 with AMQ and baselines, respectively.

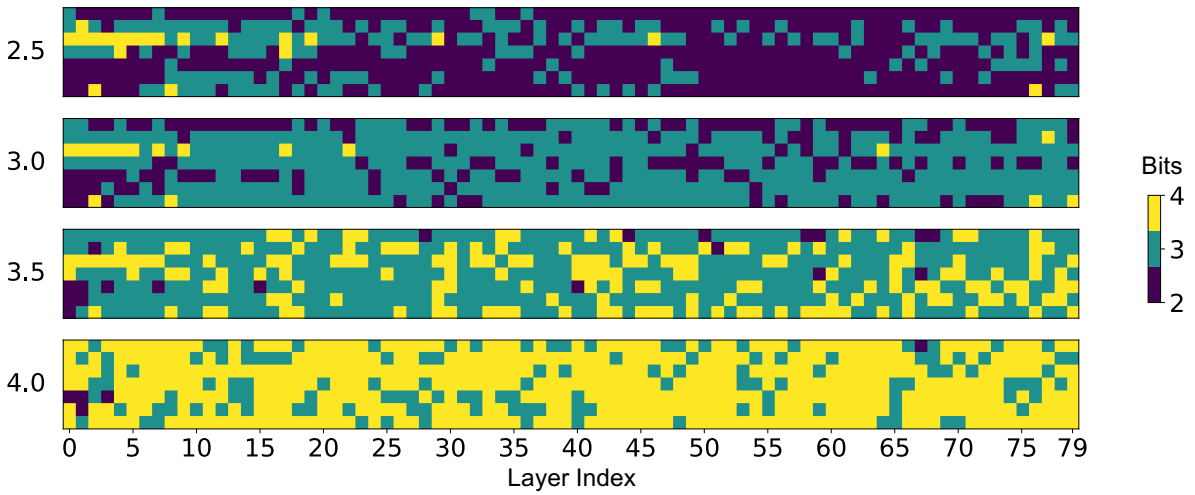

Figure 14: Visualization of bit allocation over linear layers with different average bits at Llama 2 70B. Each row in each box represents Q, K, V, O, Gate, Up, and Down. The numbers on the left indicate the average bits per configuration.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
7B	12,853	16	FP16	5.47	7.26	74.58	46.25	79.11	76.00	69.22	77.77	70.49
	2,431	2.5	One-shot	10.22	13.46	58.75	34.30	72.20	62.31	61.09	65.11	58.96
			Greedy	9.98	13.16	57.15	35.07	72.91	63.94	62.67	**66.94**	59.78
			AMQ	**9.24**	**12.37**	**58.88**	**36.86**	**73.50**	**65.01**	**62.75**	66.39	**60.56**
	2,817	3.0	One-shot	8.04	10.75	59.93	37.71	74.70	66.87	63.69	65.20	61.35
			Greedy	7.72	10.33	64.27	38.14	75.24	68.05	64.96	64.77	62.57
			AMQ	**6.83**	**9.03**	**68.22**	**41.72**	**76.55**	**71.27**	**67.32**	**68.44**	**65.59**
	3,203	3.5	One-shot	6.77	8.96	67.68	41.47	76.66	70.97	66.61	67.55	65.16
			Greedy	6.75	8.92	68.31	41.89	76.99	70.92	67.32	67.16	65.43
			AMQ	**5.95**	**7.90**	**71.55**	**44.20**	**77.86**	**73.92**	**69.06**	**73.88**	**68.41**
13B	24,826	16	FP16	4.88	6.73	77.53	49.15	80.52	79.37	72.30	80.55	73.23
	4,408	2.5	One-shot	7.74	10.54	65.24	39.51	75.03	68.37	66.46	74.16	64.79
			Greedy	7.17	9.71	**67.80**	**40.44**	76.01	69.68	**69.69**	76.02	66.61
			AMQ	**6.88**	**9.46**	**67.38**	40.19	**77.09**	**71.11**	69.38	**77.16**	**67.05**
	5,164	3.0	One-shot	6.56	8.89	70.29	41.98	77.69	71.62	69.30	75.44	67.72
			Greedy	6.32	8.66	70.88	43.69	76.99	72.35	69.46	78.62	68.66
			AMQ	**5.68**	**7.80**	72.18	**45.39**	**78.35**	**76.02**	**70.32**	**79.57**	**70.31**
	5,920	3.5	One-shot	5.80	7.80	71.51	44.62	78.94	74.26	71.51	77.68	69.75
			Greedy	5.74	7.81	72.39	45.05	79.54	75.00	70.96	78.81	70.29
			AMQ	**5.20**	**7.13**	**75.84**	**48.89**	**80.25**	**77.67**	**72.22**	**79.14**	**72.34**

Table 13: Evaluation of one-shot search, greedy search and AMQ on Llama 2 7B/13B.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
7B	12,853	16	FP16	5.47	7.26	74.58	46.25	79.11	76.00	69.22	77.77	70.49
	2,238	2.25	$GPTQ_{w2g128}$	61.77	44.10	35.40	25.17	58.32	40.17	51.46	48.62	43.19
			AWQ_{w2g128}	2.22e5	1.68e5	25.76	26.62	50.38	26.07	50.04	37.83	36.12
	2,315	2.35	AMQ	**11.49**	**15.12**	**54.92**	**33.96**	**71.06**	**60.98**	**60.93**	**65.32**	**57.86**
	2,817	3.0	$GPTQ_{w3}$	9.27	11.81	57.62	34.73	74.43	65.68	62.67	**69.05**	60.70
			AWQ_{w3}	15.45	17.44	53.54	33.53	66.21	56.53	60.69	57.52	54.67
			AMQ	**6.83**	**9.03**	**68.22**	**41.72**	**76.55**	**71.27**	**67.32**	68.44	**65.59**
	3,010	3.25	$GPTQ_{w3g128}$	6.45	8.53	**69.70**	43.09	77.53	71.94	68.11	72.97	67.22
			AWQ_{w3g128}	6.25	8.30	69.53	**44.20**	77.48	**73.33**	68.11	73.12	67.63
			AMQ	**6.20**	**8.25**	69.49	43.34	**78.13**	73.21	**68.90**	**74.56**	**67.94**
	3,589	4.0	$GPTQ_{w4}$	6.09	7.86	71.76	43.69	77.75	74.56	**68.90**	74.65	68.55
			AWQ_{w4}	5.83	7.72	70.75	44.11	78.13	74.93	68.67	**78.01**	69.10
			AMQ	**5.68**	**7.54**	**72.10**	**44.54**	**78.45**	**74.99**	68.03	77.61	**69.29**
13B	24,826	16	FP16	4.88	6.73	77.53	49.15	80.52	79.37	72.30	80.55	73.23
	4,029	2.25	$GPTQ_{w2g128}$	27.78	23.39	40.57	25.68	62.08	42.37	52.17	50.98	45.64
			AWQ_{w2g128}	1.22e5	9.55e4	27.10	27.47	49.95	25.99	50.83	62.17	40.59
	4,181	2.35	AMQ	**7.60**	**10.29**	**65.28**	**38.99**	**75.19**	**67.92**	**65.98**	**74.16**	**64.59**
	5,164	3.0	$GPTQ_{w3}$	6.75	8.96	68.10	41.38	76.99	71.40	69.69	76.76	67.39
			AWQ_{w3}	6.45	9.07	70.58	45.22	77.97	72.62	65.11	72.54	67.34
			AMQ	**5.68**	**7.80**	**72.18**	**45.39**	**78.35**	**76.02**	**70.32**	**79.57**	**70.31**
	5,542	3.25	$GPTQ_{w3g128}$	5.48	7.49	74.54	46.93	**79.22**	76.83	69.38	78.44	70.89
			AWQ_{w3g128}	**5.32**	**7.31**	**75.38**	**49.06**	78.94	**77.41**	**72.06**	**79.82**	**72.11**
			AMQ	5.36	7.33	74.96	47.27	79.00	77.18	71.43	79.27	71.52
	6,676	4.0	$GPTQ_{w4}$	5.19	7.06	75.04	47.10	**80.25**	78.39	71.35	78.99	71.85
			AWQ_{w4}	5.06	6.96	**77.40**	**49.40**	79.65	78.57	71.90	78.59	72.59
			AMQ	**5.03**	**6.91**	77.02	48.63	**80.25**	**78.84**	**72.93**	**80.18**	**72.98**
70B	131,563	16	FP16	3.32	5.71	81.02	57.25	82.70	83.78	77.98	83.79	77.75
	19,363	2.25	$GPTQ_{w2g128}$	8.33	10.71	55.35	35.41	72.42	64.59	67.56	67.06	60.40
			AWQ_{w2g128}	7.25e4	6.56e4	25.84	28.41	49.78	25.72	51.30	62.17	40.54
	20,179	2.35	AMQ	**5.17**	**7.59**	**76.05**	**50.17**	**79.82**	**77.54**	**74.51**	**82.63**	**73.45**
	25,483	3.0	$GPTQ_{w3}$	4.88	7.11	75.76	49.66	80.85	79.47	**75.45**	78.65	73.31
			AWQ_{w3}	4.36	6.63	**80.13**	55.63	80.90	80.51	73.32	80.09	75.10
			AMQ	**4.01**	**6.29**	78.54	**55.80**	**81.77**	**81.31**	**75.45**	**83.46**	**76.05**
	27,523	3.25	$GPTQ_{w3g128}$	3.88	6.11	**79.67**	54.69	**82.48**	82.34	**77.03**	**83.61**	**76.64**
			AWQ_{w3g128}	3.74	6.04	79.63	**56.48**	82.21	**82.71**	75.37	83.09	76.58
			AMQ	**3.73**	**6.03**	78.75	56.23	82.43	82.27	75.61	83.00	76.38
	33,643	4.0	$GPTQ_{w4}$	3.59	5.90	80.22	56.31	82.64	83.10	77.19	82.97	77.07
			AWQ_{w4}	3.48	5.84	**80.72**	**56.74**	**83.03**	83.27	77.27	83.43	77.41
			AMQ	**3.46**	**5.80**	79.92	56.57	82.86	**83.31**	**77.58**	**84.62**	**77.48**

Table 14: Evaluation of fixed-precision quantization methods with AMQ over Llama 2 7B/13B/70B. We omit the memory overhead of additional quantization parameters in GPTQ and AWQ at w3 and w4, since it is negligible.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
8B	15,317	16	FP16	6.24	9.54	81.02	53.24	81.23	78.94	73.16	82.17	74.96
	4,085	2.5	BitStack	23.28	38.23	59.43	32.42	71.55	52.13	62.51	**71.10**	58.19
			AMQ	**17.76**	**26.32**	**59.47**	**36.26**	**72.52**	**60.25**	**64.56**	68.75	**60.30**
	4,501	3.0	BitStack	12.55	20.47	68.64	39.33	75.41	63.35	65.67	74.01	64.40
			AMQ	**9.44**	**14.68**	**71.84**	**44.88**	**77.69**	**70.80**	**71.51**	**79.63**	**69.39**
	4,917	3.5	BitStack	9.47	15.29	74.12	43.69	77.37	68.61	68.59	79.17	68.59
			AMQ	**7.39**	**11.56**	**74.71**	**47.27**	**79.27**	**75.99**	**72.14**	**80.09**	**71.58**
	5,333	4.0	BitStack	8.39	13.47	76.64	47.78	78.94	71.61	69.53	**81.19**	70.95
			AMQ	**6.83**	**10.60**	**79.17**	**52.13**	**80.25**	**77.38**	**74.11**	80.86	**73.98**
70B	134,571	16	FP16	2.81	7.11	86.70	65.02	84.22	85.07	79.40	85.35	80.96
	24,411	2.5	BitStack	**7.55**	12.92	**80.43**	**54.18**	80.09	**77.19**	75.53	79.63	**74.51**
			AMQ	7.62	**12.14**	79.50	53.50	**80.14**	75.39	**75.85**	**81.62**	74.33
	28,491	3.0	BitStack	6.38	11.21	81.44	56.66	81.66	79.40	76.95	81.68	76.30
			AMQ	**5.84**	**9.74**	**82.28**	**59.73**	**82.86**	**80.40**	**77.19**	**84.37**	**77.81**
	32,571	3.5	BitStack	5.44	9.52	83.54	59.47	83.24	81.72	77.82	83.64	78.24
			AMQ	**4.26**	**8.20**	**84.05**	**60.92**	**83.73**	**83.10**	**78.30**	**84.59**	**79.11**
	36,651	4.0	BitStack	4.98	8.92	84.64	61.69	83.19	82.01	**79.79**	83.73	79.17
			AMQ	**3.49**	**7.61**	**85.77**	**62.80**	**84.11**	**84.12**	78.77	**85.26**	**80.14**

Table 15: Evaluation of any-size compression method with AMQ over Llama 3.1 8B/70B.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
8B	15,317	16	FP16	6.24	9.54	81.02	53.24	81.23	78.94	73.16	82.17	74.96
	3,877	2.25	$GPTQ_{w2g128}$	3247.77	734.82	27.31	23.55	52.07	26.85	51.38	43.61	37.46
			AWQ_{w2g128}	1.5.E+06	1.9.E+06	24.83	24.40	50.22	26.46	49.80	37.83	35.59
	3,961	2.35	AMQ	**50.00**	**61.40**	**47.81**	**28.41**	**65.51**	**45.03**	**56.75**	**46.70**	**48.37**
	4,501	3.0	$GPTQ_{w3}$	13.37	18.36	60.98	38.82	73.67	67.51	57.06	51.19	58.21
			AWQ_{w3}	18.13	31.70	67.09	44.28	73.78	68.85	58.80	65.84	63.11
			AMQ	**9.44**	**14.68**	**71.84**	**44.88**	**77.69**	**70.80**	**71.51**	**79.63**	**69.39**
	4,709	3.25	$GPTQ_{w3g128}$	26.95	21.35	56.52	34.90	71.22	67.05	67.72	69.20	61.10
			AWQ_{w3g128}	8.14	12.79	73.91	**47.87**	78.24	73.82	70.17	**79.36**	70.56
			AMQ	**7.96**	**12.45**	**75.34**	47.53	**78.89**	**74.39**	**71.51**	78.23	**70.98**
	5,333	4.0	$GPTQ_{w4}$	87.50	53.10	55.51	37.37	59.36	42.72	67.80	64.04	54.47
			AWQ_{w4}	7.18	11.07	76.94	51.11	**80.63**	**77.52**	73.32	80.73	73.38
			AMQ	**6.83**	**10.60**	**79.17**	**52.13**	80.25	77.38	**74.11**	**80.86**	**73.98**
70B	134,571	16	FP16	2.81	7.11	86.70	65.02	84.22	85.07	79.40	85.35	80.96
	22,371	2.25	$GPTQ_{w2g128}$	113.22	131.90	25.38	25.85	51.69	37.16	52.64	47.40	40.02
			AWQ_{w2g128}	1.8.E+06	1.5.E+06	24.54	26.02	51.52	26.43	53.20	62.17	40.65
	23,187	2.35	AMQ	**8.46**	**13.18**	**73.91**	**48.38**	**78.18**	**72.79**	**73.16**	**79.63**	**71.01**
	28,491	3.0	$GPTQ_{w3}$	1.6.E+04	1.3.E+04	25.80	25.94	52.23	26.45	48.78	37.83	36.17
			AWQ_{w3}	43.14	43.59	42.30	28.92	63.93	44.57	53.04	53.33	47.68
			AMQ	**5.84**	**9.74**	**82.28**	**59.73**	**82.86**	**80.40**	**77.19**	**84.37**	**77.81**
	30,531	3.25	$GPTQ_{w3g128}$	5.17	8.76	68.22	43.86	74.37	81.61	76.09	82.39	71.09
			AWQ_{w3g128}	**4.80**	**8.62**	**83.96**	**62.37**	83.41	**82.67**	**78.85**	83.64	**79.15**
			AMQ	5.09	8.91	82.95	60.67	**83.68**	82.41	78.06	**85.23**	78.83
	36,651	4.0	$GPTQ_{w4}$	1.4.E+04	8.8.E+03	25.29	26.79	52.12	26.43	51.85	37.86	36.73
			AWQ_{w4}	4.18	8.29	83.00	60.32	83.19	83.39	63.06	82.75	75.95
			AMQ	**3.49**	**7.61**	**85.77**	**62.80**	**84.11**	**84.12**	**78.77**	**85.26**	**80.14**

Table 16: Evaluation of fixed-precision quantization methods with AMQ over Llama 3.1 8B/70B. We omit the memory overhead of additional quantization parameters in GPTQ and AWQ at w3 and w4, since it is negligible.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
7B	14,525	16	FP16	6.85	11.89	77.69	51.45	79.92	78.96	73.01	84.56	74.26
	4,025	2.5	BitStack	20.97	38.16	65.66	37.29	71.87	54.82	**62.90**	**75.41**	61.33
			AMQ	**12.85**	**19.81**	**67.42**	**42.24**	**74.43**	**66.42**	61.64	73.52	**64.28**
	4,414	3.0	BitStack	11.92	20.31	**75.67**	**47.78**	76.01	65.48	66.46	77.58	68.16
			AMQ	**8.74**	**14.30**	73.74	46.59	**78.45**	**73.88**	**66.93**	**82.02**	**70.27**
	4,803	3.5	BitStack	9.17	15.86	**79.12**	**52.13**	78.94	70.22	**71.35**	83.98	**72.62**
			AMQ	**7.57**	**12.78**	73.36	48.04	**79.22**	**76.61**	70.09	**84.07**	71.90
	5,192	4.0	BitStack	8.36	14.43	**79.55**	**52.39**	79.60	72.43	**73.16**	**84.80**	73.65
			AMQ	**7.20**	**12.33**	77.40	51.79	**79.98**	**77.74**	71.98	83.82	**73.79**
14B	28,171	16	FP16	5.29	10.35	79.38	58.96	82.37	82.90	75.93	85.32	77.48
	6,909	2.5	BitStack	13.14	22.54	67.68	41.38	75.19	64.40	**70.88**	70.76	65.05
			AMQ	**10.18**	**16.38**	**71.42**	**46.33**	**75.52**	**70.94**	68.03	**71.80**	**67.34**
	7,697	3.0	BitStack	9.12	15.71	67.97	45.22	77.69	72.50	**75.85**	75.44	69.11
			AMQ	**7.23**	**12.31**	**80.56**	**54.44**	**80.41**	**77.83**	72.85	**81.87**	**74.66**
	8,484	3.5	BitStack	7.29	13.22	**80.30**	53.33	79.43	75.67	**77.27**	80.49	74.42
			AMQ	**6.27**	**11.15**	80.01	**55.89**	**80.58**	**80.78**	74.74	**85.29**	**76.21**
	9,272	4.0	BitStack	6.72	12.30	**82.62**	56.66	79.76	77.62	**77.11**	82.29	76.01
			AMQ	**5.81**	**10.73**	80.85	**59.30**	**81.77**	**82.06**	75.69	**84.37**	**77.34**

Table 17: Evaluation of any-size compression method with AMQ over Qwen2.5 7B/14B.

Model	Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
7B	15,317	16	FP16	6.85	11.89	77.69	51.45	79.92	78.96	73.01	84.56	74.26
	3,830	2.25	$GPTQ_{w2g128}$	57.77	55.94	32.91	27.65	57.56	41.11	51.93	47.52	43.11
			AWQ_{w2g128}	1.1.E+07	1.3.E+07	26.26	26.54	51.36	25.93	49.72	37.83	36.27
	3,908	2.35	AMQ	**15.08**	**23.57**	**63.59**	**39.16**	**72.69**	**63.86**	**61.01**	**70.46**	**61.80**
	4,414	3.0	$GPTQ_{w3}$	13.37	18.36	60.98	38.82	73.67	67.51	57.06	51.19	58.21
			AWQ_{w3}	18.13	31.70	67.09	44.28	73.78	68.85	58.80	65.84	63.11
			AMQ	**8.74**	**14.30**	**73.74**	**46.59**	**78.45**	**73.88**	**66.93**	**82.02**	**70.27**
	4,608	3.25	$GPTQ_{w3g128}$	8.19	13.28	64.69	43.77	77.75	75.74	67.96	82.14	68.67
			AWQ_{w3g128}	8.03	13.47	**78.66**	**49.49**	78.35	75.27	**68.43**	**84.98**	**72.53**
			AMQ	**7.91**	**13.21**	73.91	48.04	**79.05**	**75.81**	66.93	84.13	71.31
	5,197	4.0	$GPTQ_{w4}$	7.65	12.74	74.45	50.17	79.87	77.04	68.27	83.06	72.14
			AWQ_{w4}	7.63	13.13	76.77	50.26	79.22	**77.74**	70.88	83.49	73.06
			AMQ	**7.20**	**12.33**	**77.40**	**51.79**	**79.98**	**77.74**	**71.98**	**83.82**	**73.79**
14B	28,172	16	FP16	5.29	10.35	79.38	58.96	82.37	82.90	75.93	85.32	77.48
	6,515	2.25	$GPTQ_{w2g128}$	39.38	46.25	33.71	25.09	58.76	40.32	49.17	56.24	43.88
			AWQ_{w2g128}	3.7.E+07	3.4.E+07	25.08	27.05	52.12	26.20	49.25	62.17	40.31
	6,673	2.35	AMQ	**12.45**	**19.90**	**64.44**	**40.44**	**74.32**	**65.92**	**65.27**	**65.81**	**62.70**
	7,697	3.0	$GPTQ_{w3}$	9.81	14.68	68.56	43.34	77.20	72.05	63.06	57.65	63.64
			AWQ_{w3}	8.53	14.04	70.83	47.70	78.94	77.56	66.22	71.28	68.76
			AMQ	**7.23**	**12.31**	**80.56**	**54.44**	**80.41**	**77.83**	**72.85**	**81.87**	**74.66**
	8,090	3.25	$GPTQ_{w3g128}$	6.96	11.65	**82.37**	**58.02**	**80.85**	**79.77**	72.22	**84.65**	**76.31**
			AWQ_{w3g128}	6.65	11.60	80.77	55.80	80.79	79.48	**75.14**	83.00	75.83
			AMQ	**6.61**	**11.49**	79.71	54.78	80.58	79.76	72.53	84.46	75.30
	9,272	4.0	$GPTQ_{w4}$	6.31	11.10	81.48	57.34	**81.77**	81.37	74.90	84.28	76.86
			AWQ_{w4}	6.06	11.02	**81.57**	58.02	81.45	**82.25**	74.43	**85.87**	77.26
			AMQ	**5.81**	**10.73**	80.85	**59.30**	**81.77**	82.06	**75.69**	84.37	**77.34**

Table 18: Evaluation of fixed-precision quantization method with AMQ over Qwen2.5 7B/14B. We omit the memory overhead of additional quantization parameters in GPTQ and AWQ at w3 and w4, since it is negligible.

Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
13,825	16	FP16	5.32	8.48	78.37	52.30	82.43	80.42	73.88	82.11	74.92
2,593	2.5	BitStack	10.68	16.32	64.44	35.67	75.24	61.71	66.54	74.22	62.97
		AMQ	**8.34**	**12.62**	**66.41**	**38.82**	**77.09**	**70.19**	**66.69**	**79.60**	**66.47**
3,009	3	BitStack	7.94	12.20	70.71	40.87	77.48	69.57	68.75	78.44	67.63
		AMQ	**6.46**	**10.06**	**73.57**	**45.48**	**79.49**	**76.09**	**69.38**	**82.54**	**71.09**
3,425	3.5	BitStack	6.69	10.39	73.15	44.20	79.16	73.17	70.96	78.07	69.78
		AMQ	**5.69**	**9.01**	**75.04**	**49.15**	**81.07**	**79.09**	**71.98**	**82.75**	**73.18**
3,841	4	BitStack	6.24	9.70	75.29	46.25	79.92	75.43	70.48	80.83	71.37
		AMQ	**5.49**	**8.74**	**77.36**	**51.28**	**81.77**	**79.71**	**72.30**	**82.91**	**74.22**

Table 19: Evaluation of any-size compression method with AMQ over Mistral 7B v0.3.

Memory (MB)	Average Bits	Method	Wiki2(↓)	C4(↓)	ARC-e(↑)	ARC-c(↑)	PIQA(↑)	HellaS.(↑)	WinoG.(↑)	BoolQ(↑)	Avg.(↑)
13,825	16	FP16	5.32	8.48	78.37	52.30	82.43	80.42	73.88	82.11	74.92
2,385	2.25	$GPTQ_{w2g128}$	23.71	27.85	40.32	28.84	60.07	42.68	54.62	51.04	46.26
		AWQ_{w2g128}	3.7.E+04	3.7.E+04	26.05	28.67	51.14	25.83	49.88	37.83	36.57
2,469	2.35	AMQ	**10.34**	**15.47**	**64.18**	**35.58**	**76.88**	**65.78**	**63.93**	**75.96**	**63.72**
3,009	3	$GPTQ_{w3}$	9.55	13.57	66.08	42.06	77.42	72.27	63.30	68.72	64.97
		AWQ_{w3}	7.54	12.14	72.56	43.94	78.94	74.49	64.48	70.46	67.48
		AMQ	**6.46**	**10.06**	**73.57**	**45.48**	**79.49**	**76.09**	**69.38**	**82.54**	**71.09**
3,217	3.25	$GPTQ_{w3g128}$	6.20	9.63	73.11	46.76	79.82	77.68	71.35	78.38	71.18
		AWQ_{w3g128}	5.92	9.34	**75.29**	**48.89**	**80.36**	77.43	71.03	79.27	72.05
		AMQ	**5.91**	**9.33**	75.08	**48.89**	79.82	**78.28**	**71.74**	**82.17**	**72.66**
3,841	4	$GPTQ_{w4}$	5.74	9.01	75.97	49.49	80.58	77.22	71.11	80.64	72.50
		AWQ_{w4}	5.72	9.02	77.02	50.60	80.63	79.20	**72.69**	79.17	73.22
		AMQ	**5.49**	**8.74**	**77.36**	**51.28**	**81.77**	**79.71**	72.30	**82.91**	**74.22**

Table 20: Evaluation of fixed-precision quantization methods with AMQ over Mistral 7B v0.3. We omit the memory overhead of additional quantization parameters in GPTQ and AWQ at w3 and w4, since it is negligible.