

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration

Ji Lin^{1*} Jiaming Tang^{1,2*} Haotian Tang¹ Shang Yang^{1,3} Xingyu Dang^{1,3} Song Han¹

¹MIT ²SJTU ³Tsinghua University

<https://github.com/mit-han-lab/llm-awq>

Abstract

Large language models (LLMs) have shown excellent performance on various tasks, but the astronomical model size raises the hardware barrier for serving (memory size) and slows down token generation (memory bandwidth). In this paper, we propose Activation-aware Weight Quantization (AWQ), a hardware-friendly approach for LLM low-bit weight-only quantization. Our method is based on the observation that weights are not equally important: protecting *only 1%* of salient weights can greatly reduce quantization error. We then propose to search for the optimal per-channel scaling that protects the salient weights by *observing the activation, not weights*. AWQ does not rely on any backpropagation or reconstruction, so it can well preserve LLMs’ generalization ability on different domains and modalities, without overfitting to the calibration set; it also does not rely on any data layout reordering, maintaining the hardware efficiency. AWQ outperforms existing work on various language modeling, common sense QA, and domain-specific benchmarks. Thanks to better generalization, it achieves excellent quantization performance for *instruction-tuned* LMs and, for the first time, *multi-modal* LMs. We also implement efficient tensor core kernels with reorder-free online dequantization to accelerate AWQ, achieving a **1.45×** speedup over GPTQ and is **1.85×** faster than the cuBLAS FP16 implementation. Our method provides a turn-key solution to compress LLMs to 3/4 bits for efficient deployment.

1 Introduction

Large language models (LLMs) based on transformers [48] have shown excellent performance on various benchmarks [5, 58, 47, 43]. However, the large model size leads to the high serving costs. For example, GPT-3 has 175B parameters, which is 350GB in FP16, while the latest H100 GPU only has 96GB memory, let alone edge devices.

Low-bit weight quantization for LLMs can save memory but is hard. Quantization-aware training (QAT) is not practical due to the high training cost, while post-training quantization (PTQ) suffers from large accuracy degradation under a low-bit setting. The closest work is GPTQ [17], which uses second-order information to perform error compensation. But it relies on a reordering technique to work for some models (*e.g.*, OPT-66B and LLaMA-7B), which is hardware inefficient (about $2\times$ slower, Figure 6). It may also over-fit the calibration set during reconstruction, distorting the learned features on out-of-distribution domains (Figure 7), which could be a problem for multi-modal models.

In this paper, we propose Activation-aware Weight Quantization (AWQ), a hardware-friendly low-bit weight-only quantization method for LLMs. Our method is based on the observation that *weights are not equally important* for LLMs’ performance. There is a small fraction (0.1%-1%) of *salient* weights; skipping the quantization of these salient weights will significantly reduce the quantization loss (Table 1). To find the salient weight channels, the insight is that we should refer to the *activation*

* indicates equal contributions.

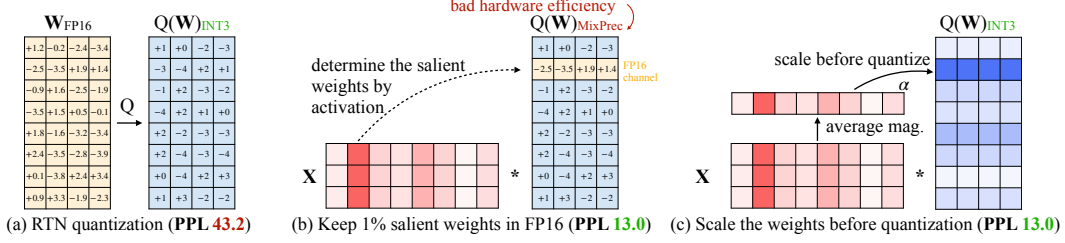


Figure 1. We observe that we can find 1% of the salient weights in LLMs by observing the *activation distribution* (middle). Keeping the salient weights in FP16 can significantly improve the quantized performance (PPL from 43.2 (left) to 13.0 (middle)), but the mixed-precision format is not hardware-efficient. We follow the activation-awareness principle and propose AWQ (right). AWQ performs per-channel scaling to protect the salient weights, leading to reduced quantized error. PPL is measured with OPT-6.7B under INT3-g128 quantization.

distribution instead of the *weight* distribution, despite we are doing *weight-only* quantization: weight channels corresponding to larger activation magnitudes are more salient since they process more important features. To avoid mixed precision, we follow the activation-awareness principle and design a per-channel scaling method to search for the optimal scaling factors that minimize the quantization error, and quantize all the weights.

AWQ does not rely on any backpropagation or reconstruction, so it can well preserve LLMs’ generalization ability on various domains and modalities without overfitting to the calibration set. It does not rely on any data layout reordering and preserves memory regularity, which is $2\times$ faster than reordering based methods. We also implement efficient GPU kernels from scratch to support AWQ. We employ *reorder-free* online dequantization to efficiently convert low-bit weights to FP16 and effectively map to high-throughput tensor cores rather than cuda cores.

Experiments show that AWQ outperforms existing work on various tasks (language modeling, common sense QA, and domain-specific benchmarks) for different model families (*e.g.*, LLaMA [47], OPT [58]) and model sizes. Thanks to better generalization, it also achieves good quantization performance for *instruction-tuned* LMs (*e.g.*, Vicuna) and, for the first time, *multi-modal* LMs (OpenFlamingo [2]). Thanks to our efficient kernels, AWQ achieves $1.45\times$ and $2\times$ speedup over GPTQ and GPTQ with reordering on A100. It is also $1.85\times$ faster than an FP16 cuBLAS implementation. AWQ makes multi-modal LM more efficient without hurting the generalization capability.

2 Activation-aware Weight Quantization (AWQ)

Preliminaries. *Quantization* maps a floating-point number into lower-bit integers. It is an effective method to reduce the model size and inference costs of LLMs [12, 17, 55, 54]. There are typically two quantization settings for LLM quantization: 1. W8A8 quantization [12, 54], which maps both activation and weights into INT8; 2. Low-bit *weight-only* quantization [17, 39], which quantizes weights into low-bit integers (typically ≤ 4 bits). The decoding stage dominates LLMs’ total runtime [17, 32], which is highly memory-bounded with a single batch size. Given that the memory is dominated by weights, we focus on weight-only quantization. We study group-wise quantization (with group size 128 [17]) since it helps to improve the accuracy-cost trade-off [13] at little overhead.

In this section, we first propose a method to improve quantized performance *without training/regression* by protecting more “important” weights. And then develop a data-driven method to search for the optimal scaling that reduces quantization errors (Figure 1).

2.1 Improving LLM Quantization by Preserving 1% Salient Weights

We observe that the weights of LLMs are *not equally important*: there is a small fraction of *salient* weights that are much more important for LLMs’ performance compared to others. Skipping the quantization of these salient weights can help bridge the performance degradation due to the quantization loss *without* any training or regression (Figure 1(b)). To verify the idea, we benchmark the performance of quantized LLMs when skipping part of the weight channels in Table 1. We measured the performance of INT3 quantized models while keeping some ratios of weight channels in FP16. A widely used method to determine the importance of weights is to look at its magnitude or L_2 -norm [22, 16]. But we find skipping the weight channels with large norm (*i.e.*, FP16% (based on W)) does not significantly improve the quantized performance, leading to a similar marginal

PPL ↓	FP16	RTN (w3-g128)	FP16% (based on act.)			FP16% (based on W)			FP16% (random)		
			0.1%	1%	3%	0.1%	1%	3%	0.1%	1%	3%
OPT-1.3B	16.41	206.5	28.00	18.51	18.30	187.1	173.1	165.5	211.2	201.4	88.44
OPT-6.7B	12.29	43.16	13.14	13.02	12.97	43.51	38.59	39.78	42.73	37.83	46.49
OPT-13B	11.50	45.37	12.14	12.04	12.00	46.21	48.07	54.38	45.95	44.47	40.01

Table 1. Keeping a small fraction of weights (0.1%-1%) in FP16 significantly improves the performance of the quantized models over round-to-nearest (RTN). It is only effective when we select the important weights in FP16 by looking at *activation* distribution instead of *weight* distribution. We highlight results with a decent perplexity in **green**. We used INT3 quantization with a group size of 128 and measured the WikiText perplexity (↓).

PPL ↓	FP16	RTN (w3-g128)	Heuristic scaling				Search to scale			
			×1.25	×1.5	×2	×4	s _W -only	s _X -only	s _X & s _W	+clip
OPT-1.3B	16.41	206.5	30.89	21.75	21.05	21.95	19.45	19.07	19.03	18.53
OPT-6.7B	12.29	43.16	15.45	14.49	14.07	14.42	14.46	13.18	13.13	12.99
LLaMA-7B	9.49	12.10	11.71	11.57	11.48	13.26	11.42	11.24	11.27	10.82

Table 2. Simply scaling up the salient weight channels by a factor greater than 1 can largely reduce the quantization error (heuristic scaling). We further propose to search for the optimal scale. Considering the activation distribution (*i.e.*, s_X) is the most important factor during scaling factor search.

improvement as random selection. Interestingly, selecting weights based on *activation magnitude* can significantly improve the performance: keeping only 0.1%-1% of the channels corresponding to larger activation significantly improves the quantized performance, even matching a strong reconstruction-based method GPTQ [17]. We hypothesize that the input features with larger magnitudes are generally more important. Keeping the corresponding weights in FP16 can preserve those features, which contributes to better model performance.

Limitations: Despite keeping 0.1% of weights in FP16 can improve the quantized performance without a noticeable increase in model size (measured in total bits), such a mixed-precision data type will make the system implementation difficult. We need to come up with a method to protect the important weights without actually keeping them as FP16.

2.2 Protecting Salient Weights by Activation-aware Scaling

We propose a method to reduce the quantization error of the salient weight by *per-channel scaling*, which does not suffer from the hardware inefficiency issue.

Heuristic-based scaling. We first implement a heuristic-based scaling method to reduce the quantization error. Since we are using a min-max based quantization following prior work [12, 54, 17], the quantization scales are determined by the extreme values in each group (*i.e.*, $s = (\mathbf{w}_{\max} - \mathbf{w}_{\min}) / (2^N - 1)$, where N is the quantization bit width), thus there is *no quantization loss* for the maximum and minimum values*. A straightforward way to protect the outlier weight channels is to multiply the channel by a certain scaling ratio (>1) such that they can be precisely quantized. As shown in Table 2 (Heuristic scaling column), such methods do improve the performance of the quantized model, but there is still a gap compared to directly keeping 1% weights in FP16. The quantized performance generally gets better as the scaling factor increases since important weights are better represented. It then decreases since non-salient channels are forced to use a smaller dynamic range (or smaller effective bits) if we use a very large scaling factor. We need to reduce the quantization error for the important weights while not increasing the error for other weights. We need an automated way to find a scaling ratio for each input channel that achieves the goal.

Searching to scale. We automatically search for an optimal (per input channel) scaling factor that minimizes the output difference after quantization for a certain layer. Formally, we want to optimize the following objective:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} \mathcal{L}(\mathbf{s}), \quad \mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\| \quad (1)$$

*There is a small chance that multiple outliers appear in the same quantization group. But it would be rare when using a small outlier ratio like 0.1%, so we will ignore it for the study.

Here Q means the weight quantization function (e.g., INT3/INT4 quantization with group size 128), \mathbf{W} is the original weights in FP16, and \mathbf{X} is the input features cached from a small calibration set (we take a small calibration set from the pre-training dataset in order not to overfit to a specific task). s is a per-(input) channel scaling factor; for $s^{-1} \cdot \mathbf{X}$, it can usually be fused into the previous operator [52, 54]. Since the quantization function is not differentiable, we are not able to directly optimize the problem with vanilla backpropagation. There are some techniques relying on approximated gradients [3, 15], which we found still converges poorly and has the potential risk of over-fitting.

We design the *search space* for the optimal scale by analyzing the factors that will affect the choice of scaling factor. Intuitively, the optimal scale should be related to: **1. Activation magnitude:** as mentioned above, we are relying on input activation magnitude to pick out the salient weight channels. Therefore, we should consider the magnitude of \mathbf{X} to protect the *salient* weight channels. We compute the average activation magnitude $s_{\mathbf{X}} = \text{mean}_{c_{\text{out}}} |\mathbf{X}|$ as the importance factor. **2. Weight magnitude:** to minimize the quantization loss of *non-salient* weights, we should flatten their distribution so that it is easier for quantization [54]. A reasonable choice is to *divide* the weight channels by their average magnitude $s_{\mathbf{W}} = \text{mean}_{c_{\text{out}}} |\hat{\mathbf{W}}|$ [54], where $\hat{\mathbf{W}}$ refers to \mathbf{W} after normalization within each group. We determine the final scaling factors by a function considering both of the factors $s = f(s_{\mathbf{X}}, s_{\mathbf{W}})$. For simplicity, we multiply both scales and solve:

$$s = f(s_{\mathbf{X}}, s_{\mathbf{W}}) = s_{\mathbf{X}}^{\alpha} \cdot s_{\mathbf{W}}^{-\beta}, \quad \alpha^*, \beta^* = \arg \min_{\alpha, \beta} \mathcal{L}(s_{\mathbf{X}}^{\alpha} \cdot s_{\mathbf{W}}^{-\beta}) \quad (2)$$

α and β are hyper-parameters to control the strength of each component. We can pick the optimal α and β with a simple grid search over the interval $[0, 1]$. As shown in Table 2, scaling based on $s_{\mathbf{X}}$ significantly outperforms $s_{\mathbf{W}}$, demonstrating the importance of activation-awareness. Further introducing $s_{\mathbf{W}}$ only brings a marginal improvement, which again shows activation-awareness is most crucial. Furthermore, we find adjusting the clipping range by searching a shrinking ratio (denoted as “+clip”) can sometimes further improve the quantized performance. Adjusting the clipping range leads to a nudged scaling factor [15] which may help better protect the salient weights. We combine both scaling and clipping to form AWQ.

Explaining GPTQ reordering. Our principle of activation-awareness can also explain why we need reordering to make GPTQ work on some models (e.g., LLaMA-7B and OPT-66B, please refer to the `--act-order` option in GPTQ’s repo[†]). Different weight channels have different importance; updating the salient channels to compensate for the non-salient ones will likely destroy the performance. Reordering prevents it by quantizing important channels first. However, it will lead to bad hardware efficiency due to irregular memory access (Figure 2), while our scaling method does not suffer from the issue.

Advantages. Furthermore, our method does not rely on any regression [17] or backpropagation, which is required by many quantization-aware training methods. It has minimal reliance on the calibration set since we only measure the average magnitude per channel, thus preventing over-fitting (Figure 7). Therefore, our method requires fewer data for the quantization process and can preserve LLMs’ knowledge outside of the calibration set’s distribution. See Section 3.4 for more details.

2.3 Co-design Efficient Quantized GPU Kernels

The hardware-friendly nature of AWQ facilitates the development of highly efficient GPU kernels, effectively translating theoretical memory savings into measured speedups.

Unlike GPTQ [17] which formulates linear layers as *matrix-vector* (MV) products, we instead model these layers as *matrix-matrix* (MM) multiplications. MV can only be executed on slow *CUDA cores* while MM can be executed on the $16\times$ faster *tensor cores* on A100 and H100. Our formulation also minimizes structural hazards compared to [17] since MM and other instructions such as dequantization and memory access are executed on separate function units. We also outperform a recent Triton [46] implementation [40] for GPTQ by $2.4\times$ since it relies on a high-level language and forgoes opportunities for low-level optimizations.

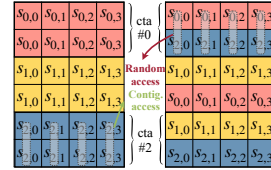


Figure 2. Reordering leads to inefficient random memory access.

[†]<https://github.com/IST-DASLab/gptq>

LLaMA-7B		MMLU (5-shot) \uparrow					Common Sense QA (0-shot) \uparrow				
		Hums.	STEM	Social	Other	Avg.	PIQA	Hella.	Wino.	ARC-e	Avg.
FP16	-	39.17%	32.32%	42.72%	42.56%	38.41%	78.35%	56.44%	67.09%	67.30%	67.30%
	RTN	31.37%	31.10%	36.04%	36.49%	33.43%	75.84%	53.10%	63.22%	66.04%	64.55%
INT3	GPTQ	29.29%	29.04%	33.03%	31.65%	30.53%	70.89%	46.77%	60.93%	60.06%	59.66%
g128	GPTQ-R	33.98%	30.71%	37.78%	36.49%	34.26%	77.31%	53.81%	67.56%	63.72%	65.60%
	AWQ	35.15%	31.61%	39.27%	37.75%	35.43%	76.66%	53.63%	66.14%	65.70%	65.53%
	RTN	36.15%	33.03%	41.41%	41.21%	37.37%	77.86%	55.81%	65.59%	66.25%	66.38%
INT4	GPTQ	35.55%	30.95%	39.29%	38.12%	35.39%	77.20%	53.98%	65.67%	61.62%	64.62%
g128	GPTQ-R	37.28%	31.36%	40.23%	40.77%	36.72%	78.45%	56.00%	66.85%	66.88%	67.05%
	AWQ	38.32%	32.00%	41.38%	42.07%	37.71%	78.07%	55.76%	65.82%	66.84%	66.62%

Table 3. AWQ consistently outperforms RTN and GPTQ for LLaMA-7B under 3/4-bit quantization with a group size of 128. Note that GPTQ needs the hardware-inefficient reordering trick to work on the 7B model, denoted as GPTQ-R, which is significantly slower (Figure 6), so we exclude it for performance comparison.

Weight-only quantization methods require online weight dequantization. In Figure 2, we assume that the weights are stored in row major (*i.e.* IC \times OC). Each OC has different scales and zero points and every two ICs within each OC share dequantization parameters (*i.e.* $g = 2$). For GPTQ with reordering (right), since these $g = 2$ ICs are not continuous, irregular DRAM accesses are required to fetch scaling factors and zero points when dequantizing each weight. However, for AWQ (left), all memory accesses are contiguous, resulting in a $2\times$ end-to-end speedup for LLaMA models.

3 Experiments

3.1 Settings

Quantization. We focus on *weight-only grouped* quantization in this work. As shown in previous work [13, 17], grouped quantization is always helpful for improving performance/model size trade-off. We used a group size of 128 throughout the work, except otherwise specified. We focus on INT4/INT3 quantization since they are able to mostly preserve the LLMs’ performance [13]. For AWQ, we used a small calibration set from the Pile [18] dataset in order not to overfit to a specific downstream domain. We used a grid size of 20 to search for the optimal α and β in Equation 2.

Models. We benchmarked our method on LLaMA [47] and OPT [58] families. There are other open LLMs like BLOOM [43], but they are generally worse in quality, so we do not include them in our study. We further benchmark an instruction-tuned model Vicuna [8] and visual language models OpenFlamingo-9B [2] and LLaVA-13B [31] to demonstrate the generability of our method.

Evaluations. Following previous literature [12, 54, 17, 13, 55], we profiled the quantized models on language modeling tasks (WikiText-2 [33]) and common sense QA benchmarks (PIQA [4], HellaSwag [56], WinoGrande [41], ARC [11]; we do not use LAMBADA [38] due to a known bug for LLaMA evaluation). We notice that these benchmarks do not reflect LLMs’ domain-specific knowledge. Therefore, we further benchmark the models on MMLU [23], which consists of 57 tasks covering STEM, humanities, social science, *etc.* It also helps to evaluate the *in-context learning* ability under a few-shot setting. We used `lm-eval-harness` [19] to carry out all the evaluations.

Baselines. Our primary baseline is vanilla round-to-nearest quantization (RTN). It is actually quite strong when using a small group size like 128 [17, 13]. We also compare with a state-of-the-art method GPTQ [17] for LLM weight quantization. For GPTQ, we also compare with an updated version that uses a “reorder” trick (denoted as GPTQ-Reorder or GPTQ-R), which improves the performance but leads to worse decoding efficiency. Other techniques like ZeroQuant [55], AdaRound [34], and BRECQ [28] rely on backpropagation to update the quantized weights, which may not easily scale up to large model sizes; they also do not outperform GPTQ [17], thus not included for study.

3.2 Accuracy Evaluation

Results on LLaMA models. We focus our study on LLaMA models due to their superior performance compared to other open-source LLMs [58, 43]; it is also the foundation of many popular open-source models [45, 8]. We evaluate different quantization methods on common sense QA tasks

LLaMA Family		MMLU (5-shot) average \uparrow				Common Sense QA (0-shot) average \uparrow			
		7B	13B	30B	65B	7B	13B	30B	65B
FP16	-	38.41%	45.21%	56.84%	60.50%	67.30%	70.65%	72.97%	74.49%
INT3 g128	RTN	33.43%	39.20%	50.58%	57.77%	64.55%	68.63%	72.07%	72.58%
	GPTQ	30.53%	40.90%	52.32%	58.04%	59.66%	68.71%	70.77%	73.03%
	AWQ	35.43%	41.84%	53.22%	58.83%	65.53%	69.22%	72.10%	73.39%

Table 4. LLaMA quantization results across various scales. AWQ outperforms round-to-nearest (RTN) and GPTQ [17] across different model scales (7B-65B), task types (common sense vs. domain-specific), and test settings (zero-shot vs. in-context learning).

OPT / PPL \downarrow		125M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	-	31.95	16.41	14.32	12.29	11.5	10.67	10.09
INT3 g128	RTN	58.49	206.54	595.28	43.16	45.37	28.84	423.39
	GPTQ	41.93	18.53	15.79	13.13	12.01	11.00	11.48
	AWQ	41.10	18.53	15.62	12.99	12.03	11.03	10.46
INT4 g128	RTN	35.51	17.70	15.12	13.02	11.89	11.00	10.44
	GPTQ	34.23	16.92	14.69	12.51	11.60	10.74	10.24
	AWQ	33.96	16.85	14.61	12.44	11.60	10.75	10.16

Table 5. AWQ improves over round-to-nearest quantization (RTN) for all model sizes and different bit-precisions. It achieves better WikiText-2 perplexity compared to GPTQ on smaller OPT models and on-par results on larger ones, demonstrating the generality to different model sizes and families. We found OPT performs poorly on MMLU and has limited in-context learning capacity (even FP16 model), so we exclude those evaluations.

(0-shot) and MMLU benchmark with in-context learning (5-shot). We found in preliminary experiments that quantization may hurt a model’s in-context learning performance, which is a fundamental ability of LLMs [6]; thus, MMLU (5-shot) serves as an essential benchmark in our study. We provide the INT3-g128/INT4-g128 results of LLaMA-7B in Table 3; for larger models (Table 4), we find the 4-bit RTN accuracy is already quite good, so we only include the 3-bit results. We can see that AWQ outperforms round-to-nearest (RTN) and GPTQ [17] across different model scales (7B-65B), task types (common sense vs. domain-specific), and test settings (zero-shot vs. in-context learning). Note that the RTN baseline with a group size of 128 is already quite strong (only a few percent accuracy drop), so a 1% error reduction is considered significant. For the LLaMA-7B model, we need to include the reordering trick to make GPTQ work, which leads to hardware-inefficient memory access and a $2\times$ slow down in kernel implementation (Figure 6). Therefore, we only provide the results for reference but exclude them for comparison (in gray).

Results on OPT models. We also provide the results on OPT models [58]. We found that OPT models generally achieve very low MMLU accuracy (may not be meaningful) and have limited in-context learning capacity (please see Table 9 in supplementary). Therefore, we focus on WikiText-2 perplexity evaluation following [17]. As shown in Table 5, AWQ improves both INT3 and INT4 grouped quantization. For smaller models ($<13B$), our method can outperform the competitive GPTQ results despite its simplicity; while for larger models, we are able to obtain on-par performance. This demonstrates the generality to different model families and model sizes.

Quantization of instruction-tuned models. Instruction tuning can significantly improve the models’ performance and usability [50, 42, 37, 10]. It has become an essential procedure before model deployment. We further benchmark our method’s performance on a popular instruction-tuned model Vicuna [8] in Figure 3. We used the GPT-4 score to evaluate the quantized models’ performance against the FP16 counterpart on 80 sample questions [8]. We compare the responses with both orders (quantized-FP16, FP16-quantized) to get rid of the ordering effect (we found GPT-4 tends to increase the rating of the first input), leading to 160 trials. AWQ consistently improves the INT3-g128 quantized Vicuna models over RTN and GPTQ under both scales (7B and 13B), demonstrating the generability to instruction-tuned models.

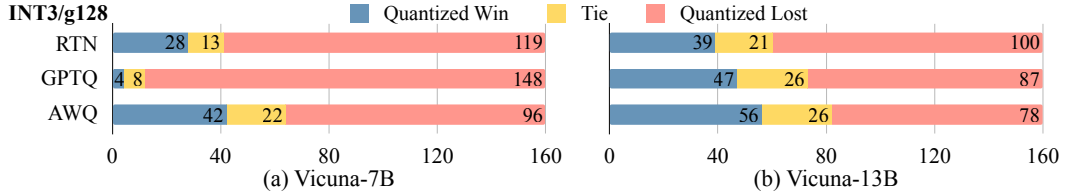


Figure 3. Comparing INT3-g128 quantized Vicuna models with FP16 counterparts under GPT-4 evaluation protocol [8]. More winning cases (in blue) indicate better performance. AWQ consistently improves the quantized performance compared to RTN and GPTQ [17], showing generalization to instruction-tuned models.

COCO (CIDEr \uparrow)		0-shot	4-shot	8-shot	16-shot	32-shot	$\Delta(32\text{-shot})$
FP16	-	63.73	72.18	76.95	79.74	81.70	-
INT4 g128	RTN	60.24	68.07	72.46	74.09	77.13	-4.57
	GPTQ	59.72	67.68	72.53	74.98	74.98	-6.72
	AWQ	62.57	71.02	74.75	78.23	80.53	-1.17
INT3 g128	RTN	46.07	55.13	60.46	63.21	64.79	-16.91
	GPTQ	29.84	50.77	56.55	60.54	64.77	-16.93
	AWQ	56.33	64.73	68.79	72.86	74.47	-7.23

Table 6. Quantization results of a visual language model OpenFlamingo-9B [2] on COCO Captioning datasets. AWQ outperforms existing methods under zero-shot and various few-shot settings, demonstrating the generability to different modalities and in-context learning workloads. AWQ reduces the quantization degradation (32-shot) from 4.57 to 1.17 under INT4-g128, providing 4 \times model size reduction with negligible performance loss.

Quantization of multi-modal language models. Large multi-modal models (LMMs) or visual language models (VLMs) are LLMs augmented with vision inputs [1, 27, 26, 14, 57, 31]. Such models are able to perform text generation conditioned on image/video inputs. Since our method does not have the overfitting issue to the calibration set, it can be directly applied to VLMs to provide accurate and efficient quantization. We perform experiments with the OpenFlamingo-9B model [2] (an open-source reproduction of [1]) on COCO captioning [7] dataset (Table 6). We measured the average performance of 5k samples under different few-shot settings. We only quantize the language part of the model since it dominates the model size. AWQ outperforms existing methods under zero-shot and various few-shot settings, demonstrating the generability to different modalities and in-context learning workloads. It reduces the quantization degradation (32-shot) from 4.57 to 1.17 under INT4-g128, providing 4 \times model size reduction with negligible performance loss. We further provide some qualitative captioning results in Figure 4 to show our advantage over RTN. Our method provides a push-the-button solution for LMM/VLM quantization. It is the *first* study of VLM low-bit quantization to the best of our knowledge.

Visual reasoning results. We further provide some qualitative visual reasoning examples of the LLaVA-13B [31] model in Figure 5. AWQ improves the responses compared to the round-to-nearest (RTN) baseline for INT4-g128 quantization, leading to more reasonable answers. In this first example, the AWQ model can understand the meme as it resembles the Earth when looking from space, while RTN produces wrong descriptions (marked in red). In the second example, AWQ correctly answers the question (the artist of the painting), while RTN does not provide any information about the artist. In the last example, RTN falsely points out a bird in the picture, while AWQ provides more information by noticing the image is taken in a mountain area. AWQ improves the visual reasoning ability of VLMs by reducing factual errors in the responses; RTN is not good enough even for 4 bits.

Extreme low-bit quantization. We further quantize LLM to INT2 to accommodate limited device memory (Table 7). RTN completely fails, and AWQ brings significant perplexity improvement on top of GPTQ, though there is still a performance gap compared to FP16. Our method is orthogonal to GPTQ. We can combine our method with GPTQ to further improve the INT2 quantization performance, making it a more practical setting.

3.3 Efficiency Evaluation

Comparison with GPTQ kernels. In Figure 6(a), we compare the end-to-end efficiency of LLaMA models quantized using AWQ and GPTQ. Since the official GPTQ implementation [17] does not

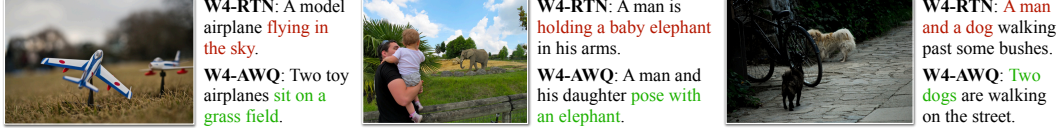


Figure 4. Qualitative results of quantized OpenFlamingo-9B [2] on COCO captioning dataset (4-shot, INT4-g128 quantization). Our method significantly improves the captioning quality compared to the round-to-nearest (RTN) baseline. We color the text to show the **correct** or **wrong** captions.



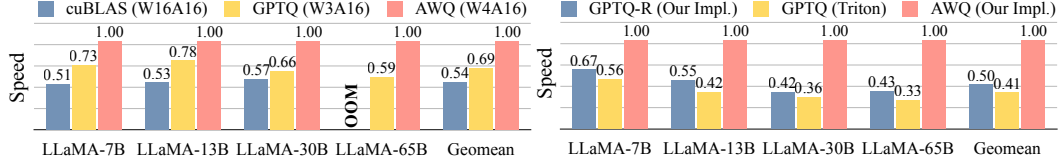
Figure 5. Visual reasoning examples from LLaVA-13B model [31]. AWQ improves over the round-to-nearest (RTN) baseline, providing more reasonable answers. We color the text to show the **correct** or **wrong** responses.

support 4-bit weights and reordering, we present the latency of 3-bit GPTQ without reordering. Remarkably, despite utilizing an additional bit per weight, AWQ achieves an average speedup of $1.45\times$, and a maximum speedup of $1.7\times$ over GPTQ. We attribute this efficiency improvement to our formulation of linear layers in LLMs as *matrix-matrix* instead of *matrix-vector* product. As such, we can take advantage of high-throughput tensor cores on the A100 GPU and avoid resource contention between computation and data preparation (*e.g.* dequantization and memory access) instructions.

Overhead of reordering and compilers. Figure 6(b) demonstrates the significance of eliminating reordering during dequantization. To support GPTQ-R, we made only one modification to the AWQ kernels: introducing indirect addressing when loading zero points and scales in weight dequantization. It turns out that AWQ with *reorder-free* kernels are $1.4\text{--}2.4\times$ faster than GPTQ-R. This clearly indicates that the irregular memory access patterns in GPTQ-R kernels are not hardware-friendly and have a significant negative impact on performance. Furthermore, we note that a direct CUDA-level implementation is essential to achieve consistent performance gains over cuBLAS and GPTQ. We benchmark AWQ kernels against an open-source implementation [40] based on the Triton [46] compiler and achieved $1.8\text{--}3.1\times$ speedup. The Triton kernels become $1.6\times$ slower than the FP16 cuBLAS kernels in LLaMA-30B, while AWQ kernels are still $1.8\times$ faster. This further emphasizes the limitations of compilers in identifying and exploiting low-level optimization opportunities.

OPT / Wiki PPL↓		1.3B	2.7B	6.7B	13B	30B
FP16	-	16.41	14.32	12.29	11.5	10.67
INT2 g64	RTN	26615	740860	22290	28923	15198
	GPTQ	50.05	30.41	19.04	16.8	12.91
	AWQ +GPTQ	35.26	24.02	17.34	14.58	12.5

Table 7. Our method is orthogonal to GPTQ: it further closes the performance gap under extreme low-bit quantization (INT2-g64) when combined with GPTQ. Results are WikiText-2 perplexity of OPT models.



(a) 4-bit AWQ is **1.45×** faster than 3-bit GPTQ (w/o reordering) (b) Reorder elimination brings brings **2×** speedup to W4A16

Figure 6. Left: LLaMA models quantized with 4-bit AWQ is **1.45×** faster than 3-bit GPTQ. **Right:** The reordering trick in GPTQ-R significantly hampers its hardware efficiency. AWQ is reorder-free and **2.0×** faster than GPTQ-R. AWQ is GPU optimization friendly which is **2.4×** faster than the Triton implementation. All numbers are measured on a single 80G A100 GPU.

3.4 Analysis

Better data-efficiency for the calibration set. Our method requires a smaller calibration set since we do not rely on regression/backpropagation; we only measure the average activation scale from the calibration set, which is data-efficient. To demonstrate the idea, we compare the perplexity of the OPT-6.7B model with INT3-g128 quantization in Figure 7 (a). AWQ needs a much smaller calibration to reach a good quantized performance; it can achieve better perplexity using 10× smaller calibration set compared to GPTQ (16 sequences vs. 192 sequences).

Robust to the calibration set distributions. Our method is less sensitive to the calibration set distribution since we only measure the average activation scale from the calibration set, which is more generalizable across different dataset distributions. We further benchmarked the effect of the different calibration set distributions in Figure 7(b). We took two subsets from the Pile dataset [18]: PubMed Abstracts and Enron Emails [25]. We use each of the subsets as the calibration set and evaluate the quantized model on both sets (the calibration and evaluation sets are split with no overlapping; we used 1k samples for evaluation). Overall, using the same calibration and evaluation distribution works the best (PubMed-PubMed, Enron-Enron). But when using a different calibration distribution (PubMed-Enron, Enron-PubMed), AWQ only increases the perplexity by 0.5-0.6, while GPTQ has 2.3-4.9 worse perplexity. This demonstrates the robustness of AWQ to the calibration set distribution.

Differentiation with SmoothQuant. At first glance, the derived formula (Equation 2) is related to SmoothQuant [54], where it balances activation and weight smoothness (actually it is different: we only need s_x , as the weight distribution has a minimal contribution to the performance, see Table 2). However, our method is *fundamentally different* from SmoothQuant. Firstly, the intuition and mechanism are different. SmoothQuant balances the smoothness between weight and activation since *both* of them are quantized using the same INT8 quantizer. It should only consider weight distributions in our setting (only weights are quantized), while we found that activation distribution is essential for the weight quantization but not the weight distribution (Table 2). Secondly, we are focusing on low-bit weight-only quantization, where the SmoothQuant formula cannot achieve decent performance. Take LLaMA-7B INT3-g128 quantization as an example: the default SmoothQuant ($\alpha=0.5$) achieves 11.55 WikiText-2 perplexity. Using $\alpha = 0$ (migrate quantization difficulty from weight to activation) actually makes the perplexity worse at 12.00, which contradicts the story of SmoothQuant and is the same as RTN baseline at 12.10, while our method can achieve 11.07.

4 Related Work

Model quantization methods. Quantization reduces the bit-precision of deep learning models [21, 24, 35, 49, 34, 30], which helps to reduce the model size and accelerate inference. Quantization

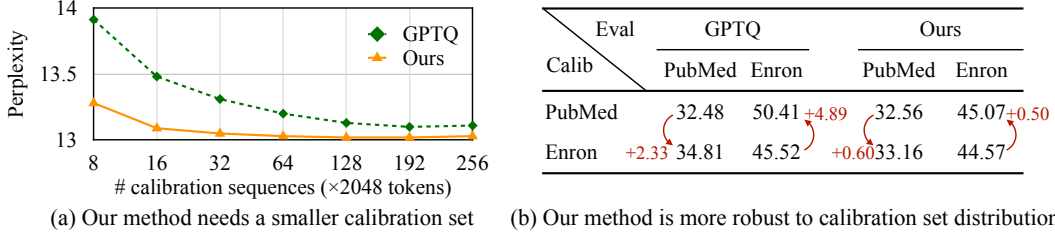


Figure 7. Left: AWQ needs a much smaller calibration set to reach a good quantized performance. It can achieve better perplexity using $10\times$ smaller calibration set compared to GPTQ. **Right:** Our method is more robust to the calibration set distribution. Overall, using the same calibration and evaluation distribution works the best (PubMed-PubMed, Enron-Enron). But when using a different calibration distribution (PubMed-Enron, Enron-PubMed), AWQ only increases the perplexity by 0.5-0.6, while GPTQ has 2.3-4.9 worse perplexity. All experiments are done with the OPT-6.7B model under INT3-g128 quantization.

techniques generally fall into two categories: quantization-aware training (QAT, which relies on backpropagation to update the quantized weights) [3, 20, 36, 9] and post-training quantization [24, 35, 34] (PTQ, usually training-free). The QAT methods cannot easily scale up to large models like LLMs. Therefore, people usually use PTQ methods to quantize LLMs.

Quantization of LLMs. People study two settings for LLM quantization: (1) W8A8 quantization, where both activation and weights are quantized to INT8 [12, 54, 55, 53, 51]; (2) Low-bit weight-only quantization (*e.g.*, W4A16), where only weights are quantized into low-bit integers [17, 13, 44, 39]. We focus on the second setting in this work since it not only reduces the hardware barrier (requiring a smaller memory size) but also speeds up the token generation (remedies memory-bound workload). Apart from the vanilla round-to-nearest baseline (RTN), GPTQ [17] is the closest to our work. However, the reconstruction process of GPTQ leads to an over-fitting issue to the calibration set and may not preserve the generalist abilities of LLMs for other modalities and domains. It also requires a reordering trick to work for some models (*e.g.*, LLaMA-7B [47] and OPT-66B [58]), leading to hardware-inefficient irregular memory access and slowing down the inference.

System support for low-bit quantized LLMs. Low-bit quantized LLMs have been a popular setting to reduce inference costs. There are some system supports to achieve a practical speed-up. GPTQ [17] provides INT3 kernels for OPT models and GPTQ-for-LLaMA extends kernel support for INT4 reordered quantization with the help of the Triton compiler [46]. FlexGen [44] and llama.cpp[‡] perform group-wise INT4 quantization to reduce I/O costs and offloading. FasterTransformer[§] implements FP16 \times INT4 GEMM for weight-only per-tensor quantization but does not support group quantization. LUT-GEMM [39] performs bitwise computation on GPU CUDA cores with the help of lookup tables. Our AWQ kernels are executed on tensor cores, suitable for both context and generation phases in LLM inference, and do not require hardware-inefficient reordering. Therefore, our kernels are $1.45\times$ faster than the best competitor when running LLaMA models.

5 Conclusion

In this work, we propose Activation-aware Weight Quantization (AWQ), a simple yet effective method for low-bit weight-only LLM compression. AWQ is based on the observation that weights are not equally important in LLMs and performs per-channel scaling to reduce the quantization loss of salient weights. AWQ does not over-fit the calibration set and preserves the generalist abilities of LLMs in various domains and modalities. It outperforms existing work on language modeling and QA benchmarks and can be applicable to instruction-tuned LMs and multi-modal LMs. AWQ is also hardware-efficient without irregular memory access. We further implement efficient kernels, achieving $1.45\times$ speedup over GPTQ and $1.85\times$ speed up over cuBLAS FP16 implementation.

Acknowledgements

We thank MIT AI Hardware Program, National Science Foundation, NVIDIA Academic Partnership Award, MIT-IBM Watson AI Lab, Amazon and MIT Science Hub, Qualcomm Innovation Fellowship, Microsoft Turing Academic Program for supporting this research.

[‡]<https://github.com/ggerganov/llama.cpp>

[§]<https://github.com/NVIDIA/FasterTransformer>

References

- [1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.
- [2] Anas Awadalla, Irena Gao, Joshua Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Jenia Jitsev, Simon Kornblith, Pang Wei Koh, Gabriel Ilharco, Mitchell Wortsman, and Ludwig Schmidt. Openflamingo, March 2023.
- [3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [4] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [7] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.
- [8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.
- [9] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.
- [10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- [11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- [12] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.
- [13] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*, 2022.
- [14] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.
- [15] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.
- [16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- [17] Elias Frantar, Saleh Ashkboos, Torsten Hoeffer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- [18] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.
- [19] 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. A framework for few-shot language model evaluation, September 2021.

- [20] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.
- [21] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.
- [22] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [23] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300, 2020.
- [24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.
- [25] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings 15*, pages 217–226. Springer, 2004.
- [26] Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Grounding language models to images for multi-modal generation. *arXiv preprint arXiv:2301.13823*, 2023.
- [27] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023.
- [28] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.
- [29] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.
- [30] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mccnet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.
- [31] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. 2023.
- [32] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. Deja Vu: Contextual Sparsity for Efficient LLMs in the Inference Time. In *ICML*, 2023.
- [33] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.
- [34] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.
- [35] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.
- [36] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.
- [37] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [38] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics.
- [39] Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.
- [40] qwopqwop200. Gptq for llama. <https://github.com/qwopqwop200/GPTQ-for-LLaMa>, 2023.
- [41] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

- [42] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.
- [43] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.
- [44] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.
- [45] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.
- [46] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 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, 2019.
- [47] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [49] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *CVPR*, 2019.
- [50] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.
- [51] Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.
- [52] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *arXiv preprint arXiv:2209.13325*, 2022.
- [53] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models, 2022.
- [54] Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.
- [55] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers, 2022.
- [56] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *CoRR*, abs/1905.07830, 2019.
- [57] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.
- [58] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

A Broader Impacts and Limitations

Broader impacts. In this paper, we propose a general technique to enable accurate and efficient low-bit weight-only quantization of large language models (LLMs). It makes LLMs more efficient and accessible and thus may inherit the impacts of LLMs. On the positive side, quantization helps to democratize LLMs, which helps to benefit more people (especially those with lower income). It reduces the costs and hardware barrier of deploying LLMs and facilitates edge inference of these models, addressing the data privacy issue (since we no longer need to send data to the cloud). On the negative side, LLMs may be exploited by malicious users to produce misinformation and manipulation. Quantization can not prevent such negative effects but it does not make it worse.

Limitations. In this paper, we follow previous work [12, 17, 54, 55, 13] to mostly benchmark the quantized models on standard accuracy metrics like perplexity and accuracy. However, besides accuracy, there are other important metrics for LLM benchmark like robustness, fairness, bias, toxicity, helpfulness, calibration, *etc.* [29]. We think it would be helpful to perform a more holistic evaluation of quantized LLMs covering these aspects, which we leave to future work. Furthermore, we only study low-bit integer quantization of LLMs due to easier data type casting on hardware. There might be a further improvement from changing data types (*e.g.*, FP4 [13]), which we do not include in the study.

B Amount of Computation

We study the post-training quantization (PTQ) of LLMs in this work. The computation requirement is generally modest since we do not rely on any backpropagation. We used one NVIDIA A100 GPU for smaller models (<40B parameters) and 2-4 A100 GPUs for larger models due to memory limits.

The quantization process is generally fast, requiring a few GPU hours (ranging from 0.1 to 3, depending on the model size). The accuracy measurement time depends on the model and dataset sizes: testing LLaMA-65B (the biggest model we tested on multiple datasets) on 4 common sense QA tasks requires 3 GPU hours; testing it on MMLU (consisting of 57 sub-datasets) requires 5 GPU hours. The GPU hours would be smaller for smaller models and datasets (*e.g.*, WikiText-2).

C Limitation with No-group Quantization

Our method searches for good scaling to protect the salient weight channels. It works pretty well under grouped quantization, matching the same accuracy as keeping salient weights in FP16 (Figure 1). However, such a scaling-based method can only protect *one* salient channel for *each group*. It is not a problem for grouped quantization (we only need to protect 0.1%-1% of salient channels, the group size is usually small, like 128, so we need to protect fewer than 1 channel in each group on average). But for no-group quantization, we can only protect one input channel for the *entire weight*, which may not be enough to bridge the performance degradation. As shown in Table 8, under INT3-g128 quantization, AWQ achieves similar performance compared to keeping 1% salient weights in FP16. While under INT3 no-group quantization, there is still a noticeable gap. Nonetheless, we want to stress that the performance of no-group quantization is still far behind grouped quantization at a similar cost. Therefore, grouped quantization is a *more practical solution* for LLM compression for edge deployment and AWQ can effectively improve the quantized performance under this setting.

PPL ↓	FP16	INT3 (group 128)			INT3 (no group)		
		RTN	1% FP16	AWQ	RTN	1% FP16	AWQ
OPT-6.7B	12.29	43.16	13.02	12.99	21160	14.67	18.11
LLaMA-7B	9.49	12.10	10.77	10.82	50.45	14.06	20.52

Table 8. AWQ can match the performance of keeping 1% salient weights in FP16 under grouped quantization without introducing mixed-precisions, but not for no-group quantization. Nonetheless, grouped quantization has a far better performance compared to no-group, making it a far more practical setting for weight-only quantization of LLMs, while AWQ performs quite well under this setting. Results are perplexity on the WikiText-2 dataset.

D OPT Model Performance on MMLU

In this paper, we only report the accuracy of LLaMA models [47] on MMLU [23] but not OPT models [58]. This is because we found OPT models generally have bad performance on the benchmark. As shown in Table 9, the accuracy of OPT models is much worse compared to LLaMA models at a similar scale. Actually, the 13B OPT model underperforms the 7B LLaMA model. Considering the fully random baseline is 25% on MMLU, the OPT model’s accuracy is barely meaningful. Furthermore, OPT models have limited in-context learning

capability (measured by the $\Delta\text{Acc.}$ of 5-shot compared to 0-shot). Therefore, we think benchmarking OPT models on MMLU may not present very meaningful results.

	MMLU (0-shot)	MMLU (5-shot)	$\Delta\text{Acc.}$
OPT-6.7B	27.04%	27.90%	+0.86%
LLaMA-7B	30.78%	38.41%	+7.68%
OPT-13B	27.70%	29.58%	+1.88%
LLaMA-13B	33.29%	45.21%	+11.92%

Table 9. OPT models have limited accuracy on MMLU datasets. Benchmarking OPT on MMLU may not provide meaningful results. Therefore, we only benchmarked OPT models on the WikiText-2 dataset.