

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

Ji Lin^{1*} Jiaming Tang^{1,2*} Haotian Tang¹ Shang Yang¹ Xingyu Dang³ Chuang Gan¹ 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. AWQ outperforms existing work on various language modeling 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. Alongside AWQ, we implement an efficient and flexible inference framework tailored for LLMs on the edge, offering more than $3\times$ speedup over the Huggingface FP16 implementation on both desktop and mobile GPUs. It also democratizes the deployment of the 70B Llama-2 model on mobile GPU (NVIDIA Jetson Orin 64GB).

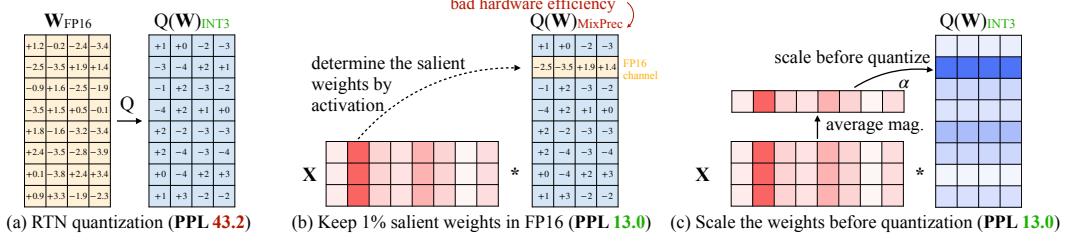
1 Introduction

Large language models (LLMs) based on transformers [40] have shown excellent performance on various benchmarks [4, 49, 38, 34]. 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 [14], which uses second-order information to perform error compensation. It may over-fit the calibration set during reconstruction, distorting the learned features on out-of-distribution domains (Figure 6), which could be problematic since LLMs are *generalist* 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* 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

* indicates equal contributions.



(a) RTN quantization (PPL 43.2)

(b) Keep 1% salient weights in FP16 (PPL 13.0)

(c) Scale the weights before quantization (PPL 13.0)

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.

important features. To avoid the hardware-inefficient mixed-precision implementation, we analyze the error from weight quantization and derive that *scaling up the salient channels can reduce their relative quantization error* (Equation 2). Following the intuition, we designed a per-channel scaling method to automatically search for the optimal scaling that minimizes the quantization error under full-weight quantization. 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. Furthermore, we implemented an efficient serving framework to convert theoretical memory savings from AWQ to practical speedup. Our framework takes advantage of kernel fusion to minimize the inference overhead (*e.g.*, intermediate DRAM access and kernel launch overhead), so that we can better realize the speed up from quantizing linear layers (AWQ is applied to linear layers which consist most of the parameters).

Experiments show that AWQ outperforms existing work on various tasks for different model families (*e.g.*, LLaMA [38], OPT [49]) 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]). With our efficient system implementation, we consistently observe a **3.2-3.3 \times** average speedup compared to the FP16 implementation by Huggingface across a diverse spectrum of LLMs. Furthermore, it facilitates effortless deployment of the Llama-2-70B model on a single NVIDIA Jetson Orin with 64GB of memory. It also democratizes LLMs with up to 13 billion parameters at an interactive pace of 30 tokens per second on a laptop RTX 4070 GPU with only 8GB of memory.

AWQ has been widely adopted by various open-source LLM serving solutions including [FastChat](#), [vLLM](#), [HuggingFace TGI](#), [LMDeploy](#), etc.

2 AWQ: Activation-aware Weight Quantization

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 [9, 14, 47, 46]. In this section, we first propose a weight-only quantization method to improve accuracy *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 [18, 13]. 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 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

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	14.62	119.00	25.03	16.91	16.68	108.71	98.55	98.08	119.76	109.38	61.49
OPT-6.7B	10.86	23.54	11.58	11.39	11.36	23.41	22.37	22.45	23.54	24.23	24.22
OPT-13B	10.13	46.04	10.51	10.43	10.42	46.07	48.96	54.49	44.87	42.00	39.71

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 (↓).

activation significantly improves the quantized performance, even matching a strong reconstruction-based method GPTQ [14]. 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 an alternative method to reduce the quantization error of the salient weight by *per-channel scaling*, which does not suffer from the hardware inefficiency issue.

Analyzing the quantization error. We start by analyzing the error from weight-only quantization. Consider a group/block of weight \mathbf{w} ; the linear operation can be written as $y = \mathbf{w}\mathbf{x}$, and the quantized counterpart is $y = Q(\mathbf{w})\mathbf{x}$. Specifically, the quantization function is defined as:

$$Q(\mathbf{w}) = \Delta \cdot \text{Round}\left(\frac{\mathbf{w}}{\Delta}\right), \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}, \quad (1)$$

where N is the number of quantization bits, and Δ is the quantization scaler determined by the absolute maximum value. Now consider a weight element $w \in \mathbf{w}$, if we multiply w with $s > 1$ and the inversely scale x , we will have $Q(w \cdot s)(x/s)$, which is:

$$Q(w \cdot s) \cdot \frac{x}{s} = \Delta' \cdot \text{Round}\left(\frac{ws}{\Delta}\right) \cdot x \cdot \frac{1}{s}, \quad (2)$$

where Δ' is the new quantization scaler after applying s . We empirically find that: (1) The expected error from $\text{Round}(\cdot)$ (denoted as *RoundErr*) does not vary: since the round function maps a floating-point number to an integer, the error is roughly uniformly distributed from 0-0.5, resulting in an average error of 0.25; (2) Scaling up a single element w usually does not change the extreme value from the group \mathbf{w} . Therefore we have $\Delta' \approx \Delta$; (3) The error from equation 2 can be expressed as $\text{Err}' = \Delta' \cdot \text{RoundErr} \cdot \frac{1}{s}$, the ratio compared to the original error *RoundErr* is $\frac{\Delta'}{\Delta} \cdot \frac{1}{s}$. Given $\Delta' \approx \Delta$ and $s > 1$, the relative error is smaller for the salient weight w .

To verify the idea, we multiply the 1% salient channels with $s > 1$ for the OPT-6.7B model, and measure the change in Δ for each group in Table 2. We find that scaling up the salient channels is quite effective: the perplexity improves from 23.54 for $s = 1$ (simply RTN) to 11.92 for $s = 2$. As s goes larger, the percentage of changed Δ generally gets larger, but the proportion is still quite small for $s < 2$; the relative error for the salient channels continues to go smaller as s increases. Nonetheless, the best PPL actually appears at $s = 2$. This is because if we use a very large s , it will increase the relative error for the *non-salient* channels when Δ increases (the error of non-salient channels will be amplified by $\frac{\Delta'}{\Delta}$, and the ratio is larger than 1 for 21.2% of the channels under $s = 4$), which can damage the model’s overall accuracy. Therefore, we need to also consider the error from the non-salient channels when protecting salient ones.

Searching to scale. To consider both salient and non-salient weights, we choose to automatically search for an optimal (per input channel) scaling factor that minimizes the output difference after

OPT-6.7B	$s = 1$	$s = 1.25$	$s = 1.5$	$s = 2$	$s = 4$
proportion of $\Delta' \neq \Delta$	0%	2.8%	4.4%	8.2%	21.2%
average Δ'/Δ	1	1.005	1.013	1.038	1.213
average $\frac{\Delta'}{\Delta} \cdot \frac{1}{s}$ (error reduction rate)	1	0.804	0.676	0.519	0.303
Wiki-2 PPL	23.54	12.87	12.48	11.92	12.36

Table 2. Statistics when multiplying the 1% salient channels by $s > 1$. Scaling up the salient channels significantly improves the perplexity (23.54 to 11.92). As s goes larger, the percentage of changed Δ increases, and the error reduction rate for salient channels also increases. However, the best perplexity is achieved at $s = 2$, since further increasing s will increase the quantization error for *non-salient* channels.

OPT / PPL↓	1.3B	2.7B	6.7B	13B	30B
FP16	-	14.62	12.47	10.86	10.13
	RTN	119.47	298.00	23.54	46.04
INT3	1% FP16	16.91	13.69	11.39	10.43
g128	$s = 2$	18.63	14.94	11.92	10.80
	AWQ	16.32	13.58	11.39	10.56
					9.77

Table 3. AWQ protects salient weights and reduces quantization error by using a scaling-based method. It consistently outperforms Round-to-nearest quantization (RTN) and achieves comparable performance as mixed-precision (1% FP16) while being more hardware-friendly.

quantization for a certain layer. Formally, we want to optimize the following objective:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} \mathcal{L}(\mathbf{s}), \quad \mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{WX}\| \quad (3)$$

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). \mathbf{s} is a per-(input) channel scaling factor; for $\mathbf{s}^{-1} \cdot \mathbf{X}$, it can usually be fused into the previous operator [44, 46]. 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, 12], which we found still suffers from unstable convergence.

To make the process more stable, we define a *search space* for the optimal scale by analyzing the factors that will affect the choice of scaling factor. As shown in the last section, the saliency of weight channels is actually determined by the activation scale (thus “activation-awareness”). Therefore, we simply use a very simple search space:

$$\mathbf{s} = \mathbf{s}_{\mathbf{X}}^{\alpha}, \quad \alpha^* = \arg \min_{\alpha} \mathcal{L}(\mathbf{s}_{\mathbf{X}}^{\alpha}) \quad (4)$$

\mathbf{s} is only related to the magnitude of activation $\mathbf{s}_{\mathbf{X}}$, and we use a single hyper-parameter α to balance between the protection of salient and non-salient channels. We can find the best α by a fast grid search over the interval of $[0, 1]$ (0 means we do not scale; 1 corresponds to the most aggressive scaling). We further apply weight clipping also by minimizing the MSE error, since clipping the weights can further help to reduce Δ' in Equation 2; thus reducing quantization error. We provide an ablation study on OPT models under INT3-g128 quantization in Table 3; AWQ consistently outperforms round-to-nearest quantization (RTN) and achieves comparable performance as mixed-precision (1% FP16) while being more hardware-friendly.

Advantages. Our method does not rely on any regression [14] 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 6). Therefore, our method requires fewer data for the quantization process and can preserve LLMs’ knowledge outside of the calibration set’s distribution. See Section 3.3 for more details.

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	6.24	5.32	3.74	6.35	5.52	4.61	3.95
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	5.60	4.97	3.41	5.78	5.19	4.21	3.62

Table 4. AWQ improves over round-to-nearest quantization (RTN) for different model sizes and different bit-precisions. It consistently achieves better perplexity than GPTQ (w/ and w/o reordering) on LLaMA & Llama-2 models.

3 Experiments

3.1 Settings

Quantization. We focus on *weight-only grouped* quantization in this work. As shown in previous work [10, 14], 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 [10]. For AWQ, we used a small calibration set from the Pile [15] dataset in order not to overfit to a specific downstream domain. We used a grid size of 20 to search for the optimal α in Equation 4.

Models. We benchmarked our method on LLaMA [38] and OPT [49] families. There are other open LLMs like BLOOM [34], but they are generally worse in quality, so we do not include them in our study. We further benchmark an instruction-tuned model Vicuna [6] and visual language models OpenFlamingo-9B [2] and LLaVA-13B [26] to demonstrate the generability of our method.

Evaluations. Following previous literature [9, 46, 14, 10, 47], we mainly profiled the quantized models on language modeling tasks (perplexity evaluation on WikiText-2 [27]) since perplexity can stably reflect the LLM’s performance [10].

Baselines. Our primary baseline is vanilla round-to-nearest quantization (RTN). It is actually quite strong when using a small group size like 128 [14, 10]. We also compare with a state-of-the-art method GPTQ [14] 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). Other techniques like ZeroQuant [47], AdaRound [28], and BRECQ [23] rely on backpropagation to update the quantized weights, which may not easily scale up to large model sizes; they also do not outperform GPTQ [14], thus not included for study.

3.2 Evaluation

Results on LLaMA models. We focus our study on LLaMA models (LLaMA [38] and Llama-2 [39]) due to their superior performance compared to other open-source LLMs [49, 34]; it is also the foundation of many popular open-source models [36, 6]. We evaluate the perplexity before and after quantization in Table 4. We can see that AWQ consistently outperforms round-to-nearest (RTN) and GPTQ [14] (w/ and w/o reordering) across different model scales (7B-70B) and generations.

Quantization of instruction-tuned models. Instruction tuning can significantly improve the models’ performance and usability [42, 33, 31, 8]. It has become an essential procedure before model deployment. We further benchmark our method’s performance on a popular instruction-tuned model Vicuna [6] in Figure 2. We used the GPT-4 score to evaluate the quantized models’ performance against the FP16 counterpart on 80 sample questions [6]. 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

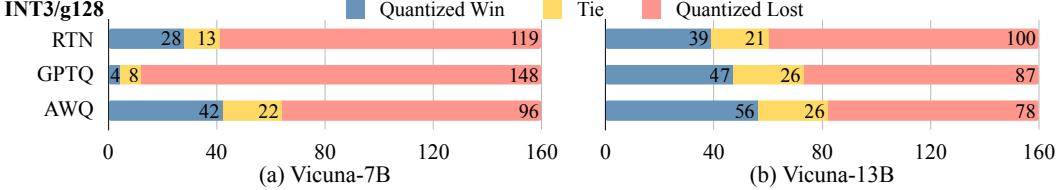


Figure 2. Comparing INT3-g128 quantized Vicuna models with FP16 counterparts under GPT-4 evaluation protocol [6]. More winning cases (in blue) indicate better performance. AWQ consistently improves the quantized performance compared to RTN and GPTQ [14], showing generalization to instruction-tuned models.

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 5. 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.

the INT3-g128 quantized Vicuna models over RTN and GPTQ under both scales (7B and 13B), demonstrating the generability to instruction-tuned models.

Quantization of multi-modal language models. Large multi-modal models (LMMs) or visual language models (VLMs) are LLMs augmented with vision inputs [1, 22, 21, 11, 48, 26]. 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 [5] dataset (Table 5). 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 3 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 [26] model in Figure 4. 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 6). 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.



Figure 3. 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.

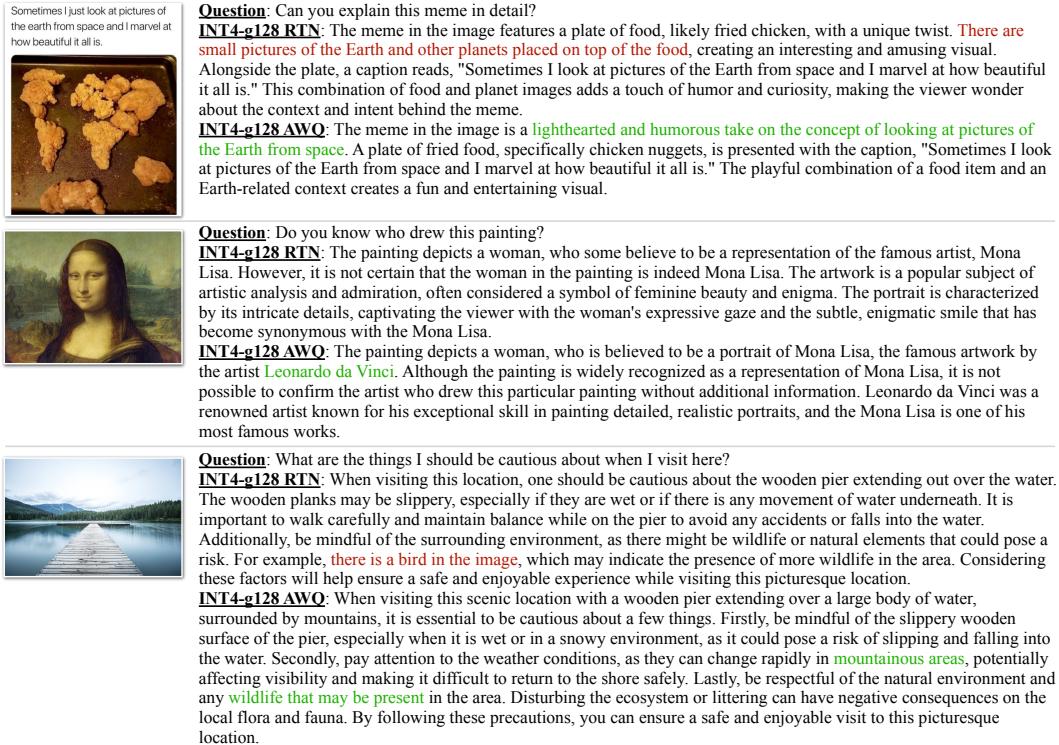


Figure 4. Visual reasoning examples from LLaVA-13B model [26]. AWQ improves over the round-to-nearest (RTN) baseline, providing more reasonable answers. We color the text to show the **correct** or **wrong** responses.

Speedup Evaluation. In Figure 5, we demonstrate the system acceleration results for AWQ. We optimize both linear layers and layers that do not have quantized weights. We conduct benchmarking experiments on RTX 4090 (desktop GPU), RTX 4070 (laptop GPU) and Jetson Orin (mobile GPU). We perform batch size = 1 inference for all LLMs using a fixed prompt length of 4 tokens. We generate 200 tokens for each inference run and calculate the median latency as the final result. As in Figure 5(a), our system brings **2.7-3.9 \times** speedup to three families of LLMs (Llama-2, MPT and Falcon) on 4090 compared with the Huggingface FP16 implementation. Notably, on the laptop 4070 GPU with only 8GB memory, we are still able to run Llama-2-13B models at 33 tokens / second, while the FP16 implementation cannot fit 7B models.

Our system also exhibits promising performance on the NVIDIA Jetson Orin (32GB). As shown in Figure 5(b), our system achieves an interactive processing rate of **33 tokens per second** when running Llama-2 models. Thanks to AWQ, even larger models such as MPT-30B can operate smoothly on this resource-constrained edge device, delivering a processing speed of 7.8 tokens per second. It's worth noting that we implement the forward pass for all AWQ models using native PyTorch APIs, and this code is reused across various GPU architectures. Consequently, our system provides the best of both worlds: state-of-the-art inference speed and exceptional extensibility.

OPT / Wiki PPL \downarrow		1.3B	2.7B	6.7B	13B	30B
FP16	-	14.62	12.47	10.86	10.13	9.56
INT2 g64	RTN	10476	193210	7622	17564	8170
	GPTQ	46.67	28.15	16.65	16.74	11.75
AWQ +GPTQ		35.71	25.70	15.71	13.25	11.38

Table 6. 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.

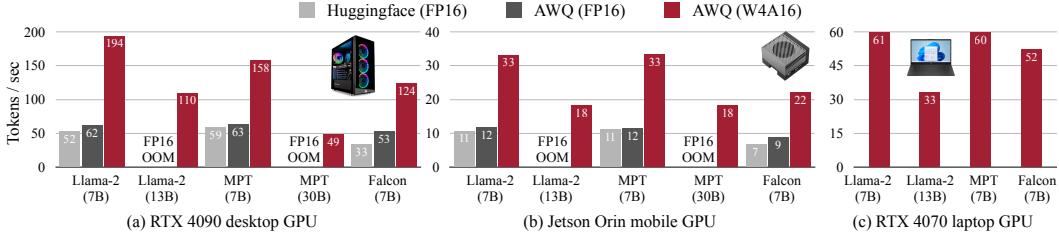


Figure 5. AWQ provides a turn-key solution to transform the theoretical memory footprint reduction into a quantifiable speedup. As a result, AWQ is up to **3.9 \times** and **3.5 \times** faster than the FP16 implementation from Huggingface on 4090 (desktop GPU) and Orin (mobile GPU), respectively. AWQ also democratizes Llama-2-13B deployment on laptop GPUs (4070) with merely 8GB memory.

3.3 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 6 (a). AWQ needs a much smaller calibration to reach a good quantized performance; it can achieve better perplexity using 10 \times smaller calibration set compared to GPTQ (16 sequences v.s. 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 6(b). We took two subsets from the Pile dataset [15]: PubMed Abstracts and Enron Emails [20]. 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.

4 Related Work

Model quantization methods. Quantization reduces the bit-precision of deep learning models [17, 19, 29, 41, 28, 25], which helps to reduce the model size and accelerate inference. Quantization techniques generally fall into two categories: quantization-aware training (QAT, which relies on backpropagation to update the quantized weights) [3, 16, 30, 7] and post-training quantization [19, 29, 28] (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 [9, 46, 47, 45, 43]; (2) Low-bit weight-only quantization (*e.g.*, W4A16), where only weights are quantized into low-bit integers [14, 10, 35, 32]. 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 [14] is the closest to our work. However, the reconstruction process of GPTQ leads to an over-fitting issue to the calibration set and

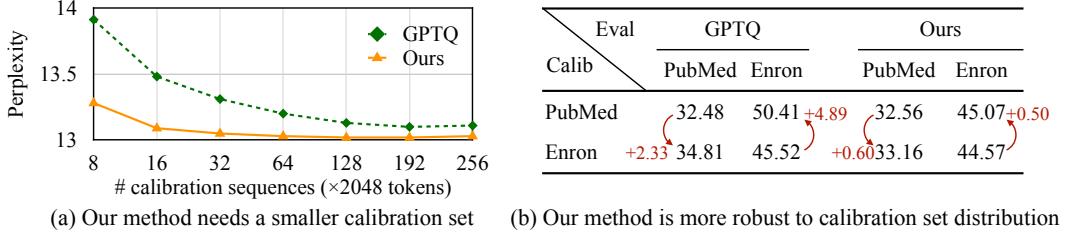


Figure 6. Left: AWQ needs a much smaller calibration set to reach a good quantized performance. It can achieve better perplexity using 10× smaller calibration set compared to GPTQ. **Right:** Our method is more robust to the calibration set distribution. Overall, using the same calibration and evaluation distribution works the best (PubMed-PubMed, Enron-Enron). But when using a different calibration and evaluation 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.

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 [38] and OPT-66B [49]).

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 [14] provides INT3 kernels for OPT models and GPTQ-for-LLAMA extends kernel support for INT4 reordered quantization with the help of Triton [37]. FlexGen [35] and `llama.cpp`^{*} perform group-wise INT4 quantization to reduce I/O costs and offloading. FasterTransformer[†] implements FP16×INT4 GEMM for weight-only per-tensor quantization but does not support group quantization. LUT-GEMM [32] performs bitwise computation on GPU CUDA cores with the help of lookup tables. AWQ kernels are adaptively executed on both tensor cores and CUDA cores, suitable for both context and generation phases in LLM inference. Consequently, we run state-of-the-art LLaMA models with **3.2–3.3×** speedup over the FP16 implementation from Huggingface.

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 can be applicable to instruction-tuned LMs and multi-modal LMs. Our system implementation further translates the theoretical memory savings achieved by AWQ into **3.2–3.3×** measured speedups over the FP16 implementations from Huggingface on desktop and mobile GPUs, democratizing LLM deployment on the edge.

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.

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.

^{*}<https://github.com/ggerganov/llama.cpp>

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

- [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] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*, 2022.
- [11] 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.
- [12] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.
- [13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- [14] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- [15] 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.
- [16] 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.
- [17] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Grounding language models to images for multi-modal generation. *arXiv preprint arXiv:2301.13823*, 2023.
- [22] 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.
- [23] 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.

- [24] 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.
- [25] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.
- [26] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. 2023.
- [27] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] Rohan Taori, Ishaaq 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.
- [37] 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.
- [38] 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.
- [39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [40] 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.
- [41] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *CVPR*, 2019.
- [42] 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.
- [43] 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.
- [44] 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.

- [45] 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.
- [46] 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.
- [47] 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.
- [48] 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.
- [49] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui 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 [9, 14, 46, 47, 10] 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.* [24]. 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 [10]), 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 7, 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 7. 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.