# LQER: Low-Rank Quantization Error Reconstruction for LLMs

**Cheng Zhang** [1] **Jianyi Cheng** [2] **George A. Constantinides** [1] **Yiren Zhao** [1]

## Abstract

Post-training quantization of Large Language Models (LLMs) is challenging. In this work, we introduce **L**ow-rank **Q**uantization **E**rror **R**eduction (LQER), which combines quantization and low-rank approximation to recover the model capbility. LQER leverages an activation-induced scale matrix to drive the singular value distribution of quantization error towards a desirable distribution, which enables near-lossless W4A8 quantization on various LLMs and downstream tasks without the need for knowledge distillation, grid search, or gradient-based iterative optimization. Unlike existing methods, the computation pattern of LQER eliminates the need for specialized Scatter and Gather processes to collect high-precision weights from irregular memory locations. Our W4A8 LLMs achieve near-lossless performance on six popular downstream tasks, while using $1.36\times$ fewer hardware resources than the leading state-of-the-art method. We open-sourced our framework at github.com/ChengZhang-98/lqer.

## 1. Introduction

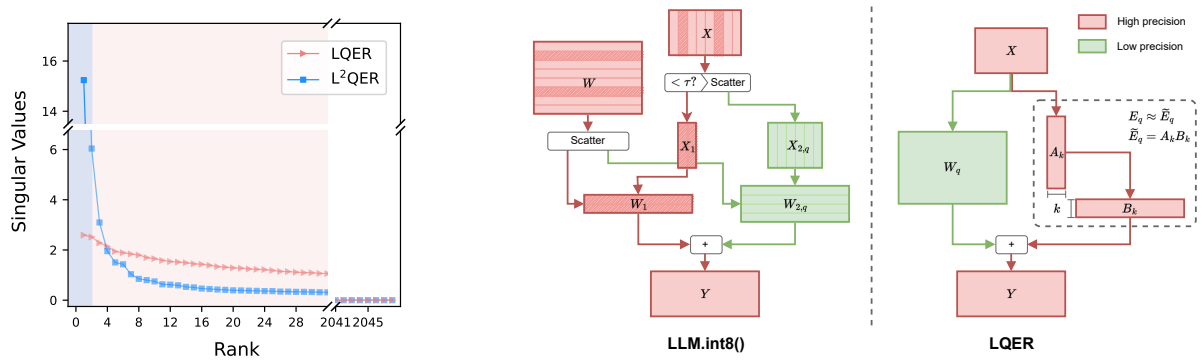
Large Language Models (LLMs) have exhibited impressive capability on various natural language processing (NLP) tasks (Brown et al., 2020). However, the substantial model size and its associated computation costs demand considerable energy and hardware resources. For instance, deploying BLOOM-176B (Workshop et al., 2022) requires 16 NVIDIA A100 GPUs and consumes more than 2000 Watts of total power (Luccioni et al., 2023). Meanwhile, empirical evidence suggests that only models with a sufficiently large parameter count begin to show *emergent capabilities* (Hoffmann et al., 2022), thereby motivates the construction of even larger models. Quantization then emerges as a promising technique to enhance the accessibility of LLMs by reducing the model size and simplifying inference computation.

Low-precision Post-Training Quantization (PTQ) of LLMs has recently become an attractive solution for reducing computational and memory cost (Nagel et al., 2021). However, it remains challenging due to the fact that 1) no further weight training is allowed and 2) the presence of magnitude outliers in model weights and activations. PTQ is a technique that quantizes a pre-trained LLM directly, without additional training, as fine-tuning LLMs usually requires substantial resources. Many researchers have observed that the main building block of LLMs, the transformer layer, produces magnitude outliers in weights and activations (Wei et al., 2022; Bondarenko et al., 2021; Tang et al., 2023). A simple fixed-point quantization then either suffers from considerable clipping or overflow error or from considerable rounding error, depending on the choice of scaling. In both cases, the quantization error propagates and accumulates through the LLMs, leading to substantial task accuracy degradation. To overcome this challenge, recent LLM PTQ methods investigate the statistical properties of LLMs and propose various fine-grained solutions to accommodate (Dettmers et al., 2022; Frantar et al., 2022), mitigate (Xiao et al., 2023; Lee et al., 2023a), or eliminate (Wei et al., 2023; Bondarenko et al., 2023) these numerical outliers.

However, fine-grained treatments to numerical outliers usually come with a high optimization and/or hardware cost. The optimization cost mainly stems from iterative optimization. For example, OmniQuant (Shao et al., 2023) takes 7.3 hours to quantize a LLaMA-30B model with 20 iterations on a single NVIDIA A100 GPU (Lin, 2024). The popular weight-only quantization setup, such as GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2023), dequantizes 4-bit weights to FP16 at runtime, which actually impedes inference on models larger than 7B (Hansen, 2024). Concurrently, many existing quantization frameworks select values from irregular positions for high-precision computation, while maintaining other values in low-precision formats (Dettmers et al., 2023b; Lee et al., 2023b). For in-

[1]Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom [2]Department of Computer Science and Technology, University of Cambridge, Cambridge, United Kingdom. Correspondence to: Cheng Zhang <cheng.zhang122@imperial.ac.uk>, Jianyi Cheng <jianyi.cheng@cl.cam.ac.uk>, George A. Constantinides <g.constantinides@imperial.ac.uk>, Yiren Zhao <a.zhao@imperial.ac.uk>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).




(a) Singular value distributions of quantization error

(b) `LLM.int8()` v.s. LQER

Figure 1: Motivation and computation pattern of LQER. (a) We apply SVD to the quantization error $E_q = W - W_q$ for a 3-bit fixed-point quantized weight in OPT-1.3B, and plot their singular values distributions. Distributions are normalized to have the same Frobenius norm for a fair comparison[1]. Curves with a more asymptotic trend suggest better suitability for low-rank approximation. L$^2$QER displays a much steeper distribution with a smaller number of dominating singular values. (b) LQER approximates a trained weight $W$ with two high-precision yet low-rank matrics $A_k$ and $B_k$, and a low-precision yet high-rank matrix $W_q$. Both components are inexpensive to compute. This estbalishes a regular computation pattern that eliminates the need for irregular memory access like the Scatter and Gather operations in `LLM.int8()`.

stance, `LLM.int8()` (Dettmers et al., 2022) selects activation outliers to compute in half-precision floating-point, while casting the rest to integers. In this work, we propose a simple and efficient LLM PTQ framework that avoids iterative optimization and irregular computation patterns.

Optimizing weight quantization can be considered as a process of minimizing the quantization error $E_q = W - W_q$, where $W_q$ is the quantized weights. We are firstly interested in a novel *inference framework formulation* termed *LQER*. LQER approximates the real value of $W$ through two components ($W \approx \widetilde{E}_q + W_q$): a high-precision yet low-rank matrix $\widetilde{E}_q$ that approximates $E_q$ but with $\mathrm{rank}(\widetilde{E}_q) \ll \mathrm{rank}(W)$; and a low-precision yet high-rank matrix $W_q$ as shown in Figure 1b. Both components are inexpensive to compute and thus work together to reduce the overall computational complexity. Crucially, the high-precision, low-rank component $\widetilde{E}_q$ establishes a regular computation pattern that eliminates need of having the Scatter and Gather operations to fetch and store values from irregular memory locations like `LLM.int8()` (Figure 1b).

In this study, we explore optimizations for $W_q$ using both the integer format and the recently proposed MX number formats (Rouhani et al., 2023b). Additionally, our work emphasizes the design of $\widetilde{E}_q$. Theoretically, assuming the trained weights to be independent and identically distributed (i.i.d.), and given a sufficiently high chosen precision, $E_q$ can be approximated as a random matrix formed by the round-off error. The Marchenko–Pastur distribution suggests that *there exihibits an asymptotic behavior for the distribution of singular values of large random matrices* (Marchenko & Pastur, 1967). We then show the actual singular value distributions of $E_q$ in Figure 1a from a linear layer in OPT-1.3B (Zhang et al., 2022a) (labeled as LQER), showcasing a similar phenomenon to what the *Marchenko–Pastur law* has suggested. Further motivated by the fact that matrices with only a few large singular values like $E_q$ can be effectively approximated by low-rank matrices. We propose to left-multiply $E_q$ by a diagonal matrix $S$, derived from activation magnitudes, that pushes the singular values of $E_q$ toward an even more desirable distribution (labeled as L$^2$QER in Figure 1a). The singular values of $SE_q$ decay more rapidly than $E_q$, with the large singular values of $SE_q$ concentrates in the first few components, as illustrated in Figure 1a[1]. This observation then further motivates our LLM Post-Training Quantization (PTQ) method, ***L**eft-multiply **L**QER (**L**$^2$**QER**)*, designed to recover the performance loss caused by quantization. We make the following contributions in this work:

- We introduce a novel quantized LLM inference framework termed **L**ow-rank **Q**uantization **E**rror **R**eduction (LQER) which combines quantization and low-rank approximation. Unlike existing methods that require gathering values from irregular memory locations, LQER boasts a blocked and regular computation pattern and employs a unified number format for both memory and computation.

- We then propose L$^2$QER, a straightforward but efficient quantization method on top of LQER. L$^2$QER does

[1]To make a fair comparison, we normalize $E_q$ before SVD by multiplying $E_q$ with a scalar $\alpha$ such that the scaled $\alpha E_q$ has the same Frobenius norm as $SE_q$.

Table 1: A summary of recent LLM PTQ methods. Weight-only ($w$-only) and weight-activation ($w\&a$) quantizations are two popular setups. $w$-only quantization generally dequantize values (($\mathrm{dq}(\cdot)$)) back to FP16 before the weight-activation matrix mutiplication at inference time. $w\&a$ quantization performs low-precision mutiplication ($X_qW_q$) at inference-time but requires finding an invertible matrix $S$ to decrease the magnitude range of activations (detail explained in Section 2.1). We shortlist the recent works that belong to the two setups in the last column. * indicates the common precision of $W_q$ and $X_q$ that achieves almost lossless performance on downstream tasks.

| Q setup | WxAy* | Quantization function | Inference-time | Methods |
|---|---|---|---|---|
| $w$-only | W4 | $(W_q, \mathbf{s}) = \mathrm{q}(W)$ | $\widetilde{Y} = X\mathrm{dq}(W_q, \mathbf{s})$ | GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023), Z-fold (Jeon et al., 2023), QuiP (Chee et al., 2023), FlexRound (Lee et al., 2023b), LRQ (Luo et al., 2023) |
| $w\&a$ | W8A8 | $(X_q, \mathbf{s}_t) = \mathrm{q}(XS)$<br>$(W_q, \mathbf{s}_c) = \mathrm{q}(S^{-1}W)$ | $\widetilde{Y}_{i,j} = \mathbf{s}_{t,i}\mathbf{s}_{c,j}(X_{q,i,:} \cdot X_{q,:,j})$<br>$(Y_{q,i,:}, \mathbf{s}'_{t,i}) = \mathrm{q}([\widetilde{Y}_{i,1}, \widetilde{Y}_{i,2}, \dots])$ | SmoothQuant (Xiao et al., 2023), OS+(Wei et al., 2023), AQAS (Lee et al., 2023a), OmniQuant (Shao et al., 2023) |

not need any expensive knowledge distillation, hyper-parameter search, or other forms of iterative optimization. We showcase L$^2$QER's competitiveness with current state-of-the-art methods. L$^2$QER quantizes both weights and activations, it pushes the limit to W4A6, matching the perplexity of OmniQuant (W6A6) on WikiText. Compared to weight-only ($w$-only) quantization methods, our approach outperforms AWQ (W4A16) and maintains quantization activations staying at 8-bit (W4A8).

## 2. Related Work

### 2.1. Post-Training Quantization of LLMs

Post training quantization of LLMs is a challenging task due to presence of numerical outliers. Existing methods can be broadly categorized into two setups: weight-only ($w$-only) and weight-activation ($w\&a$) quantizations. Recent works within these two setups are summarized in Table 1.

**Weight-only quantization** Weight-only quantization usually partitions the trained weight matrix $W$ into groups, with the $i$-th group being quantized using a scale factor $\mathbf{s}_i$:

$$(W_q, \mathbf{s}) = \mathrm{q}(W) \tag{1}$$

where $W_q$ is the quantized weight matrix, $\mathbf{s}$ is a vector of scale factors, and $\mathrm{q}(\cdot)$ denotes quantization function. During inference, the low-precision weights $W_q$ is dequantized back to FP16 before the weight-activation matrix multiply:

$$\widetilde{Y} = X\mathrm{dq}(W_q, \mathbf{s}) \tag{2}$$

Here $X$ is the FP16 input, and $\mathrm{dq}(\cdot)$ is the dequantization function, and $\widetilde{Y}$ is the output. The runtime dequantization cost is negligible in memory-bound scenarios, *e.g.*, small models at small batch sizes. *This cost escalates with model sizes, and eventually impedes inference in compute-bound scenarios* (Hansen, 2024).

GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2023) are two representative $w$-only quantization methods. GPTQ employs second-order information to iteratively round grouped weights and correct the quantization error in the remaining groups. AWQ protects salient weights induced by activations using per-channel scaling. Recent advancements in $w$-only setup include Z-Fold (Jeon et al., 2023) and QuiP (Chee et al., 2023), following GPTQ to correct quantization error. FlexRound (Lee et al., 2023b), and LRQ (Luo et al., 2023) follow AWQ to study finer-grained weight scaling.

**Weight-activation quantization** $w\&a$ quantization utilizes an invertible matrix $S$ to reduce the magnitude range of activations before quantization:

$$(X_q, \mathbf{s}_t) = \mathrm{q}(XS) \tag{3}$$

where $\mathbf{s}_t$ is a vector a per-token scalars. $S^{-1}$ is fused into the weight matrix $W$, and $S$ is fused into the weight matrix of the preceding layer before quantization:

$$(W_q, \mathbf{s}_c) = \mathrm{q}(S^{-1}W) \tag{4}$$

where $\mathbf{s}_c$ is a vector of per-channel scalars. At inference time, the inner product in the activation weight matrix multiplication consists of an inner product of two fixed-point vectors and an FP16 multiplication between the token scalar and channel scalar ("Inference-time" entry in Table 1). Then each row of output activation matrix is quantized back to the input format. This style of quantization avoids the extra dequantization cost in $w$-only setup, but achieving a precision lower than W8A8 while maintaining nearly-lossless model capability proves challenging. Existing $w\&a$ quantization methods lower than 8-bit precision usually suffer from an average downstream task accuracy drop larger than 1% (Shao et al., 2023; Liu et al., 2023a).

SmoothQuant (Xiao et al., 2023) pioneered fusion of an invertible scale matrix into its preceding layer. Outlier Suppression+ (Wei et al., 2023) further introduces a bias matrix

to Equation (4) to shift the mean of activations towards zero and update the layer bias accordingly. Recent works following this line of research include AQAS (Lee et al., 2023a), and OmniQuant (Shao et al., 2023).

Another unique $w\&a$ quantization method, `LLM.int8()` decomposes the FP16 matrix multiplication into a 8-bit fixed-point and an FP16 sub-matrix multiplication using activation thresholds. Despite achieving the closest model capability to FP16, the thresholding, Scatter and Gather operations of `LLM.int8()` are expensive in large models. Similar to `LLM.int8()`, SpQR (Dettmers et al., 2023b) and EasyQuant (Tang et al., 2023) are recent works that retains salient weights in FP16 at finer granularity while quantizing the rest to low-precision.

In this work, we propose a fundamentally different PTQ framework that approximates the real value of weight through two components ($W = \widetilde{E_q} + W_q$), and demonstrate how this formuation helps us to achieve almost lossless PTQ in the $w\&a$ quantization setup with a $W4A8$ configuration.

### 2.2. The `MXINT` Arithmetic

Block floating point is a family of number formats that represents a vector of numbers using a shared exponent or exponent bias. Various block floating point formats have been explored for efficient training or inference in the past few years (Darvish Rouhani et al., 2020; Fox et al., 2020; Zhang et al., 2022b; Drumond et al., 2018). One notable representative is `MXINT`, introduced for hardware-efficient post training quantization (Darvish Rouhani et al., 2020; Rouhani et al., 2023a), this number format has recently been standardized by AMD, Arm, Intel, Meta, Microsoft, NVIDIA, and Qualcomm, for next-generation AI facilities (Micikevicius et al., 2023).

Figure 2 illustrates an example of an `MXINT` vector sharing a 4-bit exponent across four 4-bit mantissas. `MXINT` excels in hardware efficiency compared to floating point, as the inner product of two `MXINT` vectors can be computed as a inner product of two fixed-point numbers plus an exponent addition. Meanwhile, the shared exponent provides a larger dynamic range than fixed point numbers. Recent works indicate that this extended dynamic range fits the activation outliers well in LLM PTQ tasks (Zhang et al., 2023; Rouhani et al., 2023b). In this work, We adopt `MXINT` as the default number format while the idea can be applied to other formats.

### 2.3. Low-Rank Adapters for Fine-Tuning

Low-rank adapter (LoRA) (Hu et al., 2021) is a parameter efficient fine-tuning method for saving GPU memory. LoRA freezes the pretrained weight $W$, and only updates two low-rank weight matrices $L_1$ and $L_2$ during fine-tuning.

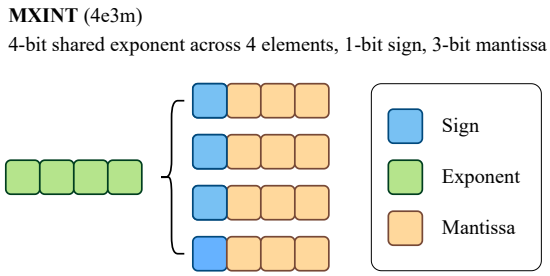



Figure 2: `MXINT` number format (Rouhani et al., 2023b). `MXINT` places a shared exponent across a group of fixed-point numbers. `MXINT` is more hardware efficient than floating point for its simplified vector inner product, and provides a large dynamic range compared to fixed-point numbers. `MXINT` has been standardized recently for next generation AI hardware systems (Micikevicius et al., 2023).

Based on LoRA, QLoRA (Dettmers et al., 2023a) keeps the quantized pretrained weights in memory and only double-dequantizes[2] it in the forward pass to further reduce fine-tuning memory footprints:

$$Y^{\text{BF16}} = X^{\text{BF16}}\text{ddq}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, W^{\text{NF4}}) + X^{\text{BF16}}L_1^{\text{BF16}}L_2^{\text{BF16}} \tag{5}$$

The advantage of LoRA-based methods is that the fine-tuned model can be deployed without extra cost as the low-rank matrices are fused into the pretrained weights after fine-tuning . For QLoRA, the fusion can be expressed as:

$$W_{\text{new}}^{\text{BF16}} = \text{ddq}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, W^{\text{NF4}}) + L_1^{\text{BF16}}L_2^{\text{BF16}} \tag{6}$$

LoftQ (Li et al., 2023b) initializes $L_1$ and $L_2$ with the Singular Value Decompostion (SVD) of quantization errors to achieves a faster fine-tuning convergence than QLoRA.

To our knowledge, LoftQ is the closest work to ours. However, our LQER framework is fundamentally different from the above as it is a PTQ method that does not target fine-tuning. The core idea of LQER is that shaping the singular value distribution of quantization error approximator ($\widetilde{E_q}$) enables a nearly-lossless inference pass quantization. LoftQ fuses the low-rank matrices back to original FP32 weights when deployed, however, the low-rank matrices in LQER remains separate from the quantized weight matrix $W_q$: this allows the matrix multiplications for the low-precision high-rank weight matrix ($W_q$) and the low-rank high-precision matrices to happen in parallel at inference time.

[2]Double-dequantize means the dequantization scales the 4-bit weight matrix twice using $c_1^{\text{FP32}}$ and $c_2^{\text{k-bit}}$

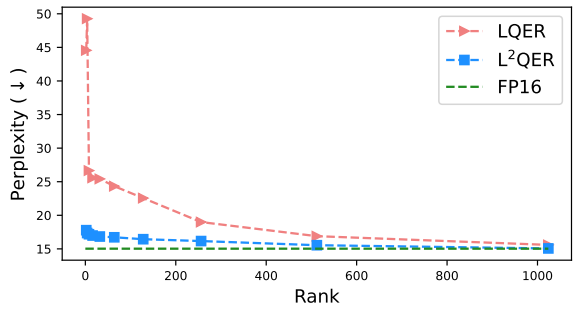



Figure 3: Perplexity (↓) vs rank. We apply W3A8 LQER and L$^2$QER to OPT-1.3B and plot the resultant perplexity. Considering the embedding dimension is 2048, LQER requires a fairly large $k \approx 600$ to reach a perplexity close to FP16 . In comparison, a small $k \approx 64$ is enough for L$^2$QER Comparison of perplexity (↓) and quantization error reconstruction between LQER and L$^2$QER.

## 3. Method

We aim to approximate the multiplication by a large dense weight matrix $W$ in a low-cost way. This low cost can be achieved through low-precison quantization or low-rank approximation. Quantization simplifies the multiplication arithmetic, while low-rank approximation reduces the overall number of computations. We judiciously combine the two: approximate $W$ as a dense low-precision matrix $W_q$ and then correct the error induced using a high-precision low-rank correction term as illustrated in Figure 1b.

### 3.1. LQER: Approximate $E_q$ using SVD

Our idea is to reconstruct the quantization error matrix $E_q$ through SVD-based low rank approximation. When a quantization is applied to a trained FP32/FP16 weight matrix $W \in \mathbb{R}^{m\times n}$, the resulting quantization error matrix $E_q$ is:

$$E_q = W - W_q \tag{7}$$

where $W_q = \mathrm{q}(W)$ is the quantized weight matrix, and $\mathrm{q}(\cdot)$ represents the quantization function. A straightforward way to reconstruct the error is to use SVD-based low-rank approximation:

$$E_q = U\Sigma V^T \approx U_k\Sigma_k V_k^T \tag{8}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix of singular values. $U_k \in \mathbb{R}^{m\times k}$, $V_k \in \mathbb{R}^{n\times k}$ and $\Sigma_k \in \mathbb{R}^{k\times k}$ are the sub-matrices of $U$, $V$ and $\Sigma$ corresponding to the largest $k$ singular values.

If two high-precision matrices $A_k = U_k$ and $B_k = \Sigma_k V_k^T$ are assigned to approximate $E_q$, *i.e.*, $A_kB_k \approx E_q$, the linear layer can be approximated as:

$$\begin{aligned}\widetilde{Y} &= XW_q + (XA_k)B_k \\ &= X(W_q + A_kB_k) \\ &= X(W_q + \widetilde{E_q}) \\ &\approx X(W_q + E_q) \\ &= XW\end{aligned} \tag{9}$$

where $X \in \mathbb{R}^{t\times m}$ and $\widetilde{Y} \in \mathbb{R}^{t\times n}$ are the layer input and the approximated output, and $t$ is the sequence length. We use $b_l$ and $b_h$ to denote the bitwidth of low-precision matrix ($W_q$) and high-precision matrices ($X$, $A_k$ and $B_k$) respectively. A pair of $b_l$ and $b_h$, *e.g.*, $(b_l, b_h) = (3, 8)$, means that we compensate the quantization error of 3-bit weight using two 8-bit low-rank matrices. We refer to this design of the inference flow as LQER.

At inference-time, LQER runs one low-precision but large matrix multiplication ($XW_q$) and two high-precision but small matrix multiplications ($XA_k$ and $(XA_k)B_k$) in parallel to save memory and achieve a speedup. Given a low-precision quantization $\mathrm{q}(\cdot)$, adjusting the rank $k$ allows tuning the trade-off between the computational cost and the model accuracy. Specifically:

- In LLMs, $W \in \mathbb{R}^{m\times n}$ is usually a large matrix. For example, $(m, n)$ is $(12288, 12288)$ , $(12288, 49152)$, or $(49152, 12288)$ in OPT-175B. A low-precision $W_q$ significantly reduces the memory footprint and $XW_q$ is faster than $XW$.
- Two high-precision but small matrices $A_k \in \mathbb{R}^{m\times k}$ and $B_k \in \mathbb{R}^{k\times n}$ estimate the quantization error at the cost of minimal computation. For a token $\mathbf{x} \in \mathbb{R}^m$, the two matrix multiplies $(\mathbf{x}A_k)$ and $((\mathbf{x}A_k)B_k)$ only introduce $(m + n) \times k$ high-precision multiplies in total while the unquantized $\mathbf{x}W$ has $m \times n$ high-precision multiplies. For the FNN layers in OPT-175B, the newly introduced multiplications is around $\frac{(m+n)\times k}{m\times n} \approx 0.01 \times k\%$.

The ideal case is that a small $k \ll \min(m, n)$, *e.g.*, $k = 32$, would successfully recover the model's accuracy/perplexity. However, our experiments reveal that the singular values of $E_q$ decay slowly for most linear layers, requiring a sufficiently large $k$ to recover the accuracy/perplexity. Figure 3 illustrates the perplexity of quantized W3A8 OPT-1.3B versus rank $k$. For LQER, a small $k \approx 64$ still falls short compared to the FP16 baseline. The following section then discusses how we can achieve a low $k$ value by analytically scaling the error term.

### 3.2. L$^2$QER: Shape Singular Value Distribution of Quantization Errors using Activation Statistics

Recent works have shown that partially preserving the weight precision according to activation magnitude recovers

the model's accuracy/perplexity. `LLM.int8()` decomposes an FP16 matrix multiply into one FP16 sub-matrix multiply for large activation magnitudes and one 8-bit fixed-point sub-matrix multiply for the rest at runtime. AWQ also presents an experiment that effectively recovers accuracy by preserving the 1% salient weights corresponding to large activation magnitudes in FP16, and quantizing other weights to 4-bit grouped fixed-point.

Motivated by this phenomenon, we propose a novel quantization error reconstruction method, named L$^2$QER, that scales the quantization error matrix $E_q$ before applying SVD and undo the scaling in low-rank matrices. We first left-multiply $E_q$ with a diagonal matrix $S = \text{diag}(s_1, s_2, \ldots, s_m)$ to scale $i$-th row of $E_q$ by a distinct scalar $s_i$, then apply SVD to the scaled matrix $SE_q$:

$$SE_q = U'\Sigma'V'^T \approx U'_k\Sigma'_k V'^T_k \tag{10}$$

where $S$ is calibrated from the pre-training data. To calculate $s_i$, we first average the $i$-th channel magnitudes across all the tokens in each calibration sample, then find the maximum average value among all samples. A detailed calculation of $S$ is in Appendix A. The calibration requires no training.

The intuition behind the scaling is that the quantization error corresponding to large activation magnitudes, *i.e.*, the salient weights identified by corresponding activation magnitudes, should be more precisely approximated. Hence, we scale up these quantization errors before SVD.

High precision $A'_k$ and $B'_k$ are employed to cancel out $S$ and reconstruct $E_q$:

$$\begin{cases} A'_k = S^{-1}U'_k \\ B'_k = \Sigma'_k V'^T_k \end{cases} \tag{11}$$

where $S^{-1}$ is the inverse of the diagonal matrix $S$. $S^{-1}$ always exists in practice since no diagonal elements in $S$ is zero (no channels in LLM activations are always zero). Now we approximate the linear layer similarly to LQER:

$$\begin{aligned} \widetilde{Y} &= XW_q + (XA'_k)B'_k \\ &= XW_q + (XS^{-1}U'_k)(\Sigma'_k V'^T_k) \\ &= X\left(W_q + S^{-1}\widetilde{(SE_q)}_k\right) \\ &\approx XW \end{aligned} \tag{12}$$

where $\widetilde{Y}$ and $\widetilde{(SE_q)}_k = U'_k\Sigma'_k V'^T_k$ are the approximated $Y$ and the approximated $(SE_q)$ of rank $k$. Note that the term $\widetilde{E}_q := S^{-1}\widetilde{(SE_q)}_k$ is the approximated quantization error.

As shown in Figure 1a, $S$ drives the singular value distribution to decay faster than LQER with large singular values concentrating in the first few components, and the scaling is counteracted by $S^{-1}$ in $A'_k$; therefore, L$^2$QER tends to recover more model capability than LQER. In Figure 3, L$^2$QER recovers the perplexity close to FP16 baseline at a very small $k \approx 64$. In Section 4.3, we will show that L$^2$QER achieves nearly lossless W4A6 LLM PTQ results comparable to state-of-the-art W6A6/W4A16 methods but with higher hardware efficiency.

Table 2: Perplexity (↓) of plain `MXINT`, LQER, and L$^2$QER on OPT-1.3B and LLaMA-7B. We apply plain `MXINT` quantization, LQER, and L$^2$QER to OPT-1.3B and LLaMA-7B in the same W4A8 setup. The decreasing perplexity proves the effectiveness of the quantization error reconstruction in LQER, and activation-induced scale matrix $S$ in L$^2$QER.

| | `MXINT` | LQER | L$^2$QER | FP16 |
|---|---|---|---|---|
| OPT-1.3B | 16.42 | 15.28 | 15.02 | 14.63 |
| Δ PPL (↓) | +1.78 | +0.65 | +0.39 | - |
| LLaMA-7B | 6.17 | 6.06 | 5.89 | 5.67 |
| Δ PPL (↓) | +0.50 | +0.39 | +0.22 | - |

## 4. Experiments

### 4.1. Experimental Setup

**Quantization** We use `MXINT` as the number format of LQER if not specified. In Section 4.3, we use W4A8 L$^2$QER with $k = 32$ to compare with both 4-bit $w$-only and 4-/6-/8-bit $w\&a$ quantization methods. In Section 4.4, we use W2A8 L$^2$QER with $k = 256$ to compare with 2-bit $w$-only quantization methods. In both subsections, `MXINT` activation matrices have 8-bit shared exponents to accomodate activation outliers, while weight matrices and low-rank matrices have 4-bit shared exponents. The block size of `MXINT` is the default [1, 16] in the original paper (Darvish Rouhani et al., 2020) for $X_q$ ([16, 1] for $W_q$, $A_k$, and $B_k$).

**Models and Baselines** We benchmarked our methods on the OPT family (Zhang et al., 2022a), the LLaMA family (including LLaMA (Touvron et al., 2023a), LLaMA-2 (Touvron et al., 2023b), Vicuna-v1.5 (Zheng et al., 2023)), and Mistral (Jiang et al., 2023). These are the representative or state-of-the-art model open-sourced for research across various model sizes and architectures. We compare our methods with FP16 model, `LLM.int4()`[3], GPTQ, AWQ, AQAS, OmniQuant[4], and QuiP. The later two have variants optimized for extremely low-precision quantization. We take the reported WikiText2 perplexity or downstream task

[3] `LLM.int4()` denotes the 4-bit verision of `LLM.int8()` open-sourced in bitsandbytes.

[4] We take W6A6 OmniQuant as an weight-activation quantization baseline, and W2A16 as a 2-bit weight-only quantization baseline.

Table 3: A comparison of perplexity(↓) on WikiText-2. Best results are marked in **bold**, and second best results are underlined in $w\&a$ setup. In a $w$-only setup, L$^2$QER-`INT` outperforms GPTQ and is on par with AWQ, while offering substantially lower hardware costs. In a $w\&a$ setup, L$^2$QER-`MXINT` outperforms all other competitors both in terms of perplexity and hardware efficiency. $^*$ means `LLM.int4()` casts the weight sub-matrices corresponding to activation outliers to 4-bit fixed-point before computation and cast them back to FP16 after, thus the weight formats in memory is FP16. † means OmniQuant and AQAS use per-channel and per-token scaled quantization. $^\ddagger$ means LLaMA-2 results were not available in (Lee et al., 2023a) and the author has not open-sourced AQAS code.

| Q Setup | Method | Q Config | OPT | | | LLaMA | | | LLaMA-2 | | | Avg. Δ PPL (↓) | Avg. $w$ bits | Circuit area (↓) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 6.7B | 13B | 30B | 7B | 13B | 33B | 7B | 13B | 70B | | | |
| - | FP16 | - | 10.86 | 10.13 | 9.56 | 5.67 | 5.10 | 4.10 | 5.48 | 4.90 | 3.32 | - | 16 | 1× |
| $w$-only | GPTQ | INT4, g128 | 11.39 | 10.31 | 9.63 | 6.12 | 5.21 | 4.24 | 5.69 | 4.98 | 3.51 | 0.22 | 4.1 | 13.99× |
| | AWQ | INT4, g128 | **10.93** | **10.21** | 9.59 | **5.78** | **5.20** | **4.22** | 5.61 | 4.98 | **3.42** | **0.09** | 4.1 | 13.99× |
| | L$^2$QER-`INT` | INT4, g128 | 10.99 | 10.24 | **9.57** | 5.89 | **5.20** | 4.24 | **5.58** | **4.96** | **3.42** | 0.11 | 4.3 | **1.34**× |
| $w\&a$ | `LLM.int4()` | $\tau = 6.0$ | 11.23 | 10.39 | 10.01 | 6.05 | 5.31 | 4.33 | 5.77 | 5.06 | 3.51 | 0.29 | 16$^*$ | 21.23× |
| | OmniQuant † | W6A6, per-c/t | **10.96** | **10.21** | **9.62** | 5.96 | 5.28 | 4.38 | 5.87 | 5.14 | 3.72 | 0.20 | 6.0 | 0.39× |
| | AQAS † | W4A8, per-c/t | 13.42 | 12.19 | 11.08 | 6.69 | 5.81 | 5.14 | -$^\ddagger$ | -$^\ddagger$ | -$^\ddagger$ | 1.45 | **4.0** | 0.45× |
| | L$^2$QER-`INT` | W4A8, g128 | 11.10 | 10.38 | 9.72 | 6.09 | 5.31 | 4.35 | 5.85 | 5.10 | 3.51 | 0.25 | 4.1 | 0.33× |
| | L$^2$QER-`MXINT` | W4A6 | 11.03 | 10.32 | 9.72 | 5.92 | 5.24 | 4.28 | 5.73 | 5.05 | 3.46 | 0.18 | 4.3 | **0.23**× |
| | L$^2$QER-`MXINT` | W4A8 | 11.00 | 10.27 | 9.69 | **5.89** | **5.21** | **4.25** | **5.69** | **5.02** | **3.44** | **0.15** | 4.3 | 0.33× |

accuracy from the original papers if available.

**Evaluation** We report the perplexity on WikiText-2 (Merity et al., 2016) and the accuracy on ARC (easy) (Clark et al., 2018), ARC (challenge) (Clark et al., 2018), LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), OpenBookQA (Mihaylov et al., 2018), and BoolQ (Clark et al., 2019) using the `lm-eval-harness` evaluation flow (Gao et al., 2023). Ideally a calibration dataset should be sampled from the pretraining dataset to calculate the activation-induced scale matrix $S$. However, none of the LLMs mentioned above open-sourced their pretraining datasets. We create a subset of SlimPajama (Soboleva et al., 2023) with Wikipedia texts excluded as the calibration dataset. This calibration dataset contains only 32 samples of 2048 tokens. As mentioned previously in Section 3.2, this calibration simply profiles values without having any SGD-based training. We also report the weight average bitwidth for memory efficiency and estimate the circuit area for the hardware cost. Circuit area is estimated with the number of Look Up Tables (LUTs) of the processing engines (PEs) if implemented on FPGAs, which is also approximately proportional to the number of gates if implemented as ASICs. We have faithfully implemented these arithmetic cores and inputted them into FPGA synthesis flows, obtaining results for circuit area. This is because `MXINT` is a newly release arithmetic standard (Micikevicius et al., 2023). Appendix D provides the detailed circuit area estimation.

### 4.2. LQER and L$^2$QER

We first focus on comparing variants of LQER in Table 2. We evaluate the variants in a W4A8 $w\&a$ quantization setup on both OPT-1.3B and LLaMA-7B. We show the results of plain `MXINT`, LQER, and L$^2$QER, where plain `MXINT` means the whole network is simply `MXINT` quantized without any special treatments.

Table 2 indicates that a plain W4A8 `MXINT` quantization leads to substantial performance degradation (ΔPPL = +1.78 on OPT-1.3B). LQER verifies that reconstructing the quantization error of weight helps to recover the model performance. Activation-induced $S$ in L$^2$QER further pushes the performance of LQER to be even closer to the FP16 baseline. In the following sections, we then mainly focus on presenting L$^2$QER results.

### 4.3. Comparing with Existing Quantization Methods

We present the perplexity (↓), the average increased perplexity over models (Δ PPL (↓)), average weight bitwidth, and circuit area (↓) of L$^2$QER and existing $w$-only/$w\&a$ methods in Table 3. Then we exclude the methods with obvious performance degradation and evaluate the average downstream task performance in Table 4. We additionally include a fixed-point version of L$^2$QER as a baseline. Best results in each setup are marked in **bold** and second best results are underlined.

**WikiText-2** In the $w$-only quantization setup, L$^2$QER-`INT` achieves a significantly better perplexity when compared to GPTQ and is on par with AWQ (both only around 0.1 higher than FP16), while has a substantially smaller hardware cost (circuit area). In the $w\&a$ setup, L$^2$QER-`MXINT` has a perplexity around 0.15 higher than FP16 when it is W4A8. L$^2$QER-`MXINT` outperforms state-of-the-art sub-8-bit methods by a significant margin. The perplexity of L$^2$QER is around 0.05 higher than OmniQuant on the OPT family, but consistently outperforms OmniQuant on LLaMA

Table 4: A comparison of downstream task accuracy (↑), averaged across six downstream tasks. **Bold** text indicates the best results, while underscore denotes the second-best. L$^2$QER *achieves the best accuracy among all LLaMA models*, and nearly lossless (around 0.3% drop) compared to the FP16 baseline. $^*$ means the results are not available in the original GPTQ paper, and we did not find open-source implementations and/or model checkpoints to run evaluation. $^\dagger$ means the results of OPT and LLaMA-2 are not reported in the original OmniQuant paper. For LLaMA-1, LAMBADA and OpenbookQA are not included in OmniQuant either, thus we replace the results of these two tasks with FP16 results *as an estimated upper limit* of OmniQuant. OmniQuant-r is the results we replicated using the official implementation[5] and checkpoints[6].

| Method | Q Config | OPT | | | LLaMA | | | LLaMA-2 | | | Avg. Δ Accu. (↑) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 6.7B | 13B | 30B | 7B | 13B | 33B | 7B | 13B | 70B | |
| FP16 | - | 55.6% | 56.2% | 59.1% | 63.2% | 65.0% | 68.4% | 63.5% | 66.5% | 69.9% | - |
| GPTQ | INT4, g128 | **55.4%** | 56.4% | -* | 60.8% | 64.7% | 66.7% | 62.2% | 65.9% | 69.8% | -0.9% |
| AWQ | INT4, g128 | 55.3% | 56.4% | **58.9%** | 62.5% | 64.8% | **68.0%** | 62.9% | 65.9% | 69.9% | -0.4% |
| `LLM.int4()` | $\tau = 6.0$ | **55.4%** | 55.9% | 58.0% | 62.2% | 64.6% | 67.7% | 62.6% | 65.8% | 69.9% | -0.7% |
| OmniQuant$^\dagger$ | W6A6, per-c/t | - | - | - | 58.4% | 59.2% | 61.0% | - | - | - | -6.0% |
| OmniQuant-r$^\dagger$ | W6A6, per-c/t | **55.4%** | 56.1% | 58.6% | 47.0% | 48.2% | 49.9% | 47.2% | 49.4% | 58.6 | -11.0% |
| L$^2$QER-`INT` | W4A8, g128 | 54.1% | 56.2% | 57.7% | 61.7% | 64.4% | 67.4% | 62.2% | 65.9% | 69.7% | -1.0% |
| L$^2$QER-`MXINT` | W4A6 | 54.7% | 56.2% | 58.5% | 62.7% | **64.9%** | 67.8% | 63.0% | 65.8% | 69.9% | -0.5% |
| L$^2$QER-`MXINT` | W4A8 | 55.1% | **56.5%** | 58.4% | **63.0%** | 64.8% | **68.0%** | **63.1%** | **66.1%** | 69.9% | **-0.3%** |

family. Note that OmniQuant was trained on WikiText2 for 20 epochs (Shao et al., 2023), but L$^2$QER only proifles the activation magnitudes using 32 samples from a calibration dataset with Wikipedia texts excluded.

**Downstream task accuracy** We reuse the quantization setup in Table 3 and conduct a thorough evaluation on downstream tasks, including ARC (easy), ARC (challenge), LAMBADA, PIQA, OpenBookQA and BoolQ and report the results in Table 4. The average accuracy of L$^2$QER on the six downstream tasks is better than other quantization methods on LLaMA models, and nearly lossless (around 0.3% drop) compared to the FP16 baseline. We reproduced the WikiText2 perplexity reported in OmniQuant paper (Shao et al., 2023) using the official implementation[5] and checkpoints[6], but failed to reproduce their downstream accuracy performance on LLaMA models. We refer to these mismatched OmniQuant results as OmniQuant-r in Table 4. We attribute the inconsistant behaviour of OmniQuant to its iterative quantization parameter training on WikiText2, which is further discussed in Appendix C. Nevertheless, our method has demonstrated substantially better downstream task capabiliteis, with a much lower hardware cost (circuit area in Table 3). A detailed discussion about hardware cost is in Appendix D. A complete table including the accuracy of each individual task is in Appendix E.

**AlpacaEval** We also evaluate the performance of L$^2$QER on AlpacaEval (Li et al., 2023a), an evaluator for instruction-following language models. We use AlpacaEval to measure the fraction of times GPT-4 Turbo prefers the outputs from the quantized model over outputs from a reference model. Here we use AWQ as the reference model and report the results of LLaMA-2-7B-Chat and LLaMA-2-13B-Chat. We observe that L$^2$QER is competitive with AWQ in both length-controlled win rate and normal win rate.

Table 5: AlpacaEval results. We use GPT-4 Turbo as the evaluator and AWQ as the reference model. The results are collected after evaluating LLaMA-2-7B-Chat/-13B-Chat on all samples. We find that L$^2$QER is competitive with AWQ in both length-controlled win rate and normal win rate.

| Model | Gen. vs Ref. | Length-controlled win rate | Win rate |
|---|---|---|---|
| 7B | L$^2$QER vs AWQ | 56.06 % | 55.32 % |
| 13B | | 52.90 % | 52.51 % |

**Hardware efficiency** L$^2$QER is more hardware friendly than the baselines. We highlight the last two columns of average weight bits and circuit area in Table 3. L$^2$QER requires less circuit area to implement a MACs when the model performance (perplexity and downstream task accuracy) and the MAC throughput are roughly matched with the baseline. We offer circuit area breakdowns of `LLM.int4()`, AWQ, and L$^2$QER in the Table 7, Table 8, and Table 9 in Appendix D.

**Optimization cost** The optimization of LQER is also efficient. The calibration and quantiation of LLaMA-33B takes around 1.2 hours in total on a single NVIDIA A100 GPU. In contrast, OmniQuant takes 7.3 hours to optimize the quantization parameters for LLaMA-33B. Furthermore, the optimization of LQER can be fully parallelized to be

[5]https://github.com/OpenGVLab/OmniQuant
[6]https://huggingface.co/ChenMnZ/OmniQuant/tree/main

faster, since there is no dependency between the quantization of each linear layer such as fusing the scale matrices to preceding layers in SmoothQuant or knowledge distillation in LLM-QAT (Liu et al., 2023b).

**Other model families** To fully evaluate the adaptiveness of L$^2$QER across model families, we have also conducted experiments to evaluate its effectiveness on Vicuna and Mistral. Vicuna is an instruction-tuned LLaMA. Mistral uses Grouped-Query Attention (GQA) (Ainslie et al., 2023) and windowed attention (Beltagy et al., 2020). The results of Vicuna-v1.5-7B/13B and Mistral-7B are included in Appendix E. These results reveal a pattern consistent with other models, indicating that L$^2$QER is agnostic to various LLM families.

### 4.4. 2-bit Quantization

To explore the limit of L$^2$QER, we evaluate L$^2$QER in the 2-bit quantization setup. Table 6 compares L$^2$QER with OmniQuant and QuiP#[7], which are both recent works optimized for extremely low-precision LLM quantization. We observe that 2-bit quantization is challenging for existing methods including L$^2$QER. These methods perform inconsistently with model sizes and families (Table 10 in Appendix E). Unlike a simple rank $k = 32$ for W4A8 quantization, L$^2$QER requires a larger $k$ for 2-bit quantization.

Table 6: 2-bit quantization perplexity (↓) on WikiText2. OmniQuant and QuiP#[7] are two state-of-the-art methods for extremely low-precision LLM quantization. We found 2-bit quantization is still challenging for existing methods.

| Q Setup | Method | Q Config | 7B | 13B |
|---|---|---|---|---|
| - | FP16 | - | 5.67 | 5.10 |
| $w$-only | AWQ | INT2 g128 | 2.6e5 | 2.8e5 |
| | QuiP# | INT2 g128 | 10.97 | 8.43 |
| | OmniQuant | INT2 g128 | 12.97 | 10.36 |
| $w\&a$ | L$^2$QER | $k = 256$ | **10.30** | **8.42** |

## 5. Conclusion

In this work, we propose a novel LLM post-training quantization framework, LQER, which judiciously combine quantization and low-rank approximation to recover model capbility. We then further propose L$^2$QER, which leverages an activation-induced scale matrix to shape the singular values of quantization error towards a desirable distribution that can be accurate approximated. L$^2$QER achieves nearly-losses perplexity (around 0.15 higher than FP16) and an average accuracy drop of only around 0.3% on six different downstream tasks. The regular computation pattern of LQER ensures a higher hardware efficiency than existing methods and takes 67% smaller circuit area than FP16.

[7]QuiP# is an improved version of QuiP released by the same research group: https://github.com/Cornell-RelaxML/quip-sharp

## Impact Statement

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

## References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Bondarenko, Y., Nagel, M., and Blankevoort, T. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

Bondarenko, Y., Nagel, M., and Blankevoort, T. Quantizable transformers: Removing outliers by helping attention heads do nothing. *arXiv preprint arXiv:2306.12929*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chee, J., Cai, Y., Kuleshov, V., and De Sa, C. Quip: 2-bit quantization of large language models with guarantees. *arXiv preprint arXiv:2307.13304*, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Darvish Rouhani, B., Lo, D., Zhao, R., Liu, M., Fowers, J., Ovtcharov, K., Vinogradsky, A., Massengill, S., Yang, L., Bittner, R., et al. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in neural information processing systems*, 33: 10271–10281, 2020.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023a.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023b.

Drumond, M., Lin, T., Jaggi, M., and Falsafi, B. Training dnns with hybrid block floating point. *Advances in Neural Information Processing Systems*, 31, 2018.

Fox, S., Rasoulinezhad, S., Faraone, J., Leong, P., et al. A block minifloat representation for training deep neural networks. In *International Conference on Learning Representations*, 2020.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Hansen, C. Autoawq. https://github.com/casper-hansen/AutoAWQ, 2024.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jeon, Y., Lee, C., Park, K., and Kim, H.-y. A frustratingly easy post-training quantization scheme for llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14446–14461, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Lee, J., Kim, M., Baek, S., Hwang, S. J., Sung, W., and Choi, J. Enhancing computation efficiency in large language models through weight and activation quantization. *arXiv preprint arXiv:2311.05161*, 2023a.

Lee, J. H., Kim, J., Kwon, S. J., and Lee, D. Flexround: Learnable rounding based on element-wise division for post-training quantization. *arXiv preprint arXiv:2306.00317*, 2023b.

Li, X., Zhang, T., Dubois, Y., Taori, R., Gulrajani, I., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 2023a.

Li, Y., Yu, Y., Liang, C., He, P., Karampatziakis, N., Chen, W., and Zhao, T. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023b.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Lin, L. LLM-Tracker: OmniQuant, 2024. URL https://llm-tracker.info/howto/OmniQuant.

Liu, J., Gong, R., Wei, X., Dong, Z., Cai, J., and Zhuang, B. Qllm: Accurate and efficient low-bitwidth quantization for large language models. *arXiv preprint arXiv:2310.08041*, 2023a.

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023b.

Luccioni, A. S., Viguier, S., and Ligozat, A.-L. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253): 1–15, 2023.

Luo, Y., Gao, Y., Zhang, Z., Fan, J., Zhang, H., and Xu, M. Long-range zero-shot generative deep network quantization. *Neural Networks*, 166:683–691, 2023.

Marchenko, V. and Pastur, L. Distribution of eigenvalues for some sets of random matrices. *Mat. Sb*, 72:507–536, 1967.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Micikevicius, P., Oberman, S., Dubey, P., Cornea, M., Rodriguez, A., Bratt, I., Grisenthwaite, R., Jouppi, N., Chou, C., Huffman, A., Schulte, M., Wittig, R., Jani, D., and Deng, S. Ocp 8-bit floating point specification (ofp8), 2023. URL https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., Van Baalen, M., and Blankevoort, T. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Rouhani, B., Zhao, R., Elango, V., Shafipour, R., Hall, M., Mesmakhosroshahi, M., More, A., Melnick, L., Golub, M., Varatkar, G., et al. Shared microexponents: A little shifting goes a long way. *arXiv preprint arXiv:2302.08007*, 2023a.

Rouhani, B. D., Zhao, R., More, A., Hall, M., Khodamoradi, A., Deng, S., Choudhary, D., Cornea, M., Dellinger, E., Denolf, K., et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023b.

Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., Li, Z., Zhang, K., Gao, P., Qiao, Y., and Luo, P. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Tang, H., Sun, Y., Wu, D., Liu, K., Zhu, J., and Kang, Z. Easyquant: An efficient data-free quantization algorithm for llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9119–9128, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wei, X., Zhang, Y., Zhang, X., Gong, R., Zhang, S., Zhang, Q., Yu, F., and Liu, X. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.

Wei, X., Zhang, Y., Li, Y., Zhang, X., Gong, R., Guo, J., and Liu, X. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

Workshop, B., Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Zhang, C., Cheng, J., Shumailov, I., Constantinides, G., and Zhao, Y. Revisiting block-based quantisation: What is important for sub-8-bit LLM inference? In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9988–10006, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.617. URL https://aclanthology.org/2023.emnlp-main.617.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022a.

Zhang, S. Q., McDanel, B., and Kung, H. Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 846–860. IEEE, 2022b.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

## A. Data Calibration

Given a calibration dataset containing $N$ samples, $\{X_i \mid i = 1, 2, \ldots, N\}$, we first profile the activation magnitude for each channel,

$$\begin{aligned} \mathbf{a}_i &= \text{mean}(|X_i|, \text{axis} = 0), \\ \bar{\mathbf{a}} &= \max(\begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_N \end{bmatrix}, \text{axis} = 0), \end{aligned} \tag{13}$$

where $|\cdot|$ calculates the element-wise absolute value and $\bar{\mathbf{a}} = [a_1, a_2, \ldots, a_m]$ is a row vector of maximum channel magnitudes across samples. We normalize $\bar{\mathbf{a}}$ to get the diagonal matrix $S$:

$$s_i = \frac{a_i}{\sqrt{\min(\bar{\mathbf{a}}) \times \max(\bar{\mathbf{a}})}}, \tag{14}$$

Equation (13) and Equation (14) are empirical implementation based on (Lin et al., 2023). We leave the exploration of an analytical derivation of $S$ as future work.

## B. Comparison between LQER and L$^2$QER

Here we visualzie the approximation error of LQER and LQER versus layer index in Figure 4. The approximation error is measured as:

$$e_a = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (|E_q - \widetilde{E}_q|_{i,j}) \tag{15}$$

where $|\cdot|$ calculate the element-wise absolute value. L$^2$QER reconstructs the quantization error more accurate than LQER on most layers, while LQER better reconstruct the K, Q, and V projection layers at the 1st, 3rd, and 4th transformer layers.

## C. Inconsistant performance of OmniQuant on WikiText2 and downstream tasks

OmniQuant is one of the state-of-the-art LLM post-training-quantization methods we compared in this work. Thanks for the official open-sourced implementation and quantization parameter checkpoints, we performed extensive experiments to compare OmniQuant to LQER. We sucessfully reproduce the perplexity and downstream task accuracy of OPT-family. However, the LLaMA models quantized by OmniQuant have obvious performance degradation on downstream tasks, around 18.9% lower than FP16 baselines on average.

We attribute this performance degradation to the iterative gradient-base training on WikiText2 in OmniQuant. As stated in (Shao et al., 2023), OmniQuant optimizes the quantization parameter (shifts and scales) by training on

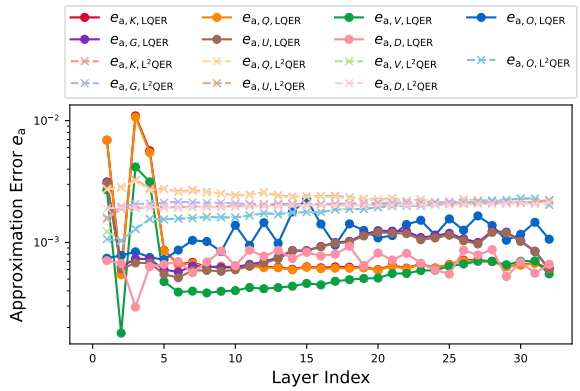



Figure 4: Approximation error of LQER and L$^2$QER across decoder layers in LLaMA-7B. L$^2$QER produces smaller approximation errors on most of the linear layers in transformer-based LLMs. However, there are a few layers better reconstructed by LQER, such as the key, value, output project layers in 1st, 3rd, and 4th decoder layer. The derivation of $S$ worths further exploration.

WikiText2 samples for 20 epochs (40 epochs for W2A16). This training requires tuning the hyper-parameters such as number of training samples, learning rates and total number of epochs, which may cause overfitting or underfitting if not tuned properly. Both cases can be the reason for performance degradation.

## D. Estimate Hardware Cost

We estimate the memory efficiency with average bitwidth. The average bitwidth of per-channel scaled quantization is considered as the average bits of an FP16 scalor and $m$ fixed-point numbers, where $m$ is the input hidden size. The average bitwidth of `MXINT` is averaged across one shared exponent and $B$ mantissas, where $B$ is the block size. For LQER/L$^2$QER, this is averaged across the low-precision $W_q$ and the high-precision $A_k$ and $B_k$. The average weight bitwidth of L$^2$QER in memory is 0.2 higher than GPTQ and AWQ, which is mainly contributed by the two low-rank matrices $A_k$ and $B_k$. L$^2$QER outperforms existing nearly lossless methods in terms of circuit area, because it is free from expensive element-wise dequantization (GPTQ and AWQ), or scatter/gather operations (`LLM.int4()`) at runtime.

We estimate the hardware cost with circuit area. We mapped the algorithms of these approaches onto custom hardware accelerators on FPGAs. To ensure fairness, these hardware accelerators have the same throughput of 16 multiply-accumulate (MAC) operations per clock cycle when computing a linear operation of the same matrix sizes. We then measure the circuit area in terms of LUTs and Digital Signal Processing blocks (DSPs) on the FPGA, where a DSP is treated as 100 LUTs. The area results were measured from the Place & Route report in Xilinx Vivado 2023.1. The FPGA family that we used for all the experiments is Xilinx Alveo U250. We summarize the area breakdown of `LLM.int4()`, AWQ, and L$^2$QER in Table 7, Table 8, and Table 9, respectively.

Table 7: Area breakdown of `LLM.int4()`, where GEMM$_l$ and GEMM$_h$ are low-precision and high-precision GEMM operations respectively.

| `LLM.int4()` | GEMM$_l$+ casting | Scatter + gather | GEMM$_h$ | Other |
|---|---|---|---|---|
| LUTs | 106959 | 11579 | 404 | 13604 |
| Percentage | 80.7 % | 8.8% | 3.0% | 10.3% |

Table 8: Area breakdown of AWQ

| AWQ | Dequantize | Matmul | Other |
|---|---|---|---|
| LUTs | 62907 | 11476 | 11131 |
| Percentage | 73.6% | 13.4% | 13.0% |

Table 9: Area breakdown of L$^2$QER, where Matmul1, Matmul2, and Matmul3 are $XW_q$, $XA_k$, and $(XA_k)B_k$ respectively.

| L$^2$QER | Matmul2 | Matmul1 | Matmul3 |
|---|---|---|---|
| LUTs | 1782 | 1028 | 992 |
| Percentage | 34.5% | 19.9% | 19.2% |

## E. More evaluation results

We present the complete results of each specific downstream tasks in Tables 11 to 18. We also tested L$^2$QER on Vicuna-7b/13b and Mistral-7b-v0.1 in Tables 19 to 21.

Table 10: More 2-bit $w$-only results. These methods perform inconsistently with model sizes and families. Unlike a simple rank $k = 32$ for W4A8 quantization, $L^2$QERrequires a larger rank $k$ for 2-bit quantization.

| Method | OPT | | | LLaMA | |
|---|---|---|---|---|---|
| | 125M | 1.3B | 2.7B | 7B | 13B |
| FP16 | 27.65 | 14.63 | 12.47 | 5.67 | 5.10 |
| OmniQuant | 75.43 | **23.95** | **18.13** | 12.97 | 10.36 |
| Quip | 347.40 | 41.64 | 2998.00 | 10.97 | 8.43 |
| $L^2$QER | **45.29** | 29.82 | 23.76 | **10.30** | **8.42** |

Table 11: OPT-6.7B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 10.86 | 65.6% | 30.5% | 67.7% | 76.3% | 66.1% | 27.6% | 55.6% |
| GPTQ | 10.95 | 65.6% | 31.1% | 68.5% | 76.2% | 65.2% | 26.2% | 55.4% |
| AWQ | 10.93 | 65.3% | 30.5% | 67.4% | 76.6% | 65.2% | 26.6% | 55.3% |
| LLM.int4() | 11.23 | 65.3% | 30.5% | 67.4% | 76.6% | 65.2% | 26.6% | 55.3% |
| OmniQuant (W6A6) | 10.96 | 65.4% | 30.9% | 66.9% | 76.0% | 66.2% | 26.8% | 55.4% |
| LQER-INT (W4A8) | 11.10 | 63.8% | 29.6% | 65.7% | 75.6% | 63.1% | 26.8% | 54.1% |
| LQER-MXINT (W4A6) | 11.03 | 65.4% | 30.5% | 65.6% | 75.4% | 64.0% | 27.6% | 54.7% |
| LQER-MXINT (W4A8) | 11.00 | 65.2% | 30.4% | 66.3% | 75.5% | 65.3% | 27.6% | 55.0% |

Table 12: OPT-13B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 10.13 | 67.1% | 32.9% | 68.6% | 76.0% | 65.8% | 27.0% | 56.2% |
| GPTQ | 10.31 | 67.5% | 32.8% | 68.8% | 76.1% | 65.9% | 27.2% | 56.4% |
| AWQ | 10.21 | 66.8% | 33.3% | 68.2% | 75.6% | 66.5% | 28.0% | 56.4% |
| LLM.int4() | 10.39 | 66.2% | 33.6% | 67.8% | 76.2% | 67.3% | 24.2% | 55.9% |
| OmniQuant (W6A6) | 10.96 | 67.1% | 33.1% | 68.4% | 76.2% | 65.3% | 26.4% | 56.1% |
| LQER-INT (W4A8) | 10.38 | 66.5% | 33.2% | 67.5% | 75.5% | 67.9% | 26.4% | 56.2% |
| LQER-MXINT (W4A6) | 10.32 | 67.2% | 32.2% | 67.9% | 75.7% | 68.3% | 25.8% | 56.2% |
| LQER-MXINT (W4A8) | 10.27 | 67.4% | 32.6% | 68.4% | 76.1% | 68.3% | 26.2% | 56.5% |

Table 13: OPT-6.7B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 9.56 | 70.0% | 34.6% | 71.5% | 77.6% | 70.5% | 30.2% | 59.1% |
| GPTQ | 9.63 | 62.2% | 29.4% | 74.9% | 67.6% | 69.1% | 23.8% | 54.5% |
| AWQ | 9.59 | 69.7% | 34.6% | 71.6% | 77.3% | 70.4% | 30.0% | 58.9% |
| LLM.int4() | 10.01 | 69.0% | 32.8% | 71.3% | 76.9% | 70.2% | 27.8% | 58.0% |
| OmniQuant (W6A6) | 9.62 | 70.1% | 34.2% | 70.4% | 77.3% | 70.2% | 29.6% | 58.6% |
| LQER-INT (W4A8) | 9.72 | | | | | | | |
| LQER-MXINT (W4A6) | 9.72 | 0.6990740741 | 0.3421501706 | 0.7050261983 | 0.7725788901 | 0.6923547401 | 0.298 | 58.5% |
| LQER-MXINT (W4A8) | 9.67 | 69.4% | 34.4% | 70.4% | 77.3% | 69.5% | 29.6% | 58.4% |

Table 14: llama-7B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 5.10 | 77.4% | 46.4% | 76.2% | 79.1% | 78.0% | 33.2% | 65.0% |
| GPTQ | 5.21 | 76.9% | 46.8% | 75.0% | 79.3% | 76.4% | 34.0% | 64.7% |
| AWQ | 5.20 | 77.2% | 46.4% | 75.6% | 79.0% | 77.8% | 32.8% | 64.8% |
| LLM.int4() | 5.31 | 77.2% | 46.0% | 75.4% | 78.9% | 77.1% | 32.8% | 64.6% |
| OmniQuant (W6A6) | 5.28 | 72.5% | 42.9% | 0.0% | 78.2% | 66.4% | 29.0% | 48.2% |
| LQER-INT (W4A8) | 5.31 | 76.9% | 45.9% | 74.0% | 78.7% | 77.2% | 33.6% | 64.4% |
| LQER-MXINT (W4A6) | 5.24 | 77.1% | 46.2% | 75.6% | 79.2% | 77.6% | 33.6% | 64.9% |
| LQER-MXINT (W4A8) | 5.21 | 77.0% | 46.3% | 75.6% | 79.6% | 77.3% | 33.2% | 64.8% |

Table 15: LLaMA-13B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 5.67 | 75.4% | 41.9% | 73.5% | 78.7% | 75.1% | 34.4% | 63.2% |
| GPTQ | 9.63 | 73.6% | 40.4% | 70.0% | 77.7% | 73.0% | 30.0% | 60.8% |
| AWQ | 9.59 | 75.5% | 41.1% | 72.5% | 78.6% | 74.9% | 32.2% | 62.5% |
| LLM.int4() | 10.01 | 74.6% | 42.1% | 70.3% | 78.6% | 74.8% | 32.8% | 62.2% |
| OmniQuant (W6A6) | 9.62 | 66.4% | 38.8% | 0.0% | 76.7% | 72.8% | 27.2% | 47.0% |
| LQER-INT (W4A8) | 6.09 | 73.9% | 40.6% | 73.4% | 77.7% | 74.0% | 30.6% | 61.7% |
| LQER-MXINT (W4A6) | 5.92 | 74.8% | 41.5% | 73.4% | 78.2% | 75.2% | 33.0% | 62.7% |
| LQER-MXINT (W4A8) | 5.89 | 74.9% | 41.6% | 73.3% | 78.6% | 76.1% | 33.6% | 63.0% |

Table 16: LLaMA-30B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 4.10 | 80.4% | 52.8% | 77.6% | 81.1% | 82.7% | 36.0% | 68.4% |
| GPTQ | 4.24 | 80.7% | 50.2% | 77.6% | 80.5% | 83.1% | 35.8% | 68.0% |
| AWQ | 4.22 | 74.1% | 46.0% | 0.0% | 79.5% | 68.3% | 31.4% | 49.9% |
| LLM.int4() | 4.33 | 79.0% | 48.9% | 75.8% | 80.2% | 82.4% | 33.6% | 66.7% |
| OmniQuant (W6A6) | 4.38 | 74.1% | 46.0% | 0.0% | 79.5% | 68.3% | 31.4% | 49.9% |
| LQER-INT (W4A8) | 4.35 | 80.1% | 49.7% | 77.0% | 80.7% | 81.5% | 35.2% | 67.4% |
| LQER-MXINT (W4A6) | 4.28 | 80.1% | 50.9% | 77.4% | 80.6% | 82.4% | 35.4% | 67.8% |
| LQER-MXINT (W4A8) | 4.25 | 80.0% | 50.8% | 77.6% | 80.7% | 82.5% | 36.2% | 68.0% |

Table 17: LLaMA-2-7B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 5.48 | 76.3% | 43.6% | 73.9% | 78.1% | 77.7% | 31.4% | 63.5% |
| GPTQ | 5.69 | 75.0% | 42.2% | 72.3% | 77.4% | 76.4% | 30.0% | 62.2% |
| AWQ | 5.61 | 75.2% | 43.3% | 72.7% | 77.6% | 77.3% | 31.4% | 62.9% |
| LLM.int4() | 5.77 | 75.1% | 42.7% | 71.9% | 77.6% | 76.2% | 32.2% | 62.6% |
| OmniQuant (W6A6) | 5.87 | 67.3% | 39.0% | 0.0% | 77.6% | 69.9% | 29.2% | 47.2% |
| LQER-INT (W4A8) | 5.85 | 74.7% | 42.4% | 71.6% | 76.7% | 76.1% | 32.0% | 62.2% |
| LQER-MXINT (W4A6) | 5.73 | 75.1% | 43.1% | 73.6% | 77.6% | 76.2% | 32.6% | 63.0% |
| LQER-MXINT (W4A8) | 5.69 | 75.3% | 42.5% | 73.7% | 77.9% | 76.3% | 32.8% | 63.1% |

Table 18: LLaMA-2-13B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 4.90 | 79.4% | 48.3% | 76.7% | 79.1% | 80.6% | 35.0% | 66.5% |
| GPTQ | 5.06 | 78.6% | 47.4% | 76.4% | 78.2% | 80.8% | 34.2% | 65.9% |
| AWQ | 4.98 | 78.9% | 46.9% | 76.2% | 78.8% | 80.1% | 34.4% | 65.9% |
| LLM.int4() | 4.98 | 77.6% | 47.0% | 76.1% | 78.9% | 80.5% | 34.8% | 65.8% |
| OmniQuant (W6A6) | 5.14 | 71.3% | 43.8% | 0.0% | 78.6% | 69.8% | 33.0% | 49.4% |
| LQER-INT (W4A8) | 5.10 | 78.5% | 47.1% | 75.8% | 78.6% | 81.0% | 34.4% | 65.9% |
| LQER-MXINT (W4A6) | 5.05 | 78.2% | 46.4% | 76.4% | 78.3% | 80.6% | 34.8% | 65.8% |
| LQER-MXINT (W4A8) | 5.02 | 78.3% | 47.0% | 76.4% | 78.8% | 81.3% | 34.6% | 66.1% |

Table 19: Vicuna-7B-v1.5

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 6.78 | 75.6% | 43.3% | 71.1% | 77.3% | 80.9% | 33.0% | 63.5% |
| GPTQ | 7.07 | 75.4% | 41.5% | 69.4% | 76.0% | 81.3% | 33.2% | 62.8% |
| AWQ | 7.00 | 75.0% | 41.8% | 70.0% | 77.1% | 81.5% | 32.2% | 62.9% |
| LLM.int4() | 7.14 | 75.0% | 42.6% | 69.3% | 76.3% | 81.3% | 34.2% | 63.1% |
| LQER-MXINT (W4A8) | 7.01 | 75.4% | 42.2% | 68.9% | 77.1% | 81.6% | 33.0% | 63.0% |

Table 20: Vicuna-13B-v1.5

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 5.92 | 78.7% | 47.8% | 73.4% | 78.9% | 85.2% | 36.8% | 66.8% |
| GPTQ | 6.00 | 77.9% | 46.4% | 72.9% | 78.1% | 85.0% | 36.8% | 66.2% |
| AWQ | 6.03 | 78.3% | 48.4% | 72.9% | 78.3% | 84.8% | 36.8% | 66.6% |
| LLM.int4() | 6.09 | 77.5% | 47.3% | 73.0% | 78.3% | 85.2% | 36.8% | 66.4% |
| LQER-MXINT (W4A8) | 6.04 | 78.5% | 46.7% | 72.7% | 77.7% | 85.0% | 36.4% | 66.2% |

Table 21: Mistral-7B

| Method | WikiText2 | ARC (easy) | ARC (challenge) | LAMBADA | PIQA | BOOLQ | OpenbookQA | Avg. Accuracy |
|---|---|---|---|---|---|---|---|---|
| FP16 | 6.47 | 82.7% | 53.5% | 70.7% | 80.4% | 86.2% | 32.8% | 67.7% |
| GPTQ | 8.13 | 81.1% | 55.8% | 72.2% | 80.9% | 86.7% | 36.0% | 68.8% |
| AWQ | 6.64 | 81.9% | 53.8% | 71.8% | 80.7% | 86.2% | 37.4% | 68.6% |
| LLM.int4() | 6.66 | 81.2% | 53.2% | 70.6% | 81.2% | 86.4% | 34.6% | 67.9% |
| LQER-MXINT (W4A8) | 6.71 | 81.7% | 53.8% | 71.2% | 81.0% | 86.5% | 34.8% | 68.2% |