Dual Precision Quantization for Efficient and Accurate Deep Neural Networks Inference

Tomer Gafni Asaf Karnieli Yair Hanani
Intel, Israel

{tomer.gafni,asaf.karnieli,yair.hanani}@intel.com

Abstract

Deep neural networks have achieved state-of-the-art results in a wide range of applications, from natural language processing and computer vision to speech recognition. However, as tasks become increasingly complex, model sizes continue to grow, posing challenges in latency and memory efficiency. To meet these constraints, post-training quantization has emerged as a promising solution. In this paper, we propose a novel hardware-efficient quantization and inference scheme that exploits hardware advantages with minimal accuracy degradation. Specifically, we introduce a W4A8 scheme, where weights are quantized and stored using 4-bit integer precision, and inference computations are performed using 8-bit floating-point arithmetic, demonstrating significant speedups and improved memory utilization compared to 16-bit operations, applicable on various modern accelerators. To mitigate accuracy loss, we develop a novel quantization algorithm, dubbed Dual Precision Quantization (DPQ), that leverages the unique structure of our scheme without introducing additional inference overhead. Experimental results demonstrate improved performance (i.e., increased throughput) while maintaining tolerable accuracy degradation relative to the full-precision model.

1. Introduction

Deep neural networks (DNNs) have demonstrated remarkable capabilities across a broad spectrum of tasks, having a significant impact in numerous fields. In recent years, model sizes have grown significantly to better capture complex data structures, accommodate longer context lengths, and enhance overall model strength. While these large-scale models exhibit outstanding functionality, their immense size poses significant challenges for deployment, including high computational costs and memory demands. To address these issues and enable efficient inference, quantization techniques are essential.

Quantization methods can be broadly categorized into two approaches: quantization-aware training (QAT) [1–4] and post-training quantization (PTQ) [5–10]. The former integrates quantization into the training process, typically requiring extensive retraining and/or fine-tuning while employing approximate differentiation techniques for the rounding operation. In contrast, PTQ quantizes a pre-trained model with significantly fewer resources, such as compute and data, and over a much shorter process. PTQ is particularly appealing for large-scale models, where full retraining or even fine-tuning is prohibitively expensive. PTQ has been successfully implemented for both language models [11–13] and vision models [14–17]. In this paper, we focus on PTQ as a practical solution for efficient deployment of massive models.

PTQ produces a smaller and faster model. Reducing the size of the model allows us to use less memory on hardware accelerators such as Nvidia GPUs and Intel HPUs and/or increase the maximum batch size. A smaller model also reduces the bandwidth required to move weights from memory, which is particularly important at low batch sizes and in auto-regressive decoding scenarios. In addition, if we quantize the activations, we can take advantage of hardware support for fast low-precision operations, where the matrix-multiplication computation can be performed 2x faster than 16-bit floating point arithmetic [18, 19]. Therefore, when considering PTQ, it is crucial to understand the different regimes of inference performance, which broadly fall into two categories: compute-bound inference, where performance is limited by computational throughput, and memory-bound inference, where the bottleneck in performance is the rate at which data is transferred from memory to processing cores. Memory-bound scenarios are especially typical at low batch size, where there are a small number of operations per byte of the loaded model. In memory-bound cases, performance gains are typically achieved by reducing memory traffic rather than increasing compute power, as hardware compute units are often underutilized while waiting for data to be fetched from memory. Taking

this into account, two primary approaches are commonly used to accelerate and optimize DNN inference, as detailed next.

FP8 quantization: Recent AI accelerators, including NVIDIA H100 & H200, AMD MI300X, and Intel Gaudi 2 & 3, supports the FP8 numerical format [18–21]. FP8 offers two key advantages over 16-bit floating-point (FP) formats: (i) cutting memory requirements in half, leading to better memory utilization and improved bandwidth efficiency; and (ii) reducing computational complexity, often doubling throughput compared to 16-bit FP operations [22–25].

INT4 quantization: For even greater compression, INT4 quantization is a leading approach that balances high compression with acceptable accuracy degradation. Storing weights in only 4 bits significantly reduces memory consumption, allowing larger models to fit on fewer machines and to increase the batch size, while improving inference speed. The low-precision representation reduces the latency associated with transferring weights from memory to processing cores, further accelerating model inference. Various works considered effective algorithms, including AWQ [13], GPTQ [12], QuaRot [26], SqueezeLLm [27], AdaRound [11], APQ-Vit [28], NoisyQuant [29] and more [8, 30–34]. Some algorithms related to this work uses information from second-order derivatives of the error function to perform network pruning and quantization [12, 35–37].

In this work, we propose a hybrid W4A8 scheme, that both stores weights in 4-bit precision (W4) accounting for the memory-bound regime, and performs the arithmetic in FP8 (A8), accounting for the compute-bound regime. Recently, [23] and [38] proposed a W4A8 quantization scheme similar to the one presented in this paper. However, their approach quantizes activations to 8-bit integers (INT8), whereas our method employs FP8 for both activations and computations.

FP8 can be advantageous over INT8 in terms of model accuracy, as INT8 represents a uniform distribution of values, making it most effective when model weights and activations also follow a uniform distribution. However, this is rarely the case, as empirical studies show that pretrained neural network weights typically exhibit a zero-centered, normal-like distribution [39]. In contrast, FP8, with its non-uniform distribution and higher dynamic range, better accommodates such data distributions while also providing higher precision for small values, and better captures outliers. This advantage becomes even more significant when quantizing activations as well, since their distributions tend to be less uniform and contain more outliers compared to weight distributions.

Our main contributions are as follows.
1) We propose a novel quantization and inference scheme that effectively accelerates inference in both the compute-bound and memory-bound regimes. Our proposed W4A8 scheme significantly reduces memory consumption, enabling larger batch sizes for faster evaluation and generation. Additionally, all matrix multiplications are performed in FP8, leveraging the capabilities of modern AI accelerators to enhance both speed and efficiency.
2) To preserve accuracy, we introduce a dedicated quantization algorithm tailored for this scheme: the Dual Precision Quantization (DPQ). DPQ is designed to minimize quantization errors, allowing us to maintain most of the accuracy of the original model on various tasks, without introducing any inference overhead.
3) Group-Aware Reordering (GAR): To further mitigate accuracy degradation, we introduce a weight quantization strategy that prioritizes important weights without inference overhead. Notably, GAR is not limited to our specific setting; it is applicable to other quantization schemes, making it a versatile technique for improving the accuracy of quantized models without introducing additional overhead. Ablation studies further confirm its effectiveness.

Finally, we conduct experiments on both language-vision and large language models across a range of tasks and sizes. Specifically, we evaluate our scheme on Llama-2 and Llama-3 models (7B to 70B) as well as the Qwen2-VL family (2B to 72B), comparing it against existing algorithms and alternative precision strategies. The results demonstrate significant throughput gains with minimal accuracy degradation compared to higher-precision schemes, while also achieving better accuracy than existing W4A8 schemes, validating the effectiveness of our approach.

2. Background

In this section, we introduce the fundamentals of DNN quantization, establish the relevant notation for our scheme, and discuss the key challenges in PTQ.

2.1. Integer Quantization

Integer quantization maps high-precision numbers to discrete levels. The process can be formulated as:

$$\begin{aligned} \boldsymbol{X}_q^{\text{INT}} &= \left\lceil \frac{\boldsymbol{X}}{s_I} + z_I \right\rfloor, \\ s_I = \frac{\boldsymbol{X}_{\max} - \boldsymbol{X}_{\min}}{q_{\max} - q_{\min}}, &\quad z_I = \left\lceil q_{\min} - \frac{\boldsymbol{X}_{\min}}{s_I} \right\rfloor, \end{aligned} \tag{1}$$

where $\lceil \cdot \rfloor$ denotes the rounding-to-nearest-integer operator, $\boldsymbol{X}$ represents the high-precision FP tensor, and $\boldsymbol{X}_q^{\text{INT}}$ is its n-bit quantized integer counterpart. The parameters s_I and z_I denote the scaling factor and zero-point, respectively, where the subscript I indicates the integer format. This is known as *asymmetric* quantization, where $\boldsymbol{X}_{\max} = \max(\boldsymbol{X})$, $\boldsymbol{X}_{\min} = \min(\boldsymbol{X})$, and $q_{\max} - q_{\min} = 2^n - 1$.

Eq. (1) can be further simplified to *symmetric* quantization, where $z_I = 0$, $\boldsymbol{X}_{\text{max}} = -\boldsymbol{X}_{\text{min}} = \max|\boldsymbol{X}|$, and $q_{\text{max}} - q_{\text{min}} = 2^n - 2$.

The dequantized tensor is represented as:

$$\hat{\boldsymbol{X}} = (\boldsymbol{X}_q^{\text{INT}} - z_I) \cdot s_I. \tag{2}$$

In this paper, weights are stored in 4-bit precision using asymmetric integer quantization.

2.2. FP8 Quantization

In this section, we describe the FP8 quantization scheme used in this paper, focusing on the E4M3 data type (4-bit exponent and 3-bit mantissa). This format is widely adopted for inference, as higher precision is generally more important than a large dynamic range, which is more relevant during training (particularly in the backward pass) where the E5M2 format is commonly used. Similarly to integer quantization, FP8 quantization applies a scaling factor to map a high-precision tensor into the FP8 dynamic range, followed by a rounding operation to the nearest representable value in the FP8 grid. We employ symmetric quantization. The process can be mathematically expressed as:

$$\boldsymbol{X}_q^{\text{FP}} = \text{nearest}_{\mathcal{Z}}(\frac{\boldsymbol{X}}{s_F}), s_F = \frac{\max(|\boldsymbol{X}|)}{Z_{\max}}. \tag{3}$$

Here, $\mathcal{Z}$ represents the E4M3 grid, and $Z_{\max}$ is its maximum value. The scaling factor s_F is used for FP quantization (with the subscript F indicating the float type). The tensor $\boldsymbol{X}_q^{\text{FP}}$ denotes the q-bit FP quantized representation (FP8 in this paper). The corresponding dequantized tensor is given by:

$$\hat{\boldsymbol{X}} = \boldsymbol{X}_q^{\text{FP}} \cdot s_{\text{F}}. \tag{4}$$

When implementing quantization algorithms, it is essential to consider the specific characteristics of the target hardware. For example, a key optimization in Intel Gaudi 2 and 3 accelerators is that when both input tensors (i.e., weights and activations) use scaling factors that are powers of two, scaling can be efficiently implemented by adjusting the exponent bias instead of multiplying individual elements [19]. This reduces computational overhead and improves FP8 throughput by several percentage points. To take advantage of this optimization, we incorporate the power-of-two scaling factors into our quantization scheme. For an in-depth analysis of using a power-of-two scaling factor, please refer to [19].

In this paper, integer quantization is applied only to weight tensors for storage, while FP quantization is used for both weight and activation tensors at computation time.

2.3. PTQ - Main Challenges

The quantization process in DNNs poses several fundamental challenges that must be carefully addressed when implementing PTQ algorithms. Below, we outline two key challenges relevant to our framework.

A primary issue is the presence of outliers, which force the scaling factor to accommodate an extended dynamic range. This results in the compression of most tensor values into a narrower range, increasing quantization step size and leading to higher reconstruction errors, which can significantly degrade model accuracy. To mitigate this, various techniques have been proposed, including clipping extreme values, per-channel or per-group quantization, and rotation methods that better capture the true distribution of values [26, 40]

Another key challenge in PTQ is that not all weights and activations contribute equally to the network's final output. As a result, quantization errors in highly influential components can have a disproportionate impact on overall model accuracy. To address this, several strategies have been explored, including reordering the weight matrix based on activation magnitudes to prioritize more critical elements [12], scaling weights and activations in a way that preserves the most important values while reducing errors in less significant ones [13], and restoring key scalar values that have an outsized influence [24, 41]. These algorithms help maintain accuracy in quantized models by focusing on the most significant elements, ensuring a more robust and efficient quantization process.

3. DPQ: Dual Precision Quantization for W4A8

We now introduce our Dual Precision Quantization (DPQ) algorithm. DPQ follows a structured flow: first, high-precision weights are quantized to INT4 offline. During inference, the 4-bit weights are loaded and dequantized to FP8. Simultaneously, activations are quantized from brain float (BF) 16 to FP8 to leverage efficient FP8 matrix multiplication, with the output in high-precision (BF16). To ensure clarity in the following formulation, we explicitly denote bit precision using subscripts to eliminate any ambiguity.

3.1. Compensating for Quantization Errors in W4A8 Flow

Note that our proposed target inference scheme involves two levels of weight quantization: 8-bit FP precision and 4-bit integer precision. Each quantization step introduces an error due to the rounding operator, which can degrade the accuracy of the model.

To mitigate this error, we take inspiration from the GPTQ algorithm [12] and build upon the Optimal Brain Quantization (OBQ) method [36] for layer-wise quantization. Specifically, our objective is to find a 4-bit weight matrix, such that its dequantized 16-bit matrix, $\hat{\boldsymbol{W}}_{16}$, minimizes the

squared error with respect to the full-precision BF16 layer output:

$$\text{argmin}_{\hat{\boldsymbol{W}}_{16}} ||\boldsymbol{W}_{16}\boldsymbol{X}_{16} - \hat{\boldsymbol{W}}_{16}\boldsymbol{X}_{16}||_2^2, \tag{5}$$

where $\hat{\boldsymbol{W}}_{16}$ incorporates the two-step dequantization process: from INT4 to FP8, and then from FP8 to BF16, and will be formally defined later in this section. This ensures that the error in Eq. (5) accounts for the cumulative error introduced by both quantization levels, as this is the error that will be encountered during inference.

As in GPTQ, we quantize one weight at a time for each row in $\boldsymbol{W}_{16}$. To mitigate the quantization error, we distribute the resulting error among subsequent weights in the row (i.e., from left to right). Each weight update is scaled by a factor determined by the Hessian matrix $\boldsymbol{H}$, which accounts for the importance of each weight.

We now provide a detailed description of the DPQ algorithm. First, we compute a scaling factor to convert the weights from BF16 to FP8, either per tensor or per channel. For clarity, we assume the scale is computed per tensor, denoted by $s_{w,16\to 8}$:

$$s_{w,16\to 8} = \frac{\max(|\boldsymbol{W}_{16}|)}{Z_{\max}}. \tag{6}$$

Next, consider the first weight in a given row of the weight matrix, denoted by w_{16}. The procedure is applied independently to all weights in the first column, but we describe it for a single row for clarity. We quantize w_{16} to FP8 using symmetric quantization:

$$w_8^{\text{FP}} = \text{nearest}_{\mathcal{Z}}(\frac{w_{16}}{s_{w,16\to 8}}). \tag{7}$$

We then compute the per-group scale and zero-point for converting FP8 to INT4. These values are determined, for a given group g (which is represented in FP8 precision), once at the beginning of the group quantization process:

$$\begin{aligned} s^g_{8\to 4} =& \frac{\max(\boldsymbol{W}^g_8) - \min(\boldsymbol{W}^g_8)}{q_{\max}}, \\ z^g_{8\to 4} =& \lceil -\frac{\min(\boldsymbol{W}^g_8)}{s^g_{8\to 4}} \rfloor, \end{aligned} \tag{8}$$

where $q_{\max} = 15$ for INT4 unsigned quantization as done in this paper. Using the computed scale and zero-point, we quantize the weight to INT4 using asymmetric integer quantization:

$$w_4^{INT} = \lceil \frac{w_8^{FP}}{s^g_{8\to 4}} + z^g_{8\to 4} \rfloor. \tag{9}$$

The weight w_4^{INT} is the weight which will be stored and used in inference. Following OBQ, we now update the remaining weights in the row (which were not quantized yet), to compensate for the quantization error. This process includes few steps. First, we dequantize the 4-bit weight back to FP8:

$$\hat{w}_8^{FP} = (w_4^{INT} - z^g_{8\to 4}) \cdot s^g_{8\to 4}. \tag{10}$$

Then, we dequantize $\hat{w}_8^{FP}$ to BF16:

$$\hat{w}_{16} = \hat{w}_8^{FP} \cdot s_{w,16\to 8}. \tag{11}$$

We then compute the quantization error and scale it by the inverse of the corresponding diagonal element of the Hessian, denoted by $[\boldsymbol{H}^{-1}]_{qq}$, where q is the index of the currently quantized weight.

Next, let F denote the set of remaining full-precision weights, i.e., the weights positioned to the right of the currently quantized weight. The optimal update for a weight $w_v \in F$, denoted by δ_F, is obtained by multiplying the scaled quantization error with the corresponding off-diagonal Hessian element $(\boldsymbol{H}_F^{-1})_{q,v}$, which captures the interaction between the quantized weight and the weight being updated: :

$$w_v \leftarrow w_v + \delta_F, \quad \delta_F = -\frac{w_{16} - \hat{w}_{16}}{[\boldsymbol{H}_F^{-1}]_{qq}} \cdot (\boldsymbol{H}_F^{-1})_{qv}. \tag{12}$$

The Hessian matrix characterizes the sensitivity of the weight to small perturbations, capturing how changes in the weight affect the loss function. The derivation of the optimal update in Eq. (12) can be found in [36].

This procedure is repeated iteratively for all remaining weights in the row. A pseudocode for the proposed DPQ algorithm is provided in Algorithm 1.

Recall that our scheme also includes activation quantization. Specifically, the activation input $\boldsymbol{X}_{16}$, which serves as the input to the linear layer, is quantized to FP8 using symmetric quantization:

$$\boldsymbol{X}_8 = \text{nearest}_{\mathcal{Z}}(\frac{\boldsymbol{X}_{16}}{s_{x,16\to 8}}), s_{x,16\to 8} = \frac{\max(|\boldsymbol{X}_{16}|)}{Z_{\max}}. \tag{13}$$

The scales $s_{x,16\to 8}$ are computed offline using a calibration dataset.

3.2. Inference-Aware Weight Reordering

A key challenge in the weight compensation approach is determining the optimal order for quantizing weights. The OBQ algorithm follows a greedy strategy, selecting the weight that introduces the least additional quantization error at each step. In contrast, GPTQ quantizes all rows in a predefined order. The original GPTQ paper proposes two schemes in the quantization process. The first preserves the original order of weights within the tensor, and the second reorders weights based on the Hessian diagonal, prioritizing the quantization of "more important" weights. While the second approach improves accuracy, it introduces inference overhead due to the need of memory indexing in

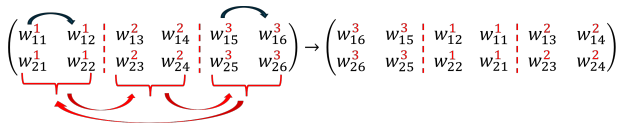

Figure 1. An illustration of the GAR method. In this example, the weight tensor is divided into three groups, indicated by the red superscript. The black arrows represent the local ordering within each group, based on the diagonal elements of the Hessian for that group, while the red arrows denote the global ordering between groups, determined by the maximum diagonal Hessian element of each group. At the end of the quantization process, the tensor is re-permuted to its original order. Notably, scales operate on consecutive weights after the ordering process (as opposed to activation reordering done in [12].)

weight grouping. Specifically, the weights $\mathbf{W}_8^g$ are consecutive in the permuted tensor, and thus the scales $s^g_{8\to4}$ and zero-points $z^g_{8\to4}$ operate on consecutive elements in the permuted tensor. However, the tensor saved and used in inference is the tensor in the original order, not the permuted tensor, and dequantizing online the weights using their precomputed scales and zero-points can not be done in one multiplication as in the original case, thus increasing the inference latency (more details are given in Sec. 6).

To address this, we propose Group-Aware Reordering (GAR), a reordering scheme that leverages Hessian information to prioritize important weights during quantization while avoiding inference overhead. GAR reorders weights under the constraint that weights can be permuted in one of two ways - within a group, and/or by moving whole groups. With GAR, scales operate on consecutive weights in the original order. This ensures that group-wise scales and weight groups remain consistent throughout inference. Although GAR can also be beneficial for other methods, such as W4A16 GPTQ, it is especially helpful for DPQ. Without GAR, due to the nature of the predefined quantization order, alongside double quantization, large errors may accumulate in the weights which are quantized last. GAR reduces this error by quantizing the hard-to-quantize weights first, which then causes a reduced accumulated error to the latter weights.

In Sec. 4 we demonstrate that our approach achieves higher accuracy compared to quantization without reordering and performs only slightly worse than unrestricted reordering while maintaining efficient inference execution. An illustration of the GAR method is shown in Fig. 1, and a detailed explanation of the scheme is provided in the supplementary material (see Sec. 6). Fig. 2 provides a high-level illustration of the proposed DPQ algorithm and Fig. 3 provides an illustration of the inference flow under our W4A8 scheme.

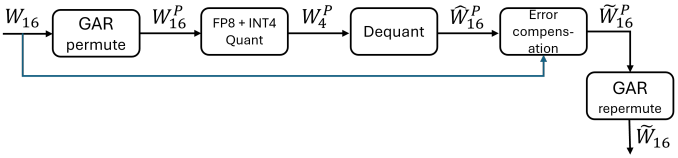

Figure 2. High-level illustration of the proposed DPQ algorithm. We begin by hybrid reordering (GAR) of the weights based on their importance. Next, we apply two quantization levels: FP8 and INT4. The quantization errors from both processes are computed and distributed to the next, less 'important' weights. Finally, we re-permute the tensor and the scales back to their original order.

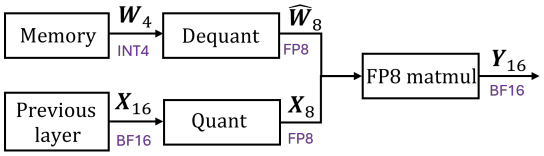

Figure 3. An illustration of the inference flow of our proposed W4A8 scheme.

3.3. Further Optimizations and Considerations

To further improve our quantization algorithm, we integrate additional techniques to enhance accuracy and efficiency. First, the per-group scaling factor $s^g_{8\to4}$ is determined by searching for the scale that minimizes the mean squared error (MSE). Additionally, we apply clipping to constrain values within the dynamic range of the low-precision tensor. As done in [12], we apply dampening to the Hessian matrix and employ the Cholesky decomposition for more efficient computation of the Hessian matrix in Eq. (12).

4. Experiments

In this section, we evaluate our proposed quantization and inference scheme in terms of both task accuracy and model performance (i.e., throughput). All experiments were conducted on Gaudi 2 and 3 devices, with BF16 precision used as the baseline for performance comparison. The quantization experiments were implemented using Intel's Neural Compressor framework[1], an open-source toolkit for neural network compression. We begin by evaluating visual reasoning on the MMMU [42], MMBench [43], and Math Vista [44] benchmarks using the Qwen2-VL model family [45], testing the 2B, 7B, and 72B models. The tasks were evaluated using VLMEvalkit [46] on Intel Gaudi 2. Next, since the language component is a crucial aspect of large vision-language models, and since previous W4A8 methods have focused on it, we also evaluate our method on the Llama-2 [47] and Llama-3 [48] model families. We evaluate the accuracy of our language models across three dis-

[1]https://github.com/intel/neural-compressor

Algorithm 1: Dual Precision Quantization (DPQ)

Input: Weight matrix $\boldsymbol{W}_{16}$ in BF16, Hessian matrix $\boldsymbol{H}$

Output: Quantized weight matrix $\mathbf{W}_4$, scaling factors $s_{16\to8}, s^g_{8\to4}$, zero-point $z^g_{8\to4}$

Compute scaling factor $s_{w,16\to8}$ (Eq. (6)) ;
foreach *row* $\boldsymbol{w} \in \boldsymbol{W}_{16}$ **do**
  foreach *group* $\mathcal{G}$ *in* $\boldsymbol{w}$ **do**
    Compute per-group scale $s^g_{8\to4}$ and zero-point $z^g_{8\to4}$ (Eq. (8)) ;
    foreach *weight* $w_{16} \in \mathcal{G}$ **do**
      $w_8 = \text{quant}(w_{16})$ (Eq. (7)) ;
      $w_4 = \text{quant}(w_8)$ (Eq. (9)) ;
      $\hat{w}_8 = \text{dequant}(w_4)$ (Eq. (10)) ;
      $\hat{w}_{16} = \text{dequant}(\hat{w}_8)$ (Eq. (11)) ;
      Compute the update δ_F (Eq. (12)) ;
      Distribute δ_F to remaining weights (Eq. (12))
    end
  end
end

tinct categories: WikiText-2 [49] for the measurement of perplexity, a comprehensive common sense reasoning suite, and MMLU [50] for the assessment of broad knowledge. The common sense reasoning evaluation is comprised of the following: HellaSwag (HS) [51], LAMBADA [52], BoolQ [53], ARC Easy (ARC-e) [54], PIQA (PQ) [55], Winogrande (WG) [56], ARC Challenge (Arc-c) [54], and OpenBookQA [57]. Results are reported as the average normalized accuracy for all tasks. The language tasks were evaluated using the LM-Evaluation-Harness framework [58] in a zero-shot setting to ensure consistent comparison between different experiments. Our code is publicly available.

4.1. Implementation Details

The implementation of E4M3 in Gaudi 2 follows the IEEE standard for floating points, which reserves the largest exponent for NaN (not-a-number) and infinite values, and has the range of ±240. In Gaudi 3 the maximal exponent can be used for normal numbers, increasing the numerical range of E4M3 to ±448.

We used static (off-line) activation quantization, where the scaling factors are determined through calibration on the WebQuestions (WebQS) dataset [59], while the calibration set for the error compensation procedure in the weight quantization is a subset of [60]. In all our experiments, we use a group size of 128 for the INT4 asymmetric quantization, and a maximum sequence length of 2048. The FP8 scales are computed per tensor with symmetric quantization.

In Llama-2 and Llama-3 models, we quantize all linear layers to INT4, except for the embedding layer and the lm-head, as is common in other works. In the Qwen-VL model, we quantize all linear layer weights of the language model. Quantization times are similar to the timings reported in [12].

4.2. Accuracy Results

We use the WxAy notation to denote a quantization scheme in which weights are stored with x-bit precision and activations with y-bit precision. Table 1 presents the results for the multi-modal dataset evaluated on the Qwen2-VL model family. The proposed DPQ algorithm demonstrates consistent performance across different model sizes and tasks, with only a minor degradation compared to the full BF16 reference. Notably, for most tasks, it outperforms the W4A16 GPTQ algorithm despite operating at a lower precision. This improvement can be attributed to the GAR method, which enhances model accuracy.

We next present a comparison of the proposed DPQ algorithm with state-of-the-art methods, including QServe [23], QQQ [38], GPTQ [12], and the vanilla round-to-nearest (RTN) method on language tasks. The results for QServe and QQQ are taken directly from their papers, where we used the 128-group size configuration as in DPQ for a fair comparison. The GPTQ method is implemented without the FP8 error compensation used in DPQ (i.e., only the INT4 error is compensated). Table 2 demonstrates that DPQ outperforms both RTN and the naive implementation of GPTQ, as expected. Additionally, DPQ achieves better results than QServe and QQQ, which follow a different approach (both perform quantization to INT8 rather than FP8). We also note that many optimizations in the QServe algorithm, such as block input/output module rotation and smoothing, are orthogonal to our method and could be integrated into DPQ to further increase its accuracy.

Table 3 presents the results for the WikiText-2, reasoning, and common sense tasks under various precisions. As expected, schemes that quantize weights to 4 bits exhibit greater accuracy degradation compared to the W8A8 algorithm, due to the significantly lower bit representation. However, our proposed W4A8 DPQ algorithm achieves a reasonable accuracy degradation compared to the full precision model, and only a small accuracy degradation compared to the W4A16 scheme (in Tab. 3 we implemented GPTQ with GAR, thus it serves as a natural upper bound for DPQ) while maintaining a more efficient trade-off between accuracy and latency (detailed in Sec. 4.3).

4.3. Performance Analysis

Using W4A8 improves model performance in several ways. With respect to full FP8 quantization (weights and activations - W8A8) - using 4 bit weights allows to further reduce the memory footprint, thus allowing for larger models

Model	Configuration	MMMU		MMBench		MathVista	
		Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑
Qwen2-VL-2B-Instruct	BF16 Reference	41.88	–	72.07	–	44.40	–
	W4A16 (GPTQ)	39.22	-6.78	70.87	-1.69	41.69	-6.50
	W4A8 (DPQ)	39.37	-6.37	71.12	-1.33	45.3	+1.98
Qwen2-VL-7B-Instruct	BF16 Reference	53.77	–	81.78	–	58.20	–
	W4A16 (GPTQ)	52.55	-2.32	81.27	-0.63	60.30	+3.48
	W4A8 (DPQ)	52.64	-2.14	81.25	-0.65	59.86	-2.77
Qwen2-VL-72B-Instruct	BF16 Reference	65.44	–	86.94	–	70.19	–
	W4A16 (GPTQ)	64.00	-2.25	86.68	-0.30	69.20	-1.43
	W4A8 (DPQ)	63.98	-2.28	86.48	-0.53	68.97	-1.77

Table 1. Accuracy of vision tasks under various precisions. BF16 and GPTQ results are taken from the Huggingface model pages.

Algorithm	AVG		ARC-e		ARC-c		HS		WG		PQ	
	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑
BF16	68.64	–	73.86	–	44.88	–	76.13	–	69.29	–	79.05	–
DPQ (ours)	**68.15**	**-0.73**	72.81	-1.45	**45.13**	**+0.55**	**75.17**	**-1.28**	**69.45**	**+0.22**	78.18	-1.11
QServe	67.95	-1.02	**73.32**	**-0.74**	44.80	-0.18	74.98	-1.53	68.59	-1.03	78.07	-1.26
QQQ	67.47	-1.74	72.94	-1.27	44.37	-1.15	74.53	-2.15	67.01	-3.41	**78.51**	**-0.69**
GPTQ	64.69	-6.11	67.88	-8.82	42.15	-6.48	71.64	-6.27	65.27	-6.17	76.55	-3.27
RTN	63.54	-8.03	66.87	-10.46	40.87	-9.87	71.31	-6.76	63.53	-9.08	75.13	-5.22

Table 2. Accuracy under various quantization algorithms for W4A8 scheme on Llama2-7B. The GPTQ results are an adaptation of GPTQ to W4A8, without accounting the FP8 quantization error. The results for QServe and QQQ were taken from [23] and [38], respectively.

to fit into smaller or less devices (when using model parallel), and improves the memory bandwidth. For example, using 4-bit weights allows models such as Llama 3.2 90B Vision to fit into a device such as Intel Gaudi 2 (96 Gb HBM) or Nvidia H100 (80Gb HBM). With respect to 4-bit weight-only quantization (W4A16), using FP8 precision for the activations enables larger batch sizes (which yields faster overall tokens per second), and for faster GeMM operations, if the hardware has support for it (Gaudi 2, Gaudi 3, Nvidia H100, H200 have support for such).

To showcase the potential gains for the W4A8 format, we provide LLM model projections in Fig. 4. We analyze performance for various input and output lengths. Since many recent VLMs use both a Vision Transformer and an LLM, we focus on LLM performance, since the majority of compute and runtime is due to the auto-regressive LLM, rather than the ViT which runs once and is a fraction of the model. We compare both 4 bit weights (W4A16) and FP8 quantization (W8A8), since both are popular quantization choices. Note that for the chosen models, a W16A16 (native BF16) model does not fit on a single device, thus not shown. The speed-up is achieved by larger batch sizes (which are enabled by smaller model weights), as well as more efficient compute, due to the FP8 GeMM engines being used.

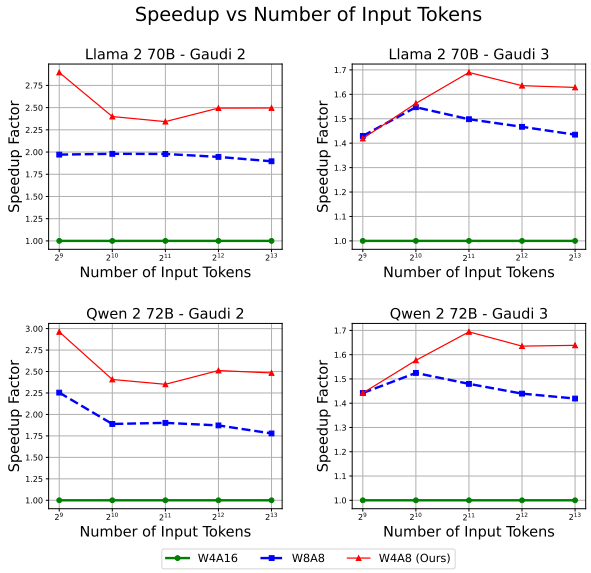

Figure 4. Speed-up comparison between 3 methods on Gaudi 2 and Gaudi 3 accelerators. Each subplot compares speed-up of W4A8 (red) and W8A8 (blue) compared to W4A16 (green). The two upper plots compare Llama 2 70B on Gaudi 2 (left) and Gaudi 3 (right). The bottom plots compare Qwen 2 72B on Gaudi 2 (left) and Gaudi 3 (right). As can be seen, W4A8 can reach up to 3x speed-up over W4A16, and up to 1.4x speed-up over W8A8. W4A8 uses GAR, thus weights are consecutive in memory during inference.

4.4. Ablation Study

We conduct an ablation study to evaluate the impact of our proposed activation hybrid reordering GAR method on model accuracy. As discussed earlier, we expect this hybrid scheme to outperform the scheme without reordering. Although our method imposes constraints on the activation order, we note that most layers are characterized by a small number of highly important output activations [24, 41]. Consequently, even though our scheme only partially or-

Model	Configuration	WikiText2		Commonsense		MMLU	
		Ppl ↓	Δ (%) ↓	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑
Llama-2-7B	BF16 Reference	5.472	–	65.851	–	43.074	–
	W8A8	5.502	+0.55	65.679	-0.26	43.006	-0.16
	W4A16* (GPTQ)	5.629	+2.79	65.049	-1.22	42.891	-0.48
	W4A8* (DPQ)	5.662	+3.36	65.582	-0.41	42.628	-1.04
	W4A16 (GPTQ)	5.745	+4.75	65.324	-0.81	41.662	-3.28
	W4A8 (DPQ)	5.781	+5.34	64.269	-2.40	41.931	-2.65
Llama-3.1-8B	BF16 Reference	6.238	–	70.809	–	66.626	–
	W8A8	6.311	+1.15	70.202	-0.86	66.021	-0.91
	W4A16* (GPTQ)	6.597	+5.45	69.657	-1.63	65.223	-2.11
	W4A8*(DPQ)	6.935	+10.04	69.588	-1.72	65.206	-2.13
	W4A16 (GPTQ)	6.631	+5.93	69.635	-2.04	64.073	-3.83
	W4A8 (DPQ)	6.704	+6.95	69.074	-2.45	63.987	-3.96
Llama-2-70B	BF16 Reference	3.320	–	73.561	–	67.691	–
	W4A16 (GPTQ)	3.420	+2.92	73.159	-0.55	66.828	-1.28
	W4A8 (DPQ)	3.466	+4.21	73.129	-0.59	66.346	-1.99

Table 3. Accuracy of language tasks under various precisions. The notations $W4A16^*$ and $W4A8^*$ refer to schemes that use full activation reordering, which reduces accuracy degradation at the cost of increased latency. GPTQ (W4A16) is included in the results, since it is a common quantization algorithm. Also, since DPQ adds an additional level of quantization over GPTQ, it serves as an upper-bound to DPQ.

ders activations, it effectively captures the most significant weights. Thus, we expect only a slight accuracy degradation compared to the full reordering scheme.

Our results, presented in Table 4, support this hypothesis. Under both W4A16 and W4A8 quantization schemes, GAR yields a notable improvement over the scheme without reordering, while exhibiting only a minor accuracy degradation compared to full reordering scheme. Note that GAR requires a ranking criterion to determine which group should be quantized first. In our experiments, we ordered groups based on their maximum Hessian diagonal element. Alternative criteria are possible, for example, ranking groups by the average of the top 10% largest elements before sorting. We leave the exploration of other ranking strategies for future work.

Another key insight into our algorithm emerges from the improved accuracy of DPQ compared to the naive implementation of GPTQ, as shown in Tab. 2. DPQ mitigates nearly 4% of the accuracy degradation relative to the full-precision reference. This is because our method compensates for the error using the dequantized weight from both quantization levels, allowing it to account for both sources of error and produce a more accurate layer output. In contrast, a naive implementation of GPTQ in the W4A8 scheme introduces the FP8 quantization error on top of the compensated INT4 quantization error, resulting in the degradation observed in Tab. 2.

4.5. Discussion

We next discuss some key points regarding our work.

1. The W4A16 scheme (when implemented with GAR as in Tab. 3), which only quantizes weights and performs computation in BF16, serves as a natural upper bound for our W4A8 approach . That is, we primarily compare our accuracy degradation against W4A16. Optimizing W4A16 (weight-only) is beyond the scope of this paper.
2. Our quantization method is orthogonal to many existing quantization strategies, such as AWQ, QuaRot, and others. Incorporating these techniques may further enhance our results. We leave this for future work.

Prec.	Config.	PPL - WikiText2		Commonsense		MMLU	
		Acc ↓	Δ (%) ↓	Acc ↑	Δ (%) ↑	Acc ↑	Δ (%) ↑
W4A16	BF16	6.238	–	70.809	–	66.626	–
	Ordered	6.597	+5.45	69.657	-1.63	65.223	-2.11
	GAR (ours)	6.600	+5.48	70.319	-0.69	65.114	-2.27
	Unordered	7.680	+18.78	69.285	-2.15	63.499	-4.69
W4A8	BF16	6.238	–	70.809	–	66.626	–
	Ordered	6.679	+6.59	69.239	-2.22	64.368	-3.39
	GAR (ours)	6.704	+6.95	69.074	-2.45	63.987	-3.96
	Unordered	32.812	+80.99	69.026	-2.52	61.716	-7.37

Table 4. Accuracy under different activation ordering methods for Llama-3.1-8B. 'Ordered' refers to full activation reordering, 'Unordered' means preserving the tensor in its original form. The scales for all methods are selected to minimize the mean squared quantization error between the dequantized and original tensors.

5. Conclusion

In this paper, we introduce DPQ, a dual precision aware W4A8 quantization algorithm designed for efficient inference on modern accelerators. Our method leverages a two-step quantization process, mapping high-precision weights to INT4 and activations to FP8, enabling efficient FP8 matrix multiplication while minimizing accuracy degradation. We further enhance our approach with error compensation technique and Hessian-guided weight reordering, ensuring a good balance between performance and accuracy. Experimental analysis on Qwen2-VL, Llama-2, and Llama-3 demonstrate that our W4A8 quantization method achieves significant throughput improvements while maintaining competitive accuracy relative to reference schemes.

References

[1] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019. 1

[2] B. Zhuang, M. Tan, J. Liu, L. Liu, I. Reid, and C. Shen, "Effective training of convolutional neural networks with low-bitwidth weights and activations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6140–6152, 2021.

[3] J. H. Lee, J. Yun, S. J. Hwang, and E. Yang, "Cluster-promoting quantization with bit-drop for minimizing network quantization loss," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5370–5379.

[4] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra, "Llm-qat: Data-free quantization aware training for large language models," *arXiv preprint arXiv:2305.17888*, 2023. 1

[5] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1325–1334. 1

[6] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," *arXiv preprint arXiv:2006.10518*, 2020.

[7] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D.Soudry, "Accurate post training quantization with small calibration sets," in *International conference on machine learning*. PMLR, 2021, pp. 4466–4475.

[8] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021. 2

[9] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, "Loss aware post-training quantization," *Machine Learning*, vol. 110, no. 11, pp. 3245–3262, 2021.

[10] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 168–27 183, 2022. 1

[11] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? adaptive rounding for post-training quantization," in *International conference on machine learning*. PMLR, 2020, pp. 7197–7206. 1, 2

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022. 2, 3, 5, 6, 1

[13] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024. 1, 2, 3

[14] C. Wang, Z. Wang, X. Xu, Y. Tang, J. Zhou, and J. Lu, "Q-vlm: Post-training quantization for large vision-language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 114 553–114 573, 2025. 1

[15] J. Xie, Y. Zhang, M. Lin, L. Cao, and R. Ji, "Advancing multimodal large language models with quantization-aware scale learning for efficient adaptation," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 10 582–10 591.

[16] Y. Ding, W. Feng, C. Chen, J. Guo, and X. Liu, "Reg-ptq: Regression-specialized post-training quantization for fully quantized object detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 174–16 184.

[17] C. Lv, H. Chen, J. Guo, Y. Ding, and X. Liu, "Ptq4sam: Post-training quantization for segment anything," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 941–15 951. 1

[18] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022. 1, 2

[19] J. Lee, S. Markovich-Golan, D. Ohayon, Y. Hanani, G. Park, B. Kim, A. Karnieli, U. Livne, H. Shen, T. Huang *et al.*, "Faster inference of llms using fp8 on the intel gaudi," *arXiv preprint arXiv:2503.09975*, 2025. 1, 3

[20] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort, "Fp8 quantization: The power of the exponent," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 651–14 662, 2022.

[21] J. Kim, J. Lee, G. Park, B. Kim, S. J. Kwon, D. Lee, and Y. Lee, "An investigation of fp8 across accelerators for llm inference," *arXiv preprint arXiv:2502.01070*, 2025. 2

[22] M. Van Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga *et al.*, "Fp8 versus int8 for efficient deep learning inference," *arXiv preprint arXiv:2303.17951*, 2023. 2

[23] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *arXiv preprint arXiv:2405.04532*, 2024. 2, 6, 7

[24] M. Yu, D. Wang, Q. Shan, and A. Wan, "The super weight in large language models," *arXiv preprint arXiv:2411.07191*, 2024. 3, 7, 2

[25] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099. 2

[26] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," *arXiv preprint arXiv:2404.00456*, 2024. 2, 3

[27] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "Squeezellm: Dense-

and-sparse quantization," *arXiv preprint arXiv:2306.07629*, 2023. 2

[28] Y. Ding, H. Qin, Q. Yan, Z. Chai, J. Liu, X. Wei, and X. Liu, "Towards accurate post-training quantization for vision transformer," in *Proceedings of the 30th ACM international conference on multimedia*, 2022, pp. 5380–5388. 2

[29] Y. Liu, H. Yang, Z. Dong, K. Keutzer, L. Du, and S. Zhang, "Noisyquant: Noisy bias-enhanced post-training activation quantization for vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 20 321–20 330. 2

[30] W. Cheng, W. Zhang, H. Shen, Y. Cai, X. He, K. Lv, and Y. Liu, "Optimize weight rounding via signed gradient descent for the quantization of llms," *arXiv preprint arXiv:2309.05516*, 2023. 2

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

[32] H. Badri and A. Shaji, "Half-quadratic quantization of large machine learning models," *Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob*, 2023.

[33] J. Liu, L. Niu, Z. Yuan, D. Yang, X. Wang, and L.Wenyu, "Pd-quant: Post-training quantization based on prediction difference metric," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24 427–24 437.

[34] J. Liu, L. Niu, Z. Yuan, D. Yang, X. Wang, and W. Liu, "Pd-quant: Post-training quantization based on prediction difference metric," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24 427–24 437. 2

[35] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299. 2

[36] E. Frantar and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 4475–4488, 2022. 3, 4

[37] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *European conference on computer vision*. Springer, 2022, pp. 191–207. 2

[38] Y. Zhang, P. Zhang, M. Huang, J. Xiang, Y. Wang, C. Wang, Y. Zhang, L. Yu, C. Liu, and W. Lin, "Qqq: Quality quattuorbit quantization for large language models," *arXiv preprint arXiv:2406.09904*, 2024. 2, 6, 7

[39] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023. 2

[40] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "Spinquant: Llm quantization with learned rotations," *arXiv preprint arXiv:2405.16406*, 2024. 3

[41] M. Sun, X. Chen, J. Z. Kolter, and Z. Liu, "Massive activations in large language models," *arXiv preprint arXiv:2402.17762*, 2024. 3, 7, 2

[42] X. Yue, Y. Ni, K. Zhang, T. Zheng, R. Liu, G. Zhang, S. Stevens, D. Jiang, W. Ren, Y. Sun *et al.*, "Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 9556–9567. 5

[43] Y. Liu, H. Duan, Y. Zhang, B. Li, S. Zhang, W. Zhao, Y. Yuan, J. Wang, C. He, Z. Liu *et al.*, "Mmbench: Is your multi-modal model an all-around player?" in *European conference on computer vision*. Springer, 2024, pp. 216–233. 5

[44] P. Lu, H. Bansal, T. Xia, J. Liu, C. Li, H. Hajishirzi, H. Cheng, K.-W. Chang, M. Galley, and J. Gao, "Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts," *arXiv preprint arXiv:2310.02255*, 2023. 5

[45] P. Wang, S. Bai, S. Tan, S. Wang, Z. Fan, J. Bai, K. Chen, X. Liu, J. Wang, W. Ge *et al.*, "Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution," *arXiv preprint arXiv:2409.12191*, 2024. 5

[46] H. Duan, J. Yang, Y. Qiao, X. Fang, L. Chen, Y. Liu, X. Dong, Y. Zang, P. Zhang, J. Wang *et al.*, "Vlmevalkit: An open-source toolkit for evaluating large multi-modality models," in *Proceedings of the 32nd ACM international conference on multimedia*, 2024, pp. 11 198–11 201. 5

[47] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023. 5

[48] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024. 5

[49] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016. 6

[50] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020. 6

[51] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019. 6

[52] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. 6

[53] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," *arXiv preprint arXiv:1905.10044*, 2019. 6

[54] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018. 6

[55] Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi, "Reasoning about physical commonsense in natural language," 2019. 6

[56] K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi, "An adversarial winograd schema challenge at scale," *arXiv preprint arXiv:1907.10641*, 2019. 6

[57] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," *arXiv preprint arXiv:1809.02789*, 2018. 6

[58] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 12 2023. [Online]. Available: https://zenodo.org/records/10256836 6

[59] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on Freebase from question-answer pairs," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. [Online]. Available: https://aclanthology.org/D13-1160 6

[60] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," 2020. [Online]. Available: https://arxiv.org/abs/2101.00027 6