MobileQuant: Mobile-friendly Quantization for On-device Language Models

Anonymous ACL submission

Abstract

Large language models (LLMs) have revolutionized language processing, delivering outstanding results across multiple applications. However, deploying LLMs on edge devices poses several challenges with respect to memory, energy, and compute costs, limiting their widespread use in devices such as mobile phones. A promising solution is to reduce the number of bits used to represent weights and activations. While existing works have found partial success at quantizing LLMs to lower bitwidths, *e.g.* 4-bit weights, quantizing activations beyond 16 bits often leads to large computational overheads due to poor on-device quantization support, or a considerable accuracy drop. Yet, 8-bit activations are very attractive for on-device deployment as they would enable LLMs to fully exploit mobile-friendly hardware, e.g. Neural Processing Units (NPUs). In this work, we make a first attempt to facilitate the on-device deployment of LLMs using integer-only quantization. We first investigate the limitations of existing quantization methods for on-device deployment, with a special focus on activation quantization. We then address these limitations by introducing a simple post-training quantization method, named MobileQuant, that extends previous weight equivalent transformation works by jointly optimizing the weight transformation and activation range parameters in an end-to-end manner. MobileQuant demonstrates superior capabilities over existing methods by 1) achieving near-lossless quantization on a wide range of LLM benchmarks, 2) reducing latency and energy consumption by 20%-50% compared to current on-device quantization strategies, 3) requiring limited compute budget, 4) being compatible with mobile-friendly compute units, *e.g.* NPU.

1 Introduction

Large language models (LLMs) have markedly advanced language processing capabilities, paving the way for expansive applications in artificial intelligence. However, the deployment of LLMs is costly in terms of memory, computation, and energy, which can be prohibitive on edge devices like mobile phones. A standard approach to facilitate running these models on edge devices is to quantize them, representing weights and activations with fewer bits, thereby mitigating these costs.

Existing LLM quantization works can be grouped into two categories: *weight-only quantization* and *weight-activation quantization*. Weight-only quantization approaches (Frantar et al., 2023; Lin et al., 2024) convert model weights into low-bitwidth integers, most commonly 4-bit, and maintain the activations in 16-bit floating-point. Weight-only quantization often preserves accuracy while significantly reducing the model storage footprint. In addition, weight-only quantization can result in minor gains in inference latency due to the reduction in memory access overheads. However, these approaches still suffer from high energy consumption and high latency, as computation is performed in floating point. Costly on-the-fly weight dequantization is also required during inference. Instead, weight-activation quantization approaches forgo the need for on-the-fly dequantization by quantizing both weights and activations, and potentially utilizing efficient fixed-point operators. Despite its efficiency benefits, quantizing activations typically degrades accuracy, especially in the case where static per-tensor quantization parameters are applied. To counteract this accuracy drop, previous works include quantizing activations for certain expensive operations (Xiao et al., 2023), *e.g.* matrix multiplication, or employing dynamic per-token quantization (Shao et al., 2024; Liu et al., 2023; Ashkboos et al., 2024; Liu et al., 2024), which is often slow on Graphic Processing Units (GPUs) and, most importantly, lacks hardware support on edge devices. Notably, none of these methods support lossless 8-bit (int8) per-tensor quantization

for the activations, or fully leverage low-precision fixed-point engines, such as the Digital Signal Processor (DSP), or dedicated Neural Processing Unit (NPU) (Qualcomm, 2024; Google, 2021), commonly found in mobile devices (Mahurin, 2023). Towards on-device quantization for LLMs, we introduce MobileQuant, a post-training quantization approach that not only effectively handles the conventional accuracy and efficiency challenges of quantization but is also seamlessly supported by existing mobile hardware. To achieve this, Mobile-Quant consists of three simple yet effective methodological extensions, motivated by the shortcomings of existing state-of-the-art works when deployed on device, and building on top of these works. These extensions include: *1)* applying weight equivalent transformation on *all possible layers*, *2)*, learning the optimal quantization range for activations, *3)* jointly optimizing all weight transformation and range parameters in an end-to-end manner. As such, MobileQuant applies a combination of per-tensor and per-channel weight quantization at 4-bit or 8-bit and per-tensor activation quantization at 8-bit or 16-bit, utilizing fixed-point integer representations for all operations.

The benefits of MobileQuant over previous works are multifold. Firstly, MobileQuant enables the quantization of the weights to either 4-bit or 8-bit and the activations to 8-bit integers, except for non-linearities like softmax and normalization, with minimal impact on performance. Mobile-Quant, hence, maximizes the potential of equivalent transformation-based methods (Nagel et al., 2019; Xiao et al., 2023; Lin et al., 2024; Shao et al., 2024) that achieve linear-invariant weight equalization. Deploying LLMs on device using MobileQuant results in a significant reduction in inference speed and energy usage as the latency and energy consumption of multiply-accumulate operations correlate directly with the bit-widths. Besides substantial gains during inference, we also show that MobileQuant's end-to-end optimization benefits from more calibration samples and extended training samples through our ablation study. In contrast, previous works that adopt closed-form solutions (Nagel et al., 2019), search-based optimization (Lin et al., 2024), and block-wise error minimization (Shao et al., 2024; Liu et al., 2024) struggle to scale with the number of samples and training steps. Lastly, in comparison with other learnable-based quantization methods such as Quantization Aware Training (QAT) (Liu et al.,

2023; Bondarenko et al., 2023), MobileQuant retains the model generalizability as the model remains mathematically equivalent to its unquantized variant. Our contributions are summarized as follows:

1. We introduce a post-training quantization approach for large language models (LLMs) that is supported by current mobile hardware implementations (i.e. DSP, NPU), thus being directly deployable on real edge devices.

2. Our method improves upon prior works through simple yet effective methodological extensions that enable us to effectively quantize most activations to a lower bitwidth (*i.e.* 8-bit) with near-lossless performance.

3. We conduct a comprehensive on-device evaluation of model accuracy, inference latency, and energy consumption. Our results indicate that our method reduces both inference latency and energy usage by 20%-50% while still maintaining accuracy compared to models using 16-bit activations.

2 Related Work

2.1 Post-training Quantization (PTQ)

Previous research in post-training quantization for LLMs can be categorized into three main groups: **Weight-only Quantization** focuses on compressing the model weights to reduce storage requirements and memory transfer overheads. Representative works (Frantar et al., 2023; Lin et al., 2024; Shao et al., 2024; Liu et al., 2024) generally achieve performance comparable to full-precision models and maintain similar inference speeds on GPUs. However, these methods dequantize weights to 16-bit values on the fly, resulting in high-precision floating-point computations and hence leading to high inference latency and energy consumption, particularly on edge devices such as mobile phones. **Weight-activation Quantization** extends quantization to both model weights and activations, aiming to further reduce computational overhead. However, as indicated in prior works (Dettmers et al., 2022; Xiao et al., 2023), activations have dynamic ranges across different data distributions and are hence more challenging to quantize compared to weights. As a result, quantizing activations to a lower bit-width often results in a significant performance decline. Leading solutions either retain

2

some compute-intensive matrix multiplications in full precision (Dettmers et al., 2022; Xiao et al., 2023) or utilize dynamic per-token activation quantization, which lacks hardware support on mobile platforms. In contrast, our approach quantizes all linear operations and is compatible with current hardware support on edge devices.

Learning to Round. Notable works like (Nagel et al., 2020; Lee et al., 2023) also focus on weight-only quantization but introduce techniques for learning optimal weight rounding. The key argument is that the conventional round-to-nearest method is suboptimal, as it does not account for the interdependencies among adjacent weights. Our work is orthogonal with this research and can hence be integrated with these techniques.

2.2 Quantization Aware Training (QAT)

Quantization aware training (QAT) involves retraining or fine-tuning full-precision models using differentiable quantizers. Recent research (Liu et al., 2023; Bondarenko et al., 2023) has shown that QAT outperforms PTQ methods, particularly with in-domain training data. However, QAT requires extensive training, which is often impractical for LLMs. Additionally, QAT may be vulnerable to domain shifts if the data used for pretraining is unavailable. In contrast, our approach is zero-shot, only requiring a minimal set of calibration samples and a limited compute budget. Once trained, our model remains mathematically equivalent to the original model when unquantized, enhancing its adaptability to various downstream tasks.

3 Preliminaries

3.1 Mobile-friendly Design Choices

Quantization methods are differentiated by several main design choices, with varying levels of hardware support. In this section, we first list these design choices and then highlight the limitations of existing works with respect to these choices.

Support for mobile-friendly bitwidth: int8-int8 operations are widely supported and most often optimized for, while int4-int16 and int8-int16 are typically supported although often slower than int8-int8.

Quantization groups: Quantizing using per-tensor and per-channel statistics is widely supported while using per-token statistics is not.

Dynamic vs static: Static quantization statistics that do not depend on the input data, typically computed on a holdout calibration set, are widely supported. Dynamic quantization, on the other hand, requires online calibration from the input data and is not widely supported.

State-of-the-art quantization methods demonstrate strong performance on a server use case (i.e. high-end GPU). However, they either utilize on-the-fly dequantization and 16-bit floating point operations (Frantar et al., 2023; Lin et al., 2024), which are computationally inefficient, or dynamic per-token quantization (Xiao et al., 2023; Shao et al., 2024), which, as previously mentioned, has no support on edge devices.

We, instead, consider design choices that are widely supported and optimized on modern edge devices (*e.g.* Mobile NPUs), namely *i)* fixed-point weight and activation quantization with integer arithmetic operations, and *ii)* per-tensor/channel quantization with static pre-computed ranges. Our objective is hence to improve existing state-of-the-art approaches such as SmoothQuant (Xiao et al., 2023) and OmniQuant (Shao et al., 2024) while staying within the limits of hardware support on device.

3.2 Weight Equivalent Transformation

Prior efforts on LLM quantization (Dettmers et al., 2022; Xiao et al., 2023) observed that activations are harder to quantize compared to the model weights due to the outlier channel dimensions with diverse min-max ranges. As an example, given a fully connected layer $\mathbf{Y} = \mathbf{XW}$, $\mathbf{W} \in \mathbb{R}^{N \times M}$, $\mathbf{X} \in \mathbb{R}^N$, $\mathbf{Y} \in \mathbb{R}^M$, specific channel dimensions $\{i : 0 \leq i < N\}$ in $\mathbf{X}$ may have a wide min-max range across different data samples, causing substantial quantization errors. To counteract this, previous methods proposed a weight equivalent transform defined by a scaling vector $\mathbf{S} \in \mathbb{R}^N$:

$$\mathbf{Y} = \mathbf{XW} = (\mathbf{XS}^{-1}) \cdot (\mathbf{SW}) = \hat{\mathbf{X}}\hat{\mathbf{W}} \qquad (1)$$

The goal is to find the optimal scaling vector $\mathbf{S}$ such that both $\hat{\mathbf{X}}$ and $\hat{\mathbf{W}}$ are easier to quantize compared to the original $\mathbf{X}$ and $\mathbf{W}$. SmoothQuant (Xiao et al., 2023) reparameterized $\mathbf{S}$ as $\mathbf{s}_i = \frac{max(|\mathbf{X}_i|)^\alpha}{max(|\mathbf{W}_i|)^{(1-\alpha)}}$, $0 \leq i < N$, and searched for the hyper-parameter α. The obtained $\mathbf{S}$ is similar to the closed-form solution derived in (Nagel et al., 2019). OmniQuant (Shao et al., 2024) extended SmoothQuant (Xiao et al., 2023) by learning $\mathbf{S}$, together with the weight clipping parameters via
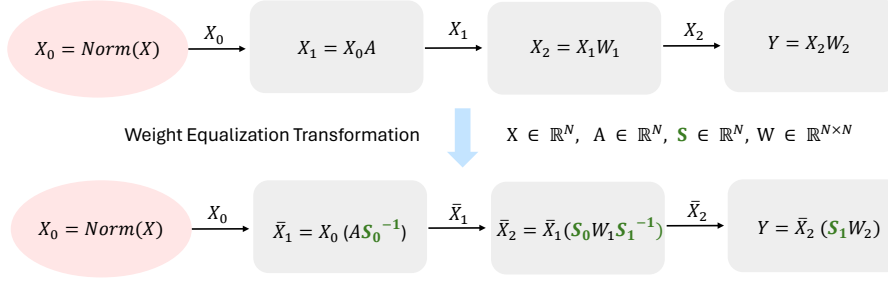
3

Figure 1: Weight equalization transformation proposed in (Nagel et al., 2019; Xiao et al., 2023; Shao et al., 2024). In this example, we use three consecutive layers: one normalization layer, *e.g.* LayerNorm (Ba et al., 2016)/RMSNorm (Zhang and Sennrich, 2019), and two linear layers, and assume the activations of all layers have the same hidden dimension N. Here $A \in \mathbb{R}^N$ refers to the affinity transformation of the normalization layer. The goal of weight transformation is to learn the scaling vector S such that the resulting weight matrices (i.e. $S_0 W_1 S_1^{-1}$ and $S_1 W_2$) and activations $\bar{X}$, are easier to quantize. S is hence the only learnable parameters. Note that the new model is mathematically equivalent to the original model when unquantized.

block-wise error minimization. Here, both $\mathbf{S}$ and $\mathbf{S}^{-1}$ can be fused to the adjacent linear layers, making the transformation mathematically equivalent to the original models. Figure 1 provides an illustration of the transformation among consecutive linear layers.

4 MobileQuant: Towards Mobile-friendly Quantization

4.1 Challenges for Mobile-friendly Quantization

The weight equivalent transformation approaches used in SmoothQuant and OmniQuant, as described in Section 3.2, demonstrate strong performance on GPU-like hardware. However, they do not work out of the box for edge devices. Specifically, two challenges remain: i) the weight transformations cannot propagate beyond non-linear operators, *e.g.* Softmax, RMSNorm (Zhang and Sennrich, 2019), LayerNorm (Ba et al., 2016), SiLU/GELU (Hendrycks and Gimpel, 2016). To counteract this, we apply weight transformations on all consecutive layers with linear components, *e.g.* between linear layers or affine transformations in the normalization layers, while keeping the non-linear activations in 16-bit integers; ii) with the weight transformation, the distribution of the activations shifts accordingly. This causes essential difficulty for learning-based approaches like Omni-Quant (Shao et al., 2024), when the min-max range for the activations changes after each training iteration. OmniQuant (Shao et al., 2024) proposed to bypass the issue with dynamic per-token quantization, which has no hardware support on-device.

4.2 Learning the Per-tensor Range of the Activations

Given the distribution of the activations shifts accordingly with the weight transformation, the ideal solution is to re-estimate the activation ranges across the training set after each training iteration. However, doing so is computationally prohibited. Hence, we propose to learn the activation range jointly with the weight transformation. Given an activation tensor $\mathbf{X}$, instead of learning the min and max values $f_{min}(\mathbf{X})$, $f_{max}(\mathbf{X})$ directly, we leverage the correlation between f_{min}, f_{max} and the scale and offset parameters, $\alpha, \beta \in \mathbb{R}$, for quantization. With the targeted bit-width bw, quantizing $\mathbf{X}$ can be formulated as:

$$q_{max} = 2^{bw} - 1, \alpha = \frac{f_{max} - f_{min}}{q_{max}}, \beta = \frac{f_{min}}{\alpha} \quad (2)$$

$$\mathbf{X}_{int} = min(max(ste(\frac{\mathbf{X}}{\alpha}) - \beta, 0), q_{max}) \quad (3)$$

Here, $\mathbf{X}_{int}$ refers to the quantized tensor of $\mathbf{X}$, ste refers to straight-through estimator. We can therefore learn $f_{min} = \alpha\beta$ and $f_{max} = \alpha q_{max} + \alpha\beta$ indirectly by learning α and β, which are computationally more stable.

4.3 End-to-end Optimization vs Layer-wise Optimization

To learn the equivalent transformation, previous works either resort to closed-form solutions (Nagel et al., 2019), search-based methods (Xiao et al., 2023; Lin et al., 2024), or layer-wise error minimization (Shao et al., 2024). These solutions require limited training budget, but, as shown in Sec-

4

WikiText ($\downarrow$)	TinyLLaMA 1.1B	StableLM-2 1.6B	Gemma 2B
FP16	14.88	28.41	17.96
W8A8			
SmoothQuant-Static	160.52	565.55	9242.60
SmoothQuant-Edge	22.29	70.21	36.03
OmniQuant-Static	49.09	210.24	7.48E+08
OmniQuant-Edge	16.00	30.72	20.77
W4A8			
OmniQuant-Static	98.43	160.02	1.04E+03
OmniQuant-Edge	18.29	37.00	24.69

Table 1: **Adapting quantization SOTA to the on-device setting**. OmniQuant and SmoothQuant are not fully supported for on-device deployment. We introduce mobile-friendly variants. Evaluation: perplexity on WikiText (Merity et al., 2016). We adopt the "Edge" variants as strong on-device baselines.

tion. 5.4, lead to sub-optimal performance. Particularly, given the restricted form of supervision, we show that these methods cannot scale with more training samples or iterations. We, instead, propose to jointly optimize all the training parameters, including the weight equalization parameters $\mathbf{S}$, weight clipping parameters used in OmniQuant (Shao et al., 2024), and the range parameters α, β for all layers in an end-to-end manner. Compared to previous PTQ approaches, which struggle with more training samples and epochs, we demonstrate that our holistic optimization approach consistently improves the performance with larger training settings for different LLM architectures. Compared to QAT, our method preserves model generalizability and does not overfit to specific calibration samples, achieving near-lossless zero-shot performance.

5 Experiments

5.1 Setup

We perform experiments by training and simulating the quantization on GPUs and further evaluate the on-device performance on a Samsung Galaxy S24, with the Snapdragon 8 Gen 3 HTP as the compute unit. All models were trained on two A100 GPUs, with a maximum sequence length of 2048.

Architectures: MobileQuant focuses on lightweight LLMs that are suitable to be deployed on mobile devices. Hence, we experiment with representative pretrained models with different architectures: TinyLlaMA-1.1B-Chat-v1.0 (Zhang et al., 2024), StableLM-2-1.6B (Bellagente et al., 2024), and Gemma-2B (Google, 2024).

Quantization details. MobileQuant use a subset of the Pile (Gao et al., 2020) dataset as the calibration set. We explore two quantization settings: i) W8A8: 8-bit weight quantization with per-tensor statistics except for the last linear projection in each MLP block (e.g. $down\_proj$ in LLaMA-like (Touvron et al., 2023) models) which uses per-channel statistics, and 8-bit per-tensor quantization for the activations, except those linked to non-linear operators. ii) W4A8: 4-bit per-channel quantization for model weights, and 8-bit per-tensor quantization, likewise excluding non-linear operators.

We consider asymmetric quantization for both settings, which can utilize the full quantized range. We also provide extra experiments on symmetric per-channel W4A8 quantization in the supplemental material, which is better supported by the current on-device toolchain we use.

Evaluation datasets. We evaluate our quantization approach in a zero-shot setting on representative tasks from the Language Model Evaluation Harness benchmark (Harness) (Gao et al., 2023) including WikiText (Merity et al., 2016), AI2 Reasoning Challenge (arc_challenge) (Clark et al., 2018), Hellaswag (Zellers et al., 2019), and MMLU (Hendrycks et al., 2021).

5.2 On-device Baselines

In this section, we extend state-of-the-art weight-activation quantization methods, SmoothQuant (Xiao et al., 2023) and OmniQuant (Shao et al., 2024) on device and use them as baselines. As these approaches utilize dynamic per-token quantization for the activation, which is not supported on edge devices, we modify these methods to work on device by using static per-tensor activation quantization, referring to these variants as OmniQuant-Static and SmoothQuant-Static respectively. Note that, for SmoothQuant, we only include evaluations on W8A8, which is the default setting used in the original work.

As shown in Table 1, both "Static" variants suffer from large performance degradation when evaluated on WikiText (Merity et al., 2016). We further observe that the performance drop is mainly caused by quantizing the activations for the last linear layer in each MLP head (*i.e.* $down\_proj$ in LLaMA-like (Touvron et al., 2023) models). To further alleviate this issue, we introduce an extra weight equalization transformation between

5

#Samples	#Epochs	TinyLlaMA-1.1B		StableLM-2-1.6B		Gemma-2B	
		Block-wise	End-to-end	Block-wise	End-to-end	Block-wise	End-to-end
128	20	18.1	19.1	<u>35.2</u>	38.9	<u>23.6</u>	30.5
128	60	<u>17.9</u>	17.2	36.6	34.7	24.2	25.2
128	120	18.2	17.0	36.9	34.0	24.0	24.0
256	60	18.0	16.9	<u>35.2</u>	33.9	24.9	23.6
1024	60	<u>17.9</u>	**16.7**	35.4	**32.9**	23.9	**22.7**

Table 2: **End-to-end range optimization:** Perplexity on WikiText for OmniQuant-Edge W4A8 setting with block-wise vs end-to-end range optimization. Best overall performance is in **bold**, best block-wise performance is <u>underlined</u>. Compared to block-wise, end-to-end optimization benefits from larger training settings with more samples/iterations, leading to better performance.

consecutive linear layers in each MLP head (*i.e.* **S** between the *up_proj* and *down_proj* layers in TinyLLaMA (Zhang et al., 2024)). The new models, which we termed *SmoothQuant-Edge* and *OmniQuant-Edge* respectively, significantly alleviate the performance degradation. For the remainder of this section, we use these adapted models as strong on-device baselines.

WikiText (↓)	TinyLLaMA 1.1B	StableLM-2 1.6B	Gemma 2B
FP16	14.88	28.41	17.96
W8A8			
OmniQuant-Edge	16.00	30.72	20.77
OmniQuant-Edge w ARL	15.77	29.75	20.05
W4A8			
OmniQuant-Edge	18.29	37.00	24.69
OmniQuant-Edge w ARL	18.06	35.15	23.56

Table 3: **Activation range learning** (ARL): Perplexity on WikiText for OmniQuant-Edge with/without ARL for W8A8 and W4A8 settings. The performance gains are larger on models with larger quantization errors.

5.3 Impact of Activation Range Learning

Table 1 shows that the learning-based approach, OmniQuant (Shao et al., 2024), outperforms the search-based method, SmoothQuant (Xiao et al., 2023), for all models by a notable margin. However, learning to transform the weights with fixed activation ranges is suboptimal, as the activation ranges shift after each training iteration. We further evaluate the impact of incorporating activation range learning (ARL), described in Section. 4.2, into OmniQuant (Shao et al., 2024). In other words, we learn the per-tensor scale and offset parameters, together with the weight transformation via block-wise error minimization.

Table 3 demonstrates that activation range learning (ARL) consistently improves the performance for all LLM models across all settings. The gains are larger for quantized models exhibiting a larger performance gap compared to the FP16 models. Notably, these models require more training steps to mitigate the quantization errors, leading to larger range shifts for the activation.

5.4 Impact of End-to-End Optimization

In the previous section, we show that incorporating ARL into our baselines results in consistent improvements. Nonetheless, there is still a notable performance gap between the quantized models and the FP16 models, especially under the W4A8 setting. In order to reduce this gap, we attempt to improve the performance by scaling up the performance, namely increasing the number of calibration samples and the number of training epochs. However, Table 2 shows that the performance of all considered models saturate as we scale the training up using the block-wise approach proposed in OmniQuant (Shao et al., 2024). We therefore conjecture that the optimization is hindered by the block-wise error minimization objective that provides limited global supervision. To verify this, we use our end-to-end training pipeline and jointly optimize all trainable parameters of the whole model, namely the weight transformation, clipping, and activation range learning parameters.

As shown in Table 2, our end-to-end trained models demonstrate consistent improvements with more training samples and iterations, only underperforming the blockwise optimized models in the smallest setting of 128 samples and 20 training epochs when the models were undertrained. We currently train the models with up to 1024 samples for 60 epochs but posit that the models could be further improved with more diverse samples and larger training settings.

5.5 Harness Benchmark Results

Following previous approaches (Xiao et al., 2023; Shao et al., 2024; Liu et al., 2023), we perform zero-shot evaluations on representative tasks from

6

		WikiText ↓	ARC-Challenge ↑	HellaSwag ↑	MMLU ↑
W8A8					
TinyLlaMA-1.1B	FP16	14.9	33	60	25
	SmoothQuant-Edge	22.3	29.7	53.6	24.2
	OmniQuant-Edge	<u>16.0</u>	<u>32.5</u>	<u>58.9</u>	<u>25.0</u>
	MobileQuant	**15.5** (-0.5)	**32.6** (+0.1)	**59.3** (+0.4)	**25.5** (+0.5)
StableLM-2-1.6B	FP16	28.4	39	65	32
	SmoothQuant-Edge	70.2	34.5	56.3	25.8
	OmniQuant-Edge	<u>30.7</u>	<u>36.4</u>	<u>63.2</u>	<u>26.4</u>
	MobileQuant	**29.7** (-1.0)	**37.5** (+1.1)	**63.8** (+0.6)	**28.7** (+2.3)
Gemma-2B	FP16	18.0	23	42	28
	SmoothQuant-Edge	36.0	**23.2**	39.1	24.3
	OmniQuant-Edge	<u>20.8</u>	<u>21.6</u>	<u>38.8</u>	<u>25.7</u>
	MobileQuant	**19.2** (-1.6)	21.4 (-1.8)	**40.9** (+1.8)	**26.8** (+1.1)
W4A8					
TinyLlaMA-1.1B	FP16	14.9	33	60	25
	OmniQuant-Edge	18.3	**31.9**	<u>56.6</u>	<u>25.2</u>
	MobileQuant	**16.7** (-1.6)	<u>30.7</u> (-1.2)	**57.4** (+0.8)	**25.7** (+0.5)
StableLM-2-1.6B	FP16	28.4	39	65	32
	OmniQuant-Edge	37.0	36.7	60.3	24.9
	MobileQuant	**32.9** (-4.1)	**37.6** (+0.9)	**60.9** (+0.6)	**27.2** (+2.3)
Gemma-2B	FP16	18.0	23	42	28
	OmniQuant-Edge	<u>24.7</u>	<u>21.1</u>	**38.4**	**26.0**
	MobileQuant	**22.7** (-2.0)	**23.8** (+2.7)	<u>37.7</u> (-0.7)	<u>24.4</u> (-1.6)

Table 4: **Comparisons with existing state-of-the-art methods on Harness:** Best performance is **bold**, second-best underlined. We indicate the gain/drop of our approach vs the next strongest on-device baseline. Our method, MobileQuant, demonstrates consistent improvements across models, quantization configurations, and tasks, achieving best performance in most cases.

TinyLlaMA-1.1B	WikiText ↓	Lambada ↑
FP16	14.9	82.9
W8A16	15.2	82.9
MobileQuant *W8A8*	15.6	82.4
full W8A8	8e5	1.3

Table 5: **On-device accuracy** of the quantized TinyLLaMA-1.1B-Chat-v1.0 on WikiText and LAMBADA. Models run on a Snapdragon 8 Gen 3 HTP processor.

the Harness benchmark (Gao et al., 2023). Table 4 shows that, in addition to the WikiText perplexity, our method also improves the quantization performance for the common sense reasoning tasks in general, without using any in-domain data. The improvements are consistent for most benchmarks and we believe that the performance of our method could be further improved with in-domain data, especially for benchmarks with a large domain shift relative to our calibration set (i.e. Pile (Gao et al., 2020)).

5.6 On-device Evaluation

On-device Setup. We further deploy the quantized LLM model on a mobile device and provide evaluations on the accuracy, latency, memory usage, and power consumption. Specifically, we evaluate the W8A8 quantized TinyLLaMA-1.1B-Chat-v1.0 (Zhang et al., 2024) model on a Samsung Galaxy S24, using the Snapdragon 8 Gen 3 HTP as the compute unit. We evaluate the model under three different quantization settings: 1) *W8A16*, which keeps activations as 16-bit; Note that the matrix multiplication for the self-attention computation is still between 8-bit and 16-bit unsigned integer activations to avoid potential overflowing, 2) *full W8A8*, keeps all activations in 8-bit, and 4) our proposed MobileQuant for W8A8.

On-device Accuracy. We compute the accuracy of the quantized models on two tasks: i) WikiText (Merity et al., 2016) from Harness (Gao et al., 2023), as we used in our previous evaluations and ii) LAMBADA (Paperno et al., 2016), which predicts the last token of a sentence given the previous context. Following SmoothQuant (Xiao et al., 2023), we use the first 1000 samples from LAMBADA for this task. Table 5 shows that using 16-bit activations (i.e. *W8A16*) achieves lossless performance. However, quantizing all activations into 8-bit leads to near-zero performance, highlighting

7

Seq. Length	Method	Avg. lat. (ms)	Avg. energy (mJ)	Peak mem. (MiB)
Prompt Encoding				
256	*W8A16*	510	1000	1019
	MobileQuant (*W8A8*)	276	490	1011
	full W8A8	89	183	1006
Autoregressive Generation				
1024	*W8A16*	54	69	1007
	MobileQuant (*W8A8*)	46	61	1005
	full W8A8	42	61	1003
2048	*W8A16*	119	165	1010
	MobileQuant (*W8A8*)	95	110	1007
	full W8A8	94	106	1006

Table 6: **On-device execution cost.** Measurements of latency, energy and memory are computed under sustained execution (30 minutes). Values are reported per single forward pass.

the difficulty of activation quantization. Our W8A8 MobileQuant model achieves near-lossless performance in both tasks, approaching the performance of the FP16 model.

On-device Latency. We provide the on-device latency evaluation by running the quantized model in two modes: i) prompt encoding with a context length of 256, ii) auto-regressive generation with a maximum sequence length of 1024 and 2048. Table 6 shows that, for prompt encoding, using lower-bitwidth activations is critical to reducing the inference latency, as some of the operations, e.g. self-attention (batched matrix multiplication), are compute-intensive. Our model demonstrates significant advantages over the full W8A16 solution, reducing the latency by 40%. However, there is still a large gap between MobileQuant and the *full W8A8* model, indicating the improvement margin. For auto-regressive generation, the latency gaps are smaller. We posit that the auto-regressive generation is not as compute-bound as prompt encoding, especially for lightweight models, but instead is partially memory access-bound. Our solution demonstrates a 20% latency reduction compared to *W8A16*, achieving the same latency as the *full W8A8* model. We include a video demo that showcases the auto-regressive generation of the quantized model on device in the supplemental material. In general, the advantage of low-bitwidth activations correlates strongly with the scale of the computation. Hence, we aim to extend the latency evaluation to larger models in our future research.

On-device Energy and Memory. Apart from latency, energy consumption is another important aspect of on-device execution, which is often overlooked by quantization research. To measure the energy requirements of different models, we run them

on a number of identical mobile phones as used before continuously for 30 minutes. The phones are connected to the testing host machine via WiFi using an internal network without access to the internet, to avoid any undesired network activity. The phones are also not being charged and their screens are turned off. All phones begin each test at the same battery level and the final energy of running a model is calculated as the ratio of the total battery discharged over the duration of a test, minus reference discharge of a phone not running any model, divided by the number of times the model was run. We repeat measurements for different models 3 times, rotating the phones each time, and report the average. We also report peak memory required to run a model as the peak resident memory recorded for the benchmarking process by the Linux Kernel (the so-called Virtual Memory High Water Mark). From Table 6, the energy consumption of each model aligns well with the latency. Compared to *W8A16*, MobileQuant reduces 50% of the power usage for prompt encoding and 35% for autoregressive generation. The peak memory usage for all models are similar as it is dominated by the model weight.

6 Conclusion

We revisited LLM quantization from the perspective of deployment on edge devices such as mobile phones. We examined the limitations of current state-of-the-art models for on-device deployment and present MobileQuant, the first framework to facilitate compute-, and energy-efficient quantized LLMs with minimal performance loss. Mobile-Quant is drop-in compatible with today's edge device hardware and low-level runtimes.

Limitations

The work explores reducing the overhead of on-device deployment for Large Language Models by hardware-friendly quantization. Our current study focuses on established pretrained LLMs with 1 to 2 billion parameters, which limits the overall capacity of the quantized models. Also, the quantized models inherit the error of the pretrained models, e.g. hallucination, which may be corrected by extra guard models (Inan et al., 2023). For now, we demonstrate the efficiency and effectiveness of MobileQuant on specific high-end mobile phones. We plan to extend our research to more LLMs with different architectures, model sizes, capacities, as well as more edge devices in the future.

References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. QuaRot: Outlier-free 4-bit inference in rotated llms. *Preprint*, arXiv:2404.00456.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Marco Bellagente, Jonathan Tow, Dakota Mahan, Duy Phung, Maksym Zhuravinskyi, Reshinth Adithyan, James Baicoianu, Ben Brooks, Nathan Cooper, Ashish Datta, et al. 2024. Stable LM 2 1.6 b technical report. *Preprint*, arXiv:2402.17834.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2023. Quantizable transformers: Removing outliers by helping attention heads do nothing. In *Advances on Neural Information Processing Systems*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances on Neural Information Processing Systems*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate post-training compression for generative pretrained transformers. In *International Conference on Learning Representations*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB dataset of diverse text for language modeling. *Preprint*, arXiv:2101.00027.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Google. 2021. Edge tpu. https://cloud.google.com/edge-tpu.

Google. 2024. Gemma: Open models based on Gemini research and technology. *Preprint*, arXiv:2403.08295.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *International Conference on Learning Representations*.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging non-linearities and stochastic regularizers with gaussian error linear units. *Preprint*, arXiv:1606.08415.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. 2023. Llama guard: LLM-based input-output safeguard for human-AI conversations. *Preprint*, arXiv:2312.06674.

Jung Hyun Lee, Jeonghoon Kim, Se Jung Kwon, and Dongsoo Lee. 2023. FlexRound: Learnable rounding based on element-wise division for post-training quantization. In *International Conference on Machine Learning*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for llm compression and acceleration. In *Conference on Machine Learning and Systems*.

Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. 2024. Qllm: Accurate and efficient low-bitwidth quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. LLM-QAT: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2307.06281*.

E. Mahurin. 2023. Qualocmm® hexagon™ NPU. In *IEEE Hot Chips Symposium*, pages 1–19. IEEE Computer Society.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *CoRR*, abs/1609.07843.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning*.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *IEEE International Conference on Computer Vision*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Annual Meeting of the Association for Computational Linguistics*.

Qualcomm. 2024. Unlocking on-device generative ai with an npu and heterogeneous computing. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/Unlocking-on-device-generative-AI-with-an-NPU-and-heterogeneous-computing.pdf.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances on Neural Information Processing Systems*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An open-source small language model. *Preprint*, arXiv:2401.02385.

765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801

A Appendix: On-device Experiments for W4A8

In this appendix, we provide further on-device evaluation for W4A8. The current on-device toolchains we use support only symmetric per-channel weight quantization. This, however, typically leads to performance degradation as the full quantization range may not be fully utilized if the weights are biased toward positive or negative. Here we first present extra W4A8 results with symmetric per-channel quantization. We then include on-device latency evaluation showcasing the advantages of using 4-bit integer representation for the weights.

A.1 Symmetric vs Asymmetric W4A8 Quantization

We train extra W4A8 models with symmetric per-channel quantization. Table 7 presents the performance of symmetric per-channel W4A8 models on Wikitext (Gao et al., 2023), confirming the performance degradation compared to the asymmetric counterparts.

WikiText (↓)	TinyLLaMA 1.1B	StableLM-2 1.6B	Gemma 2B
FP16	14.88	28.41	17.96
MobileQuant-Asym	16.7	32.9	22.7
MobileQuant-Sym	17.9	35.9	24.6

Table 7: Evaluation of symmetric vs asymmetric W4A8 per-channel quantization on Wikitext (Gao et al., 2023).

Method	TinyLlaMA-1.1B	Gemma-2B
Prompt Encoding (Seq. Length 256)		
W8A16	510	1191
MobileQuant (*W8A8*)	276	752
full W8A8	89	311
W4A16	320	617
MobileQuant (*W4A8*)	239	460
full W4A8	89	98
Autoregressive Gen. (Context Length 1024)		
W8A16	54	78
MobileQuant (*W8A8*)	46	60
full W8A8	42	59
W4A16	50	56
MobileQuant (*W4A8*)	38	40
full W4A8	40	40

Table 8: On-device latency (ms) for TinyLlaMA-1.1B (Zhang et al., 2024) and Gemma-2B (Google, 2024) across different settings.

A.2 On-device Latency for Symmetric W4A8 models

We further evaluate the on-device latency of the W4A8 models with symmetric quantization. Table 8 shows that, compared to W8A8, the W4A8 models demonstrate improved inference speed for both prompt encoding and autoregressive generation. For larger models like Gemma-2B, the improvements are more significant, *i.e.* reducing the latency of prompt encoding and autoregressive generation by 39% and 33%. Here, TinyLLaMA-1.1B achieves the same inference speed, *i.e.* 40 ms per token (25 tok/s). We conjecture that, in this setting, the autoregressive generation for these models is likely memory-bound. We plan to further investigate the performance bottleneck in future research.

11