Characterizing Deep Learning Model Compression with Post-Training Quantization on Accelerated Edge Devices

Rakandhiya D. Rachmanto*, Zaki Sukma†, Ahmad N. L. Nabhaan*,
Arief Setyanto*, Ting Jiang†, and In Kee Kim†
*Universitas Amikom Yogyakarta, {rakandhiya_daanii, ahmad.nabhaan}@students.amikom.ac.id, arief_s@amikom.ac.id
†University of Georgia, {zaki.sukma, ting.jiang1, inkee.kim}@uga.edu

***Abstract*—Edge AI has increasingly been adopted due to the rapid development of deep learning and AI. At the same time, as AI models quickly grow in size and complexity, resource-constrained edge devices face significant challenges in executing such complex models. Model compression, particularly post-training quantization, offers a viable approach by reducing model size and resource demands, making these models more suitable for the deployment on edge devices. However, despite its significance, the effects of model compression on edge devices have yet to be thoroughly explored.**

We address this gap by thoroughly characterizing post-training quantization on accelerated edge devices. We use six different deep learning models with varied sizes (in MB) and resource demands. We first characterize on-device post-training quantization on edge devices. Subsequently, we perform a detailed characterization of the performance and behaviors of quantized deep learning models with different precision modes. We discuss the benefits of post-training quantization, including reduced model size, improved inference latency, and decreased resource consumption. We also provide a detailed analysis of the downside of the quantized models, focusing on the reduction of their inference accuracy.

***Index Terms*—Edge AI; Edge Devices; On-Device Compression; Post-Training Quantization;**

I. Introduction

The rapid development and significant achievements of deep learning (DL) and artificial intelligence (AI) have led to their widespread adoption in edge computing [1]–[4]. As DL/AI models continue to grow in size and complexity [5]–[7], deploying them on edge devices, which are typically resource-constrained, presents significant challenges. Edge devices are often equipped with energy-efficient cores and limited memory spaces [8]–[10]. Additionally, their memory systems usually adopt a unified/shared model between CPUs and GPUs, and the number of GPU cores is much smaller compared to those in high-end GPUs. Consequently, directly deploying complex and large-scale DL models on edge devices involves overcoming substantial obstacles, including high computational demands, excessive energy consumption, and the need for large memory spaces that are beyond the capabilities of most edge devices.

In response to these challenges, model compression techniques have gained increasing attention [11]–[16]. By employing methods, *e.g.*, quantization and pruning, it is possible to significantly reduce the size of DL models, enabling them to run efficiently on resource-constrained edge devices. While model compression has traditionally been performed offline using powerful GPUs, there is a growing demand for on-device compression [17]–[20]. This approach can greatly enhance the capabilities of resource managers and schedulers for edge AI by allowing them to dynamically create optimized models to address varying AI inference requests and performance goals.

Designing a system capable of on-device compression requires in-depth knowledge and detailed characterization of compression methods across various edge devices. However, this area remains largely *unexplored*. It is crucial to understand the overhead and resource requirements for compression operations, as well as the performance benefits, including improved inference latency and throughput. Equally important is identifying the impact of compression on resource consumption/utilization and potential downsides, such as decreased inference accuracy.

To address this gap, we perform a comprehensive characterization study to understand the opportunities and current limitations of model compression on edge devices. Specifically, we focused on Post-Training Quantization (PTQ) [12], [21] for model compression, considering its advantages, such as lightweight and resource-efficient nature and the elimination of expensive additional training processes. These factors make PTQ suitable for edge AI and potentially for online and on-device compressions.

In this study, we employ six DL models specialized in image classification tasks, categorized by model size (in MB) into small, medium, and large models. Furthermore, we utilize two widely used accelerated edge devices with differing hardware specifications: NVIDIA's Jetson Xavier (Xavier) [22] and Orin Nano (Orin) [23]. Although both devices have similar HW specifications for CPU cores and memory size, Orin is equipped with a significantly larger number of edge GPU cores (2.7×), expected to enhance AI processing performance. By employing these devices, we aim to assess the impact of GPU size on the compression workflow and the processing efficiency of quantized DL models. We explore multiple precision modes, *e.g.*, 16-bit floating point (FP16), 8-bit integer (INT8), in PTQ to determine the advantages and disadvantages of each configuration. Our goal is to identify the most significant expected benefits and drawbacks of model compression and quantized models, specifically latency improvement and accuracy drops, on Xavier and Orin.

We found several key observations of the PTQ process and the impact of quantized DL models on edge devices.

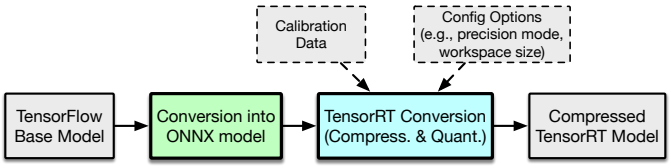


Fig. 1. **Overall Flow of PTQ-based Model Compression on Edge Devices.**

First, the compression time and overhead for on-device PTQ vary widely depending on the model size, device's HW/GPU capacity, and precision mode configurations. In particular, PTQ with INT8 precision mode leads to longer compression times due to additional calibration requirements. Second, PTQ-based compression can decrease the model size to 58% of its original, and models quantized with INT8 precision are further reduced to 77% of the original size. Additionally, despite the considerable reduction in the model size, the quantized models' accuracy loss remains relatively minor. For example, models quantized to FP16 precision mode experience up to a 4.5% drop in accuracy, whereas the INT8-based quantized models can have up to a 13.9% accuracy decrease. The reduction in model size also leads to lower GPU and memory consumption, though CPU usage becomes higher as the CPU handles scheduling for the smaller (quantized) models more frequently. Lastly, we also confirm that quantized models can achieve significant improvements in inference latency, with reductions between 55% and 67% compared to the original models.

This work makes several research contributions:

1. Characterizing the on-device PTQ process on edge devices: We divide the PTQ compression process into two phases: 1) the base model conversion to an ONNX (Open Neural Network eXchange) [24] representation and 2) TensorRT conversion (quantization). We provide detailed analyses of the overhead and resource consumption for each step of the PTQ pipeline.

2. Examining the impact of HW capacity in edge devices on PTQ: We examined the comparison of the impacts of GPU capacity (core numbers) on model compression effectiveness, specifically identifying how differences in GPU core count in edge devices impact the PTQ compression workflows.

3. Conducting a detailed analysis of the impact and performance of quantized models: We evaluate the various aspects and performance of the quantized models, including model size reduction, reduction in resource consumption after quantization, accuracy changes of the quantized model, as well as latency improvement.

The rest of this paper is organized as follows: §II describes the background of PTQ on the accelerated edge devices. §III provides our benchmark setup in detail. §IV reports our evaluation and characterization results regarding the on-device PTQ process as well as the performance and behavior of the quantized DL models. §V describes related work. Finally, §VI concludes this paper.

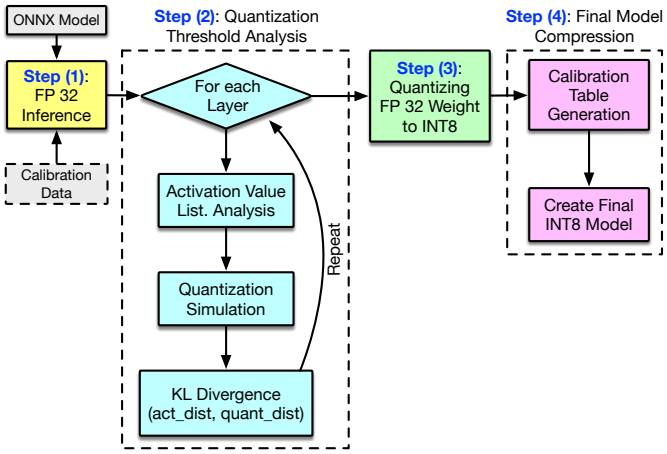


Fig. 2. **The Detailed Procedure of PTQ with INT8 Precision Mode.**

II. Background of Post-Training Quantization

Model compression techniques, such as quantization, pruning, and knowledge distillation, are increasingly becoming essential components of enabling edge AI [1], [25]. Model compression aims to reduce the complexity and size of DL models, with the goal of enhancing latency, optimizing energy consumption, and minimizing the impact on model accuracy. In particular, smaller model sizes with reduced memory requirements are well-suited to the resource constraints of edge devices, thereby being widely deployed across various edge AI inference services.

Overview of On-device Post-Training Quantization (PTQ). PTQ is a widely-used approach for compressing DL models [12], [21]. PTQ is a subclass of quantization-based model compression, along with quantization-aware training [12], [13]. In quantization, compression can be done by adjusting the precision of a DL model and weights to a lower-bit representation. In this study, we specifically investigate the PTQ-based model compression on NVIDIA Jetson edge devices and characterize the conversion of TensorFlow models from 32-bit floating point (FP32) into 16-bit floating point (FP16) and 8-bit integer (INT8) models using PTQ through TensorRT.

To perform PTQ on NVIDIA edge devices, NVIDIA offers standalone toolkits and integrations for DL frameworks, *e.g.*, trtexec[1], Polygraphy[2], and TF-TRT[3], to create compatible TensorRT models for NVIDIA's edge GPU architectures. The general compression procedure with PTQ (shown in Fig. 1) is as follows: The compression pipeline takes a pre-trained (base) model as an input. The base pre-trained model is converted into an ONNX model, an open-standard representation of ML models [24]. The TensorRT conversion process (in Fig. 1) involves compression and quantization stages with specific compression configurations. These configurations include precision modes like FP16 or INT8, workspace size parameters

[1] https://github.com/NVIDIA/TensorRT/tree/main/samples/trtexec
[2] https://github.com/NVIDIA/TensorRT/tree/main/tools/Polygraphy
[3] https://github.com/tensorflow/tensorrt

Table 1. **Deep Learning Model Information and Top-1 Accuracy.** *"Oxford": Oxford 102-flowers dataset, "C100": CIFAR-100 dataset.*

Model (Abbr.)	Size (MB)	Input Size	Accuracy	
			Oxford	C100
MobileNetV3-Small (Mob-S)	4.7	224 × 224	73.73	63.10
MobileNetV3-Large (Mob-L)	9.7	224 × 224	78.82	66.99
EfficientNetB1 (Eff-B1)	33	224 × 224	79.43	70.33
EfficientNetB3 (Eff-B3)	50	224 × 224	75.83	71.97
DenseNet169 (D169)	61	224 × 224	81.12	70.12
DenseNet201 (D201)	85	224 × 224	83.30	71.41

indicating maximum memory requirements, and options for leveraging DL accelerators. Finally, upon completion of the quantization process, the TensorRT model is generated, ready for inference tasks, such as image classifications and object detections.

However, the compression procedure can vary depending on the selection of precision mode, such as FP16 vs. INT8. Quantizing to INT8 is more complex than FP16-based quantization due to the limited value range of INT8 quantization, ranging from -128 to $+127$. As a result, PTQ with INT8 precision requires an additional calibration step, which requires a specific input referred to as calibration data [26].

Fig. 2 depicts the detailed procedure of PTQ with INT8, which consists of four steps. In **Step (1)**, TensorRT performs FP32 inferences on the calibration dataset to capture the baseline performance and activation data for the model. **Step (2)** performs a quantization threshold analysis in the ONNX model, repeated across all layers. In this step, for each layer, TensorRT analyzes the distribution of activation values and simulates quantization at various thresholds to assess the impact of different quantization levels. This simulation and analysis employ the Kullback-Leibler (KL) divergence [27] to measure the accuracy loss between the original FP32 activation distributions and the potential INT8 quantized distributions. The threshold that results in the least KL divergence is selected as the optimal scaling factor (having minimal accuracy loss), and it guides the next quantization steps. **Step (3)** involves using this optimal scaling factor to convert the FP32 weights (and activations) into INT8. Finally, in **Step (4)**, TensorRT generates a Calibration Table, which is used to determine the appropriate quantization parameters (*e.g.*, for weights and activations) for converting from FP32 to INT8 with minimal accuracy loss and memory requirements. The determined quantization parameters are then applied to create the final compressed INT8 model.

Why PTQ? While various compression methods [14]–[16] exist to reduce model size and complexity, our study focuses on characterizing the performance and overhead of PTQ on edge devices. We specifically select PTQ for its efficiency and suitability for on-device compression, which is augmented by TensorRT's additional optimizations. For example, PTQ is lightweight and faster than other compression techniques because it does not require additional re-training stages [28]. In contrast, the compression pipelines of other methods (*e.g.*, pruning and knowledge distillation) are more extensive and require extra steps for training and parameter optimizations compared to PTQ's compression pipelines. Specifically, PTQ with TensorRT requires only two inputs: the base pre-trained model and calibration data. Furthermore, TensorRT enhances model compression with PTQ through additional operations like layer fusions, reduction operations, and multilayer perceptions, resulting in smaller models with enhanced latency performance [29].

Table 2. **Two Accelerated Edge Devices: NVIDIA Jetson Xavier NX and Jetson Orin Nano.**

	Jetson Xavier NX	Jetson Orin Nano
CPU	6-core Carmel @1.9GHz	6-core Cortex-A78AE @1.5GHz
GPU	384-core Volta, 48 Tensor cores @1100 MHz	1024-core Ampere, 32 Tensor cores @625 MHz
Memory	Shared 8GB LPDDR4X	Shared 8GB LPDDR4X
Power Mode	10W, 15W, 20W	7.5W, 15W
OS	Ubuntu 20.04	Ubuntu 20.04
JetPack	ver. 5.1	ver 5.1

III. Experiment Setup and Benchmark Procedures

We now describe our experiment setup and benchmarking procedures to characterize the performance and overhead of PTQ-based model compression on edge devices.

A. Experiment Setup

DL Models and Benchmark Datasets. We employ six pre-trained DL models, as described in Table 1, and the models are selected based on their size. For example, MobileNetV3-Small (Mob-S) and MobileNetV3-Large (Mob-L) models [30] are lightweight models with model sizes of 4.7MB and 9.7MB, respectively. Medium models include EfficientNetB1 (Eff-B1) and EfficientNetB3 (Eff-B3) [31], with sizes of 33MB and 50MB. DenseNet169 (D169) and DenseNet201 (D201) [32] are the large models, sized at 61MB and 85MB, respectively. These six models are built for image classification tasks, which are widely used AI inference tasks at the edge, including self-driving vehicles, surveillance, drone tasks, and healthcare [1].

For benchmark datasets, we utilize two image classification datasets: Oxford 102-flowers [33] and CIFAR-100 [34]. Oxford 102-flowers dataset includes 102 classes, ranging from 40 to 258 images per class, totaling 1024 images for training and validation. CIFAR-100 dataset has 100 classes with 60,000 images. Table 1 also reports the image classification accuracy of six DL models on both datasets.

DL Framework and Libraries. We utilize two DL frameworks and multiple libraries essential for the PTQ compression process: TensorFlow [35] and TensorRT [36]. TensorFlow is employed to build the uncompressed baseline model, and TensorRT is used to perform the PTQ compression task, which will create a model with reduced size and improved inference speed. Furthermore, TensorRT is also utilized in our characterization study to evaluate the performance and behavior of the compressed model on edge devices.

As previously discussed, TensorRT requires an ONNX [24] model to facilitate the PTQ compression process. The tf2onnx[4] tool is responsible for converting the TensorFlow model into this ONNX representation. Following this conversion, the Polygraphy toolkit provides the necessary functionality for TensorRT to support models in the ONNX format.

Accelerated Edge Devices. We employ two edge devices, and their HW specifications are described in Table 2.

- **Jetson Xavier NX** (XAVIER) [22] is an edge device based on the power-efficient Xavier SoC. XAVIER has six NVIDIA Carmel CPU cores, a 384-core Volta GPU with 48 Tensor cores, and 8GB of LPDDR4X memory shared by the CPU and GPU.
- **Jetson Orin Nano** (ORIN) [23] is a recently released edge device from NVIDIA, based on the newer Orin SoC. ORIN features a six-core Arm Cortex-A78AE processor, a 1024-core Ampere GPU with 32 Tensor cores, and 8GB of shared LPDDR5 RAM for both CPUs and GPUs.

Default Configurations. While both devices offer different power modes (*e.g.*, high-performance mode vs. energy-efficient mode), for this study, they are set to their maximum power mode. For example, XAVIER is configured to a 20W mode, enabling all six cores at their maximum frequency of 1.9GHz and activating all GPU cores at their maximum speed of 1100MHz. ORIN operates in a 15W power mode, which allows all six cores to run at the same 1.5GHz frequency and all 1024 GPU cores at their maximum speed of 625MHz. Additionally, we enable '`jetson_clocks`' to ensure the devices operate at maximum CPU and GPU frequencies.

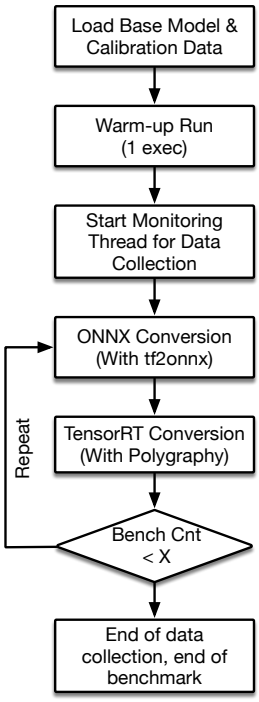


Fig. 3. **The Overall Procedure of PTQ Compression Benchmark.**

To ensure a fair comparison and characterization, we use identical versions and configurations of the necessary packages on both devices. For example, both devices are equipped with TensorFlow v2.12, TensorRT v8.5.2, ONNX v1.15.0, tf2onnx v1.16.1, Polygraphy v0.47, CUDA v11.4, and cuDNN v8.6.0. The OS and JetPack SDK versions are also the same for both devices, with Ubuntu 20.04 and JetPack 5.1 being used.

B. Benchmark Procedure

Our characterization study consists of two parts: (1) characterization of PTQ compression on edge devices and (2) analysis of the performance and behavior of compressed/quantized models on the devices. Therefore, we perform two measurements associated with these characterization studies.

PTQ Compression Benchmark. As discussed in §II, PTQ compression involves two primary steps: 1) ONNX conversion for creating an intermediate representation and 2) TensorRT conversion for quantization and compression. This benchmark aims to measure the overhead and resource consumption during these two critical steps. The PTQ compression benchmark procedure is illustrated in Fig. 3. The benchmark will repeatedly measure both conversion steps to ensure accurate data collection by minimizing the impact of outliers. The very first execution is treated as a warm-up run and is not considered in the final analysis. Subsequently, the benchmark initiates a monitoring thread to collect various metrics, including resource consumption and total compression time. The monitoring thread continues to collect necessary metrics and data during the two conversion steps of the compression pipeline. This process is repeated until the benchmark count reaches a predefined number (in our case, five measurements are performed). After completing all measurements, the benchmark compiles the collected metrics and data, concluding the benchmark process.

Compressed Model Benchmark. This benchmark aims to collect data to characterize the performance and behavior of the compressed model via PTQ. It will primarily collect data on improved inference latency and resource consumption. Fig. 4 illustrates the benchmark pipeline. The initial step involves loading the compressed DL model using TensorRT. The first five inference executions are performed as warm-up runs and will be discarded. The benchmark then initiates monitoring threads for data collection. Subsequently, it conducts actual inference tasks, during which the monitoring thread collects various performance metrics. Similar to the previous benchmark, the inference tasks are executed multiple times to ensure the accuracy of data collection; in our case, we perform 30 inference tasks. Finally, the benchmark compiles the collected data and concludes the benchmark process.

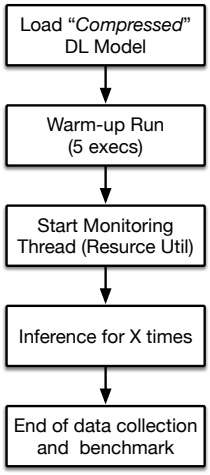


Fig. 4. **The Procedure of Benchmark for Compressed DL Models.**

Performance Metrics. For the PTQ compression benchmark, we assess the compression overhead and resource consumption (*e.g.*, CPU, memory, GPU) during the compression process. Specifically, the compression overhead arises from two main steps: ONNX conversion and TensorRT conversion. We measure the overhead associated with each step. We utilize `pidstat`[5] to measure CPU and memory utilization, and we employ `tegrastats`[6] for measuring GPU utilization.

[4] https://github.com/onnx/tensorflow-onnx

[5] https://man7.org/linux/man-pages/man1/pidstat.1.html

[6] https://docs.nvidia.com/drive/drive-os-5.2.0.0L/drive-os/index.html

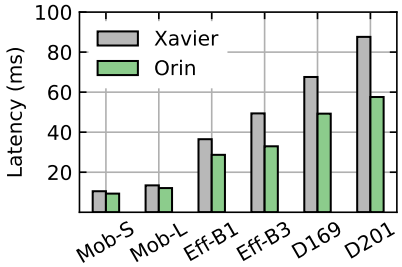


Fig. 5. **Inference Latency of Base Six Models on Both Edge Devices.**

For the compressed model benchmark, we evaluate the accuracy, latency, and size of the compressed models, as well as resource consumption during inference tasks on the devices.

IV. Characterization Results

This section provides our characterization results of a compressed DL model with PTQ. We start by characterizing the base model's performance and behaviors on both edge devices (§IV-A) and then move on to a detailed analysis of on-device PTQ compression (§IV-B). Finally, we present our analysis and characterization on the performance and behaviors of quantized models (§IV-C).

A. Base DL Model Characterization

Our base model characterization focused on measuring inference latency and resource consumption for six DL models across various edge devices.

Inference latency. Fig. 5 shows the inference latency of the six base DL models before quantization. We tested DL models' inference latency with both Oxford 102-flowers and CIFAR-100 datasets with 2500 images, and the figure shows the average inference time for a single image. Orin provides faster inference times across all six models than those observed on Xavier. This result is anticipated, given that Orin has approximately 2.7× more CUDA cores in its edge GPUs, enabling it, on average, to offer inference times that are 23% faster than Xavier.

Furthermore, inference latency appeared to increase with model size. For example, the two smaller models (Mob-S and Mob-L) processed a single image inference in 10.7ms to 12ms. The medium models (Eff-B1 and Eff-B3) showed one image inference times of 43ms on Xavier and 31ms on Orin. The two larger models (D169 and D201) exhibited inference latencies of 77.6ms on Xavier and 53ms on Orin.

Resource utilization. We measured GPU, CPU, and memory utilization while the models performed inference operations, as shown in Fig. 6. The edge GPUs on both devices showed increased utilization with larger model sizes. Additionally, Xavier had 13% to 18% higher GPU utilization (Fig. 6a), attributable to its fewer number of GPU cores. Consequently, Orin, leveraging its larger edge GPUs and more GPU cores (Fig. 6b), enabled faster inference times.

Interestingly, CPU utilization decreased as the model size increased (as shown in Fig. 6c), which is opposed to the GPU utilization patterns with larger models. Our further analysis revealed that this is mainly due to larger models' higher latency (inference time), requiring longer GPU processing time. With edge GPUs' FIFO with non-preemptive scheduling [37], [38], CPU performs fewer scheduling tasks for larger DL models but more frequent GPU scheduling operations for smaller DL models. As a result, the CPU on Xavier and Orin had high utilization with smaller DL models and lower utilization with larger DL models.

Memory consumption remains stable regardless of the model size, as shown in (Fig. 6d). This stability is primarily due to the memory utilization being determined by the TensorRT framework we utilized. Of course, there is memory consumption from DL models, but the framework is the dominant consumer of memory resources. However, we

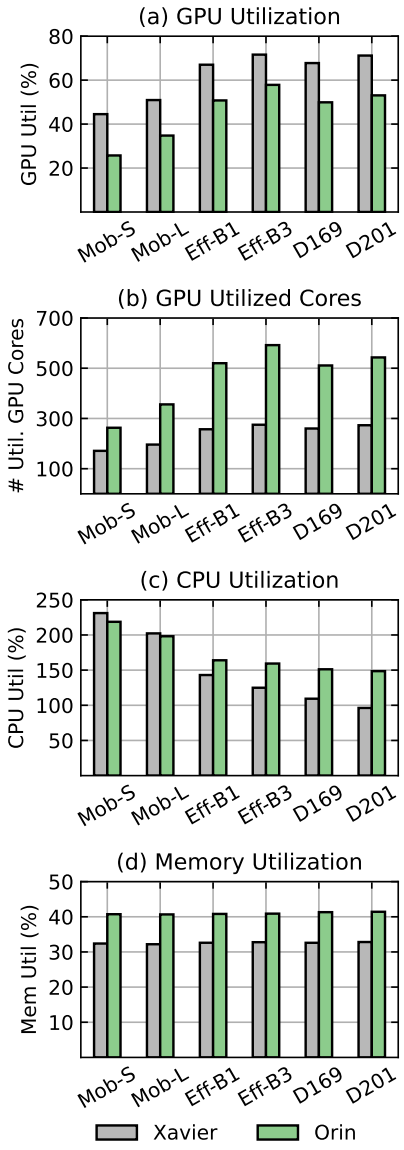


Fig. 6. **Resource Utilization of Six Base Models on Xavier and Orin.** *(a): GPU Utilization, (b) The number of Utilized CUDA Cores, (c) CPU utilization, and (d): Memory utilization.*

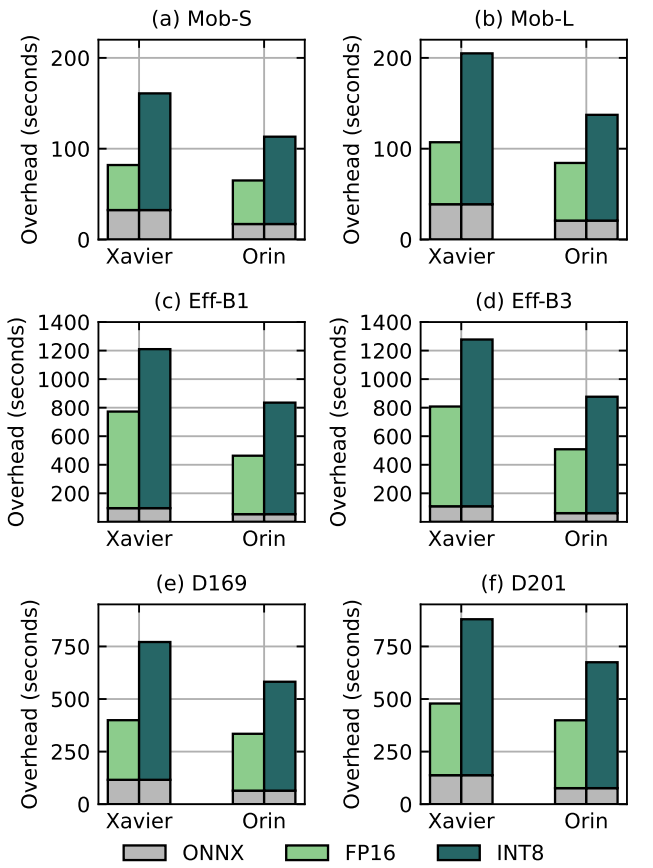


Fig. 7. **PTQ Compression Overhead (Duration in Seconds) of Six DL Models with Two Precision Modes (FP16 and INT8) on XAVIER and ORIN.**

observed that memory utilization can still vary per device. ORIN exhibited a 10% higher utilization compared to XAVIER, despite both devices having an identical memory capacity of 8GB. This difference primarily arises because Nvidia's edge devices utilize a unified memory system shared between CPUs and GPUs [39]. As a result, to support more CUDA cores, ORIN needs additional memory resources, leading to its 10% higher memory utilization.

B. On-Device PTQ Compression Characterization

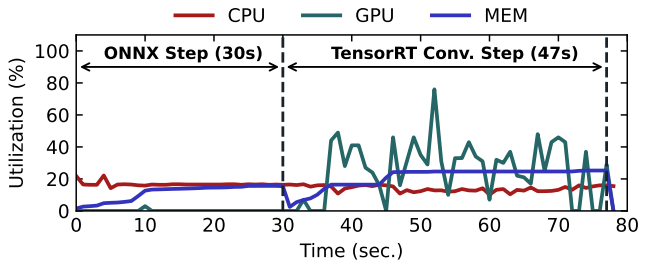
In this characterization, we evaluate the overhead and resource utilization when performing on-device PTQ compression on XAVIER and ORIN. FP-16 and INT8-based quantization processes differ slightly, potentially leading to variations in model performance and operations. Consequently, we utilize FP16 and INT8 as the precision modes for PTQ compression.

PTQ Compression Overhead. We first evaluated the overhead (the duration of PTQ compression) on XAVIER and ORIN, with the results shown in Fig. 7. PTQ compression process involves two steps: conversion to ONNX format and TensorRT conversion (quantization and compression), as detailed in §II. The ONNX conversion is a common task. Both FP16 and INT8 precision modes had identical durations for converting the base model into ONNX representation when applied to the same model on the same device. However, the TensorRT conversion step, which is to quantize and compress models, varied significantly, ranging from 32s to 1150s, as per the selection of precision mode and the device.

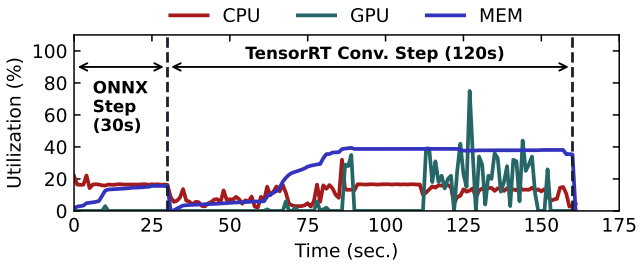
We found that three factors can significantly impact the PTQ compression time: the precision mode, the device's resource capacity, and the model's size. Specifically, the INT8 precision mode required a significantly longer compression time (1.9×) compared to the FP16 precision mode. As illustrated in Fig. 2 (§II), INT8-based PTQ needs additional calibration steps, which naturally extend the compression time. Additionally, the hardware resource capabilities of the edge devices can significantly affect the PTQ compression time. For example, the PTQ compression process on ORIN was considerably faster than on XAVIER. Although times varied across different models and precision modes, Orin demonstrated a 1.3× to 2.77× faster compression time, leveraging its larger GPU capacity. Moreover, the model size could be another determinant of compression time. Mob-S, the smallest model in our study, showed the shortest compression time compared to the medium and large models. Yet, the longest compression times for Eff-B1 and Eff-B3 models suggested that other factors could also influence the duration of PTQ compression. One possible factor could be the distinction in layer types across model architectures, which may result in different optimization steps being applied during PTQ. We observed that TensorRT performed more fusion and erase operations in Eff-B1 and Eff-B3 compared to D169 and D201. Additionally, due to architectural differences between models, a layer that exists in both models could be optimized in one model while being skipped in the optimization process of the other during quantization. For example, during on-device compression, TensorRT spent considerable time optimizing FusedBatchNormV3 operators (an optimized version of a batch normalization layer) in Eff-B1 and Eff-B3, whereas these operators were skipped in D169 and D201. In particular, TensorRT executed this optimization more than 100 times, with each optimization taking 0.6s to 8s for both Eff-B1 and Eff-B3.

Resource Utilization Variation During On-Device PTQ. We also measured specific resource utilization during the PTQ compression processes on both devices. Fig. 8 shows the variation in resource utilization during two steps in the PTQ process. In this figure, we report on six examples of PTQ compression: quantizing Mob-S with both FP16 and INT8 on XAVIER, Eff-B3 with FP16 and INT8 on XAVIER, and D201 with FP16 and INT8 on ORIN. We omit other results due to page limitations, but the outcomes of these six cases are representative of the others.

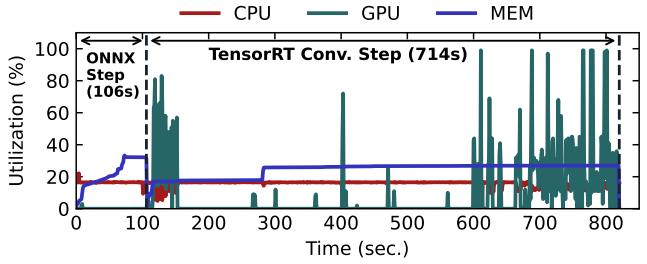
1) ONNX Conversion Step: As the results show, the ONNX conversion step utilized only CPU and memory resources. On average, this step required about 20% of CPU resources across all models. Memory consumption increased gradually over time and was directly proportional to the model size. For example, during the ONNX conversion of Mob-S, memory utilization peaked at only 16%. In contrast, the ONNX con-



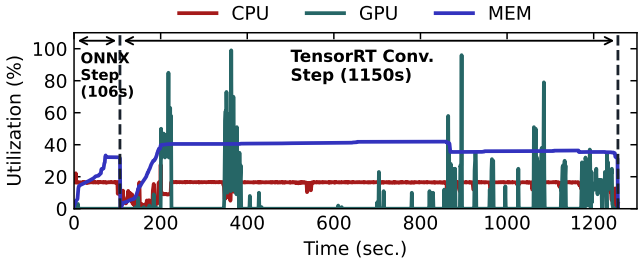
(a) Mob-S (FP16) on XAVIER



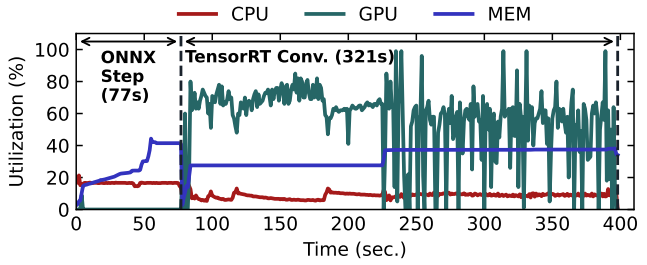
(b) Mob-S (INT8) on XAVIER



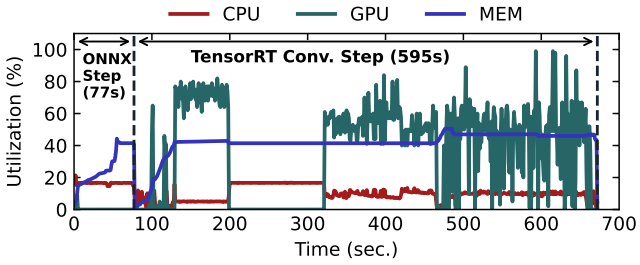
(c) Eff-B3 (FP16) on XAVIER



(d) Eff-B3 (INT8) on XAVIER



(e) D201 (FP16) on ORIN



(f) D201 (INT8) on ORIN

Fig. 8. **Resource Utilization Variations during On-Device PTQ Process.**

versions for Eff-B3 and D201 reached as high as 33.5% and 44.3% in memory usage, respectively.

2) TensorRT Conversion (Quantization) Step: The TensorRT conversion step began utilizing GPUs, with TensorRT, which is a framework designed to facilitate GPU usage on NVIDIA's edge devices. However, GPU utilization significantly varied over time, often reaching 100% but also dropping to 0%. For example, the GPU utilization often became idle during the conversion process, as observed in the Mob-S (Fig. 8d) and D201 (Fig. 8f) INT8 conversions. This fluctuation is mainly due to the necessary CPU-bound operations during the quantization and compression phases, during which CPU usage increased.

Additionally, memory and CPU resources were consistently utilized throughout this step, though not always at high utilization. Memory usage increased gradually during the conversion, with INT8-based conversions/quantizations requiring more memory than FP16-based conversions due to the additional calibration process outlined in §II. CPU usage in the TensorRT conversion step varied across different models, with fluctuations observed in all models.

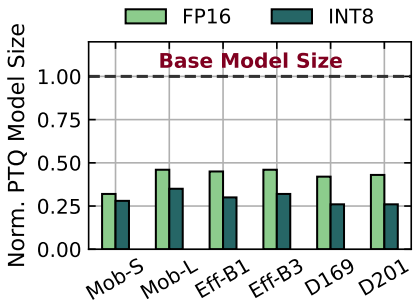


Fig. 9. **Normalized Size of Quantized Models After PTQ Compression.** *The dotted line at 1.0 represents the normalized base model size. The actual model sizes in MB are detailed in Table 1 of §III.*

C. Quantized DL Model Characterization

We now evaluate various aspects of quantized DL models, including reduced model size, accuracy changes, resource utilization, and latency improvement.

Model Size Reductions via PTQ. As expected, PTQ significantly reduces model sizes. Fig. 9 reports the normalized size reduction of six DL models achieved through PTQ com-

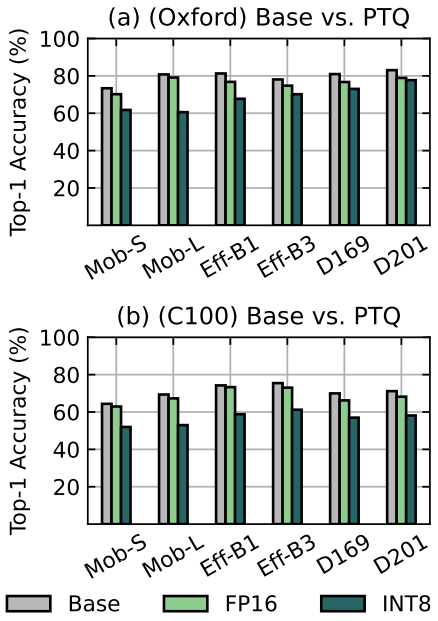


Fig. 10. **Accuracy of Quantized DL Models.** *Base: Accuracy of Base DL Models, FP16: Accuracy of FP16-based Quantized Model, INT8: Accuracy of INT8-based Quantized Model.*

pression with FP16 and INT8. FP16-based quantized models experienced a 58% reduction in size, while models quantized to INT8 could achieve more substantial reductions of over 70%. For example, MOB-S, our smallest model, compressed from its original size of 4.7MB to 1.5MB using FP16 and further to 1.3MB with INT8. Similarly, D201, our largest model, had its size reduction from 86MB to 37MB with FP16 quantization and to 22MB with INT8. This greater efficiency in size reduction with INT8-based PTQ is primarily due to its storage efficiency during the PTQ process. Specifically, INT8 requires only 1 byte per a quantized value, compared to the 2 bytes needed for FP16 values, leading to more significant reductions in model size.

Accuracy Changes in Quantized Models via PTQ. We measured the top-1 accuracy of compressed DL models via PTQ in two different precision modes (FP16 and INT8), and we present the accuracy results in Fig. 10. We observed significant differences in accuracy between the quantized models under these two precision modes. For example, FP16-based quantized models showed less top-1 accuracy loss, with an average reduction of 2.9% compared to the base models' accuracy[7], which ranged from 0.94% to 4.49%. However, INT8-based quantized models experienced a significantly higher average accuracy drop of 13.9% compared to the base models.

There are factors that contributed to the significant accuracy drop of INT8-based quantized models compared to FP16-based models. First, we trained the base models for 10 epochs without any augmentation through transfer learning, using batches of 4 and 32 for the Oxford 102-flowers and CIFAR-100 datasets, respectively. These training parameters contributed to the models' robustness. Second, INT8-based PTQ requires smaller storage for quantizing the original value, *i.e.*, 1 byte for INT8 compared to 2 bytes for FP16. INT8 has a narrower representation range, from -128 to +127, as discussed in §II, which can lead to lower accuracy compared to FP16. Moreover, INT8-based quantization relies heavily on the weights and activations of the model during the calibration process [40]. A more robust model can mitigate the potential accuracy loss resulting from this calibration. However, optimizing the calibration process and hyperparameter tuning are beyond the scope of this work; we plan to address these aspects in future research.

[7]The accuracy of the base models (with Oxford 102-flowers and CIFAR-100 datasets) is also reported in Table 2 in §III.

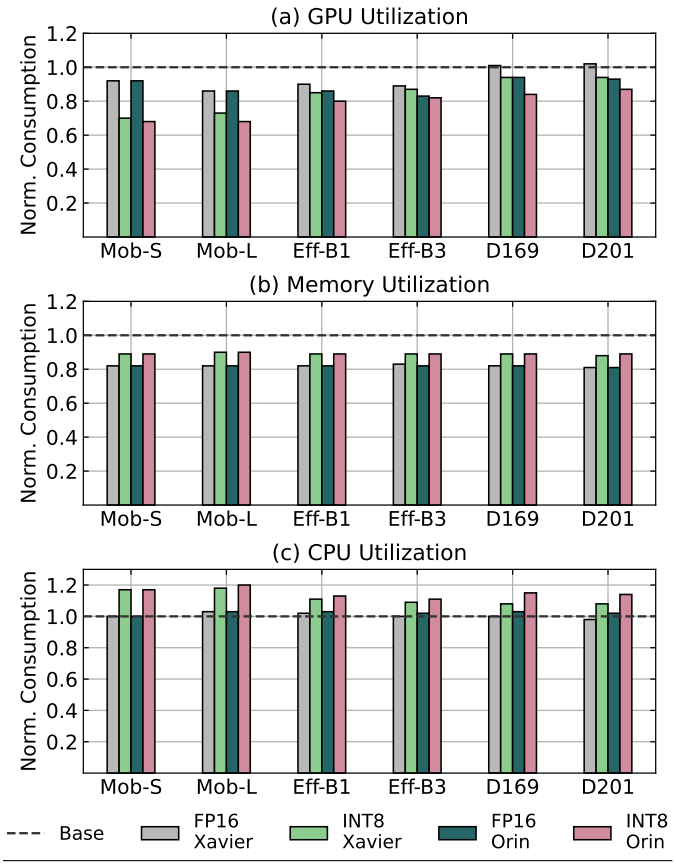


Fig. 11. **Normalized Resource Consumption of Quantized Models.**

Resource Consumption of Quantized Models. A major benefit of PTQ compression, as previously discussed, is the reduced model size, which can lead to decreased resource consumption and enable the models to be deployed on resource-constrained devices. To confirm this benefit, we also measured the resource consumption of the quantized models on XAVIER and ORIN.

Fig. 11 shows the GPU, memory, and CPU resource consumption of quantized models on both edge devices. As expected, running quantized models reduced both GPU and memory resource consumption on both devices. Specifically, on XAVIER, the quantized models consumed 19% less GPU

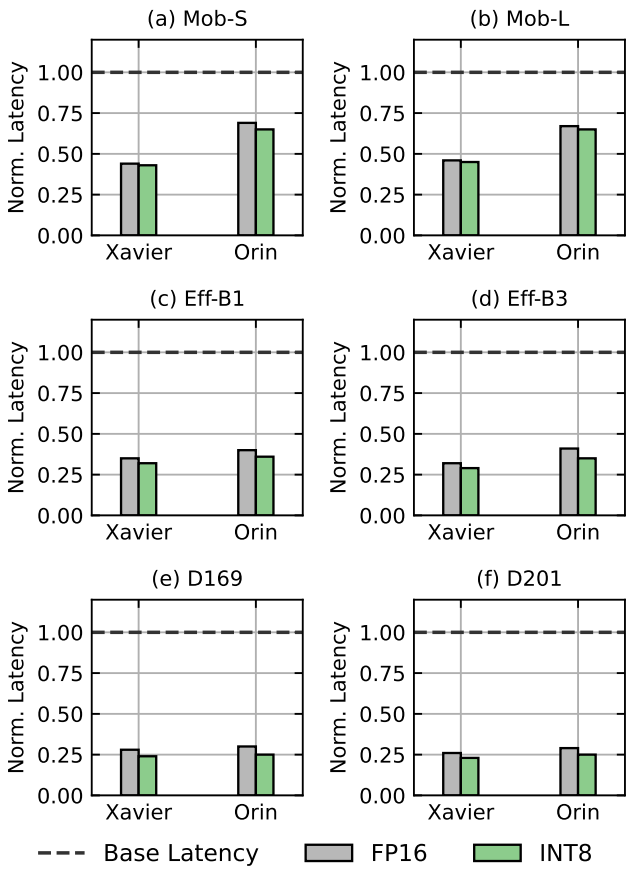


Fig. 12. **Normalized Latency of Quantized Models.** *The figures demonstrate the reduced (improved) latency of quantized models compared to the latency of the base models. These latency results were measured through inference operations on the CIFAR-100 datasets.*

and 9% less memory compared to the orignial models. They also showed a reduction of 10% in GPU and 18% in memory consumption on ORIN. Moreover, between the two precision modes, INT8 offered more benefits in GPU resource consumption compared to FP16, while memory consumption showed no meaningful differences. INT8-based quantized models could further reduce GPU resource consumption by 4.5% to 6% on both devices.

Regarding CPU consumption of quantized models, we observed a 13% increase in CPU utilization on XAVIER and a marginal increase (3%) on ORIN. Furthermore, smaller quantized models exhibited a higher pattern of increase in CPU utilization on XAVIER. Smaller models may trigger more scheduling operations, potentially leading to increased CPU consumption.

Latency Improvement of Qunatized Models. The reduction in model size through quantization can significantly improve the inference latency of the models. Fig. 12 reports the normalized latency of quantized models on both edge devices when performing inference operations on CIFAR-100 datasets. We measured the latency of both FP16- and INT8-based quantized models, which was consistent with previous results.

We observed that all quantized models achieved significant improvements in latency compared to the base models. On both devices, the quantized models showed 67% (XAVIER) and 55% (ORIN) improvements in the inference latency. Additionally, we noted that medium and large models benefit more from quantization in terms of latency improvement. This is primarily because these models can undergo a greater reduction in physical size (MB) compared to Mob-S/Mob-L, which directly contributes to reducing the models' latency. Moreover, INT8-based quantized models can achieve slightly faster inference times, 3% to 4% faster than those of FP16 models.

V. RELATED WORK

Various studies have investigated ML/AI workload characteristics on edge devices, including low-performance single-board computers and specialized AI accelerators like Raspberry PIs, Arduino microcontrollers, NVidia Jetson devices, Intel Movidius VPUs, and edge TPU accelerators [8], [9], [41]–[47]. These studies often reported on performance metrics such as inference latency, overhead, and resource consumption.

Hao and Subedi [44], [45] explored approaches to maximizing inference throughput on edge devices by employing concurrent DL model executions and dynamic model placements on heterogeneous edge AI accelerators. They also identified the maximum supported concurrency level of DL models on multiple edge devices and accelerators. Hadidi *et al.* [8] additionally focused on the thermal aspects of edge devices and conducted a detailed characterization of behaviors of each software stack involved in AI inference tasks.

DeepEdgeBench [43] focused on energy consumption alongside common characterization metrics on edge devices with AI workloads and DL models. Liang *et al.* [9] conducted an evaluation of AI and ML workload performance within edge-cloud collaborative environments examining network latency, bandwidth usage, and resource utilization while implementing model splitting and compression techniques. Their study highlighted the advantages of utilizing edge-cloud co-inference strategies. Specifically, DeepEdgeBench and the work by Liang *et al.*examined certain aspects of model compression by leveraging 8-bit quantized models. However, their focus was either primarily on edge-cloud co-inferences or enabling edge TPU accelerators, which require quantized models. Additionally, these works did not thoroughly explore the characteristics of edge devices and ML frameworks when performing on-device model compression.

pCamp [41] characterized the performance and behaviors of several ML packages and frameworks on edge devices. The study selected five packages, *e.g.*, TensorFlow, Caffe2, MXNet, PyTorch, and TensorFlow Lite, to assess their performance across five edge devices. This work evaluated latency, memory footprint, and energy consumption using two different models: AlexNet (as a larger-sized model) and SqueezeNet (as a smaller-sized model). The findings revealed that the time required to load the models exceeds the time spent executing

them. Additionally, a trade-off between memory usage and latency was observed in all evaluated packages.

Moreover, the runtime overhead of DL/AI workloads on edge devices has also been explored. Ma *et al.* [42] employed common computer vision tasks on three device categories (*e.g.*, GPU-based, ASIC-based, and general-purpose devices) to quantify the runtime overheads. Their experiments across device categories revealed runtime overhead and the impact of different neural network layer types on the runtime overheads.

Finally, Shafi *et al.* [47] also conducted an in-depth analysis of TensorRT, the primary quantization framework in our study on edge devices. The performance evaluation of models compiled with TensorRT shows that the models were able to maintain a similar accuracy compared to the original models, in addition to significantly higher throughput. Furthermore, TensorRT-compiled models led to increased concurrency and GPU utilization.

VI. Conclusion

In this work, we performed a thorough characterization study of PTQ compression on edge devices. We focused on the PTQ compression method because it is lightweight, resource-efficient, and does not require additional, expensive training stages, making it suitable for edge AI use cases. This characterization study was conducted on real-world, resource-constrained edge devices, Xavier and Orin, characterized by limited CPU cores and memory capacity, and equipped with only 384 to 1024 GPU cores. For this characterization, we employed six widely used DL models of varying sizes, from small models like Mob-S/L to larger models like D169/201. We also explored two different precision modes, *e.g.*, FP16, INT8, which can impact the quantized models' size, accuracy, latency, and overall compression time on the devices.

We evaluated the compression overhead (time) and variation in resource utilization on edge devices during on-device compression with PTQ. Subsequently, with the models quantized in two different precision modes, we identified their benefits, including reductions in model size, decreased resource consumption, and improvements in inference latency. Additionally, we identified associated challenges, such as a decrease in accuracy of the quantized models.

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments and constructive suggestions. This research was in part supported by the U.S. Department of Agriculture (USDA) under award number 2021-67019-34342, and the Garuda Ace Program of the Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the USDA or the Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia.

References

[1] Jiasi Chen and Xukan Ran. Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[2] Chinmaya Kumar Dehury, Praveen Kumar Donta, Schahram Dustdar, and Satish Narayana Srirama. CCEI-IoT: Clustered and Cohesive Edge Intelligence in Internet of Things. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Barcelona, Spain, July, 2022.

[3] Shashikant Ilager, Vincenzo De Maio, Ivan Lujic, and Ivona Brandic. Data-centric Edge-AI: A Symbolic Representation Use Case. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Chicago, IL, USA, July, 2023.

[4] Arief Setyanto, Theopilus Bayu Sasongko, Muhammad Ainul Fikri, and In Kee Kim. Near-Edge Computing Aware Object Detection: A Review. *IEEE Access*, 12:2989–3011, 2024.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, Minneapolis, MN, USA, June, 2019.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations (ICLR)*, Virtual Event, Austria, May, 2021.

[7] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. *CoRR*, abs/2302.13971, 2023.

[8] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices. In *IEEE International Symposium on Workload Characterization (IISWC)*, Orlando, FL, USA, November, 2019.

[9] Qianlin Liang, Prashant J. Shenoy, and David E. Irwin. AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures. In *IEEE International Symposium on Workload Characterization (IISWC)*, Beijing, China, October, 2020.

[10] Kaustubh Rajendra Rajput, Chinmay Dilip Kulkarni, Byungjin Cho, Wei Wang, and In Kee Kim. EdgeFaaSBench: Benchmarking Edge Devices Using Serverless Computing. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Barcelona, Spain, July, 2022.

[11] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, June, 2020.

[12] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research*, 18:187:1–187:30, 2017.

[14] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations (ICLR)*, Toulon, France, April, 2017.

[15] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the State of Neural Network Pruning? In *Third Conference on Machine Learning and Systems (MLSys)*, Austin, TX, USA, March, 2020.

[16] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing Neural Networks. In *Annual Conference on Neural Information Processing Systems (NIPS)*, Montreal, Quebec, Canada, December, 2015.

[17] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-Device Training Under 256KB Memory. In *The Thirty-sixth Annual Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, November, 2022.

[18] Achintya Kundu, Laura Wynter, Rhui Dih Lee, and Luis Angel D. Bathen. Transfer-Once-For-All: AI Model Optimization for Edge. In

IEEE International Conference on Edge Computing and Communications (EDGE), Chicago, IL, USA, July, 2023.

[19] Fred Hohman, Mary Beth Kery, Donghao Ren, and Dominik Moritz. Model Compression in Practice: Lessons Learned from Practitioners Creating On-device Machine Learning Experiences. *CoRR*, abs/2310.04621, 2023.

[20] Eric Youn, Sai Mitheran J, Sanjana Prabhu, and Siyuan Chen. Compressing Vision Transformers for Low-Resource Visual Learning. *CoRR*, abs/2309.02617, 2023.

[21] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Accurate Post Training Quantization With Small Calibration Sets. In *The 38th International Conference on Machine Learning (ICML)*, Virtual Event, July, 2021.

[22] NVIDIA Jetson Xavier NX. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[23] NVIDIA Jetson Orin Nano. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.

[24] Open Neural Network Exchange. https://onnx.ai/.

[25] Lingdong Wang, Liyao Xiang, Jiayu Xu, Jiaju Chen, Xing Zhao, Dixi Yao, Xinbing Wang, and Baochun Li. Context-Aware Deep Model Compression for Edge Cloud Computing. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Singapore, December, 2020.

[26] Eunjin Jeong, Jangryul Kim, and Soonhoi Ha. TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards. *ACM Transactions on Embedded Computing Systems*, 21(5):51:1–51:26, 2022.

[27] Fangxin Liu, Wenbo Zhao, Zhezhi He, Yanzhi Wang, Zongwu Wang, Changzhi Dai, Xiaoyao Liang, and Li Jiang. Improving Neural Network Efficiency via Post-training Quantization with Adaptive Floating-Point. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada, October, 2021.

[28] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A Survey of Quantization Methods for Efficient Neural Network Inference. *CoRR*, abs/2103.13630, 2021.

[29] NVIDIA Docs – NVIDIA TensorRT Performance: Best Practices. https://docs.nvidia.com/deeplearning/tensorrt/pdf/TensorRT-Best-Practices.pdf.

[30] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for MobileNetV3. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), October, 2019.

[31] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *The 36th International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, 2019.

[32] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, July, 2017.

[33] Maria-Elena Nilsback and Andrew Zisserman. 102 Category Flower Dataset. https://www.robots.ox.ac.uk/~vgg/data/flowers/102/.

[34] Alex Krizhevsky. CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.

[35] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, USA, November, 2016.

[36] NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.

[37] Ming Yang, Nathan Otterness, Tanya Amert, Joshua Bakita, James H. Anderson, and F. Donelson Smith. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, Barcelona, Spain, July, 2018.

[38] Ignacio Sanudo Olmedo, Nicola Capodieci, Jorge Luis Martinez, Andrea Marongiu, and Marko Bertogna. Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Sydney, Australia, April, 2020.

[39] Jiesong Liu, Feng Zhang, Hourun Li, Dalin Wang, Weitao Wan, Xiaokun Fang, Jidong Zhai, and Xiaoyong Du. Exploring Query Processing on CPU-GPU Integrated Edge Device. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):4057–4070, 2022.

[40] Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International Conference on Machine Learning (ICML)*, Honolulu, Hawaii, USA, July, 2023.

[41] Xingzhou Zhang, Yifan Wang, and Weisong Shi. pCAMP: Performance Comparison of Machine Learning Packages on the Edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*, Boston, MA, USA, July, 2018.

[42] Xiu Ma, Guangli Li, Lei Liu, Huaxiao Liu, Lei Liu, and Xiaobing Feng. Understanding the Runtime Overheads of Deep Learning Inference on Edge Devices. In *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, New York City, NY, USA, September, 2021.

[43] Stephan Patrick Baller, Anshul Jindal, Mohak Chadha, and Michael Gerndt. DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. In *IEEE International Conference on Cloud Engineering (IC2E)*, San Francisco, CA, USA, October, 2021.

[44] Piyush Subedi, Jianwei Hao, In Kee Kim, and Lakshmish Ramaswamy. AI Multi-Tenancy on Edge: Concurrent Deep Learning Model Executions and Dynamic Model Placements on Edge Devices. In *14th IEEE International Conference on Cloud Computing (CLOUD)*, Virtual Event, September, 2021.

[45] Jianwei Hao, Piyush Subedi, Lakshmish Ramaswamy, and In Kee Kim. Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy. *ACM Transactions on Internet Technology (TOIT)*, 23(1):2:1–2:33, 2023.

[46] Prashanthi S. K, Sai Anuroop Kesanapalli, and Yogesh Simmhan. Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 6(3):44:1–44:26, 2021.

[47] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan. Demystifying TensorRT: Characterizing Neural Network Inference Engine on Nvidia Edge Devices. In *IEEE International Symposium on Workload Characterization (IISWC)*, Storrs, CT, USA, November, 2021.