Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy

JIANWEI HAO, PIYUSH SUBEDI, LAKSHMISH RAMASWAMY, and IN KEE KIM,
University of Georgia

The wide adoption of smart devices and Internet-of-Things (IoT) sensors has led to massive growth in data generation at the edge of the Internet over the past decade. Intelligent real-time analysis of such a high volume of data, particularly leveraging highly accurate deep learning (DL) models, often requires the data to be processed as close to the data sources (or at the edge of the Internet) to minimize the network and processing latency. The advent of specialized, low-cost, and power-efficient edge devices has greatly facilitated DL inference tasks at the edge. However, limited research has been done to improve the inference throughput (e.g., number of inferences per second) by exploiting various system techniques. This study investigates system techniques, such as batched inferencing, AI multi-tenancy, and cluster of AI accelerators, which can significantly enhance the overall inference throughput on edge devices with DL models for image classification tasks. In particular, AI multi-tenancy enables collective utilization of edge devices' system resources (CPU, GPU) and AI accelerators (e.g., Edge Tensor Processing Units; EdgeTPUs). The evaluation results show that batched inferencing results in more than 2.4× throughput improvement on devices equipped with high-performance GPUs like Jetson Xavier NX. Moreover, with multi-tenancy approaches, e.g., concurrent model executions (CME) and dynamic model placements (DMP), the DL inference throughput on edge devices (with GPUs) and EdgeTPU can be further improved by up to 3× and 10×, respectively. Furthermore, we present a detailed analysis of hardware and software factors that change the DL inference throughput on edge devices and EdgeTPUs, thereby shedding light on areas that could be further improved to achieve high-performance DL inference at the edge.

CCS Concepts: • **General and reference** → **Measurement**; **Empirical studies**; **Performance**; • **Hardware** → *Sensor devices and platforms*; • **Computer systems organization** → *Heterogeneous (hybrid) systems*; *System on a chip*; • **Software and its engineering** → Software performance; • **Theory of computation** → *Concurrency;*

Additional Key Words and Phrases: Edge computing, AI multi-tenancy, deep learning at the edge, concurrent model executions, dynamic model placements, performance evaluation, and characterization

Jianwei Hao and Piyush Subedi contributed equally to this research.

This research was in part supported by NSF grant SES1637277 and USDA grant 2021-67019-34342. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of NSF and USDA.

Authors' address: J. Hao, P. Subedi, L. Ramaswamy, and I. K. Kim, University of Georgia, Athens, Georgia, 30602; emails: {jhao, piyush.subedi, laksmr, inkee.kim}@uga.edu.

1533-5399/2023/02-ART2 $15.00
https://doi.org/10.1145/3546192

ACM Reference format:
Jianwei Hao, Piyush Subedi, Lakshmish Ramaswamy, and In Kee Kim. 2023. Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy. *ACM Trans. Internet Technol.* 23, 1, Article 2 (February 2023), 33 pages.
https://doi.org/10.1145/3546192

1 INTRODUCTION

High network latency and privacy concerns have hindered the adoption of cloud computing for **Internet-of-things** (**IoT**) and **cyber-physical system** (**CPS**) applications that require prompt responses and collect sensitive user information [27, 47]. Wide-area data transmission from IoT sensors to cloud data centers often causes intolerable network latency [44, 71]. Moreover, IoT systems that monitor user activity (e.g., smart homes and activity trackers) have the risk of a privacy breach because all sensing data need to be stored and processed in cloud data centers [59, 74]. *Edge Computing*, a new computing paradigm, attempts to address those issues by placing computing resources at the edge of the Internet (or closer to data sources) [45, 60, 61]. The proliferation of IoT sensors and **single-board computers** (**SBCs**), e.g., Raspberry Pi [20], enables the adoption of edge computing paradigm across a wide range of latency-sensitive applications, including emergency alert services, traffic monitoring, wearable cognitive-assistance systems, and augmented reality [31, 61].

However, there are computationally-intensive edge applications and tasks that still need offloading to clouds. A typical example is ***deep learning*** (**DL**) applications. While DL technologies are increasingly leveraged in real-world edge applications, the majority of DL tasks are still relying on resources in cloud data centers rather than being processed at the Internet edge. [36, 44, 70]. Steep resource requirements (e.g., CPU, memory, and GPU) of DL tasks arising from high dimensionality of input data and a large number of floating-point operations means that the DL tasks often need to be sent to more powerful cloud data centers [2–5, 64]. Therefore, emerging edge applications, such as autonomous driving, disaster response systems, and drone-based surveillance [50, 55, 75], *may* offer poor **quality of services** (**QoS**) and user experience due to high network latency and privacy issues.

Significant efforts have been conducted to bring DL to the edge of the Internet. In particular, optimizations in hardware and software functionalities are carried out to ensure that the DL models fit in the resource-constrained devices. For example, model compression [30, 38] and quantization [35, 68] techniques are developed to downsize DL models without significant degrade in their accuracy [39, 42, 58]. Similarly, studies involving efficient partitioning of heavier models and subsequent offloading to the cloud [40, 44, 52] show how the edge-cloud hybrid model could be leveraged for DL inference tasks. Moreover, light-weight **machine learning** (**ML**) frameworks, e.g., Tensorflow Lite [51], mobile neural network [43], are developed for enabling on-device inference. Finally, the advent of edge devices with GPUs, e.g., Nvidia's Jetson Series [12, 15, 16], and AI accelerators, e.g., Google's Coral Accelerator [7] and Intel's Neural Compute Stick [11], have played a significant role in enabling edge-based DL inferencing.

Given the significant technological advancement for enabling DL inferencing at the edge, it is crucial to understand the opportunities and limitations of various edge devices and AI-accelerators for real-world DL deployment scenarios. Existing works [25, 34, 48, 49, 57, 73] have quantified the efficiency of various edge devices for DL inference tasks. However, most existing studies have focused on characterizing performance (e.g., latency and throughput) of edge devices and AI accelerators with single DL tasks, which is a significantly limiting assumption in the DL deployment scenario. In real-world edge applications, AI multi-tenancy-based deployments are widely

required, in which multiple DL tasks are co-running on edge devices. For instance, drone-based surveillance requires simultaneous executions of inference tasks on video and audio streams [72]. Furthermore, system approaches to maximize DL inference throughputs (e.g., the number of inferences per second) on edge devices have not been deeply investigated.

In this study, we start by characterizing the performance (e.g., inference throughput) of various edge devices and AI accelerators with a set of pre-trained DL models and popular DL frameworks. Through the characterization step, we obtain the baseline performance of various edge devices/AI accelerators and investigate factors that change the throughput of DL inference tasks on such devices. We then employ three techniques to maximize the DL inference throughput on the devices with single and multi-tenancy-based use cases: *batched inferencing*, ***concurrent model executions (CME)***, and ***dynamic model placements (DMP)***. Batched inferencing is for single-tenancy use cases on edge devices and maximizes the DL inference throughput by exploiting the parallel computing capabilities of computing resources on edge devices. Both CME and DMP are techniques for maximizing DL inference throughput with AI multi-tenancy. CME allows the deployment of multiple DL models on either GPU or EdgeTPU resources and runs them parallel to improve the overall DL inference throughput by simultaneously enabling the execution of different DL models. DMP enables AI multi-tenancy by deploying and executing DL models on different resources on edge devices at the same time. e.g., DL models on both GPU and EdgeTPU. DMP is particularly useful when AI accelerators (e.g., EdgeTPU) enhance edge devices, and it can significantly increase the resource utilization and the DL inference throughput by utilizing multiple resources on the devices and the accelerators. Furthermore, we evaluate DL deployment strategy on edge devices with multiple AI accelerators (e.g., EdgeTPU cluster) and report the benefits and limitations of the deployment scenarios with EdgeTPU clusters.

Our extensive evaluation results confirm that the *DL inference throughput on edge devices and EdgeTPU accelerators can be significantly improved by leveraging the three approaches* proposed in this work. For the DL single-tenancy use cases, compared to the single-batch inferencing, batched inferencing can process up to 2.4× more inferences per second on devices equipped with high-performance GPUs, including `J.Nano`, `J.TX2`, and `J.Xavier`. For the AI multi-tenancy, CME on edge devices' GPU shows up to 3× improvement, and EdgeTPUs show up to 10× improvement in the DL inference throughput over the single-tenancy (with batched inferencing). We further perform CME evaluation with EdgeTPUs cluster and, with two `USB-Accelerators`, we observe that about 20% of throughput improvement can be achieved by EdgeTPU cluster over CME with a single EdgeTPU accelerator. Additionally, DMP, which leverages the collective power of GPUs and EdgeTPUs, outperformed GPU-only and EdgeTPU-only DL inference performance by a factor of 10. Finally, a detailed analysis of the factors (hardware and software) that affect the DL inference throughput at the edge is presented, thereby shedding light on areas that could be further improved to achieve high-performance DL inference at the edge. As a result, this work has the following research contributions:

— We provide a thorough characterization and quantitative analysis of the performance (i.e., latency and throughput) of various edge devices and AI accelerators when running DL tasks for image classifications. We also provide a thorough analysis of the factors that affect the DL inference throughput.
— We propose three techniques to maximize DL inference throughput on edge devices. In particular, batched inferencing is a throughput maximization approach for single DL tasks. Furthermore, CME and DMP maximize the inference throughput with AI multi-tenancy.
— With these three techniques, we discover the empirical upper bound of batch size, throughput, and model concurrency on edge devices and AI accelerators for DL inference tasks.

— We identify the performance benefits and limitations when adopting DMP to leverage heterogeneous resources on edge resources and EdgeTPUs (AI accelerators).
— We investigate the benefits and limitations of deploying DL models on edge devices with EdgeTPU clusters.

This work is based on our preliminary version [65] and we take one step in a broader and more thorough evaluation of techniques for maximizing the DL inference throughput on edge devices and AI accelerators. In particular, to the best of our knowledge, this work is the first study on evaluating DMP on heterogeneous edge resources/EdgeTPUs and characterizing the performance of EdgeTPU cluster for DL inference tasks. Therefore, the last two contributions from the above list are unique to this article.

The rest of the article is structured as follows. Section 2 provides the description of edge devices, EdgeTPUs, DL models, and DL frameworks used in this work. Section 4 describes the evaluation design and the benchmark tools used for accessing edge devices and AI accelerators. Section 5 reports evaluation results with single DL tasks (single-tenancy). Section 6 provides evaluation results when deploying multiple DL models (AI multi-tenancy). Section 7 discusses related work, and Section 8 concludes this article.

2 BACKGROUND

We describe the background of edge devices, AI accelerators, DL models, and DL frameworks used in this study.

2.1 Edge Devices and EdgeTPU Accelerators

We use three categories of devices; (1) general-purpose edge devices, (2) edge devices with GPU accelerators, and (3) EdgeTPU-based AI Accelerators. The summary of these devices is shown in Table 1.

General-purpose Edge Devices. Raspberry Pi 4 (`RPi4`) [20] and Odroid-N2 (`ODN2`) [18] are chosen for this category. `RPi4` is a small, low-cost, representative computing board for edge and IoT devices. `RPi4` is based on Broadcom BCM2711 SoC and has a quad-core ARM Cortex-A72 (1.5 GHz) and 4 GB LPDDR4 RAM. `RPi4` relies on CPUs for performing DL inference tasks, and it neither has a GPU nor specialized HW accelerators for DL processing.

`ODN2` is a computing board with 4GB LPDDR4 RAM and six CPU cores composed of a quad-core Cortex-A73 at 1.8 GHz and a dual-core Cortex-A53 at 1.9 GHz. While `ODN2` has a GPU (Mali-G52 GPU), we cannot use this GPU for DL inference tasks due to a software compatibility issue.

Edge Devices with GPU Accelerators. Three Jetson devices developed by Nvidia are chosen for this category. Jetson Nano (`J.Nano`) [12] is a small yet powerful SBC specialized in DL processing. `J.Nano` has a similar HW specification to `RPi4` except for the GPU accelerator. It has a quad-core ARM Cortex-A57 (1.5 GHz), a 128-core Nvidia Maxwell GPU, and 4 GB LPDDR4 RAM (shared by both CPU and GPU). For `J.Nano`, we use a power mode of `mode-0`, which is the default mode for maximizing the device performance.

Jetson TX2 (`J.TX2`) [15] is a high-performance edge device with six CPU cores (a dual-core Denver 2 CPU and a quad-core ARM Cortex-A57 at 2 GHz) and a 256-core Nvidia Pascal GPU for DL processing. `J.TX2` has 8 GB LPDDR4 RAM, which is shared by CPU and GPU. Among five different power modes in `J.TX2` [17], we use `mode-0` (MaxN), which enables all six cores and provides the highest frequency of both CPU (2.0 GHz) and GPU (1.3 GHz).

Jetson Xavier NX (`J.Xavier`) [16] is one of the high-end edge devices. `J.Xavier` is equipped with six cores of Carmel ARM CPUs, a Volta GPU (384 CUDA cores and 48 Tensor cores), and 8 GB of memory (shared by CPU and GPU). As `J.Xavier` has the largest number of GPU cores

Table 1. Specifications of Edge Devices and AI Accelerators

	Raspberry Pi 4 (Model B)	Odroid N2	Jetson Nano	Jetson TX2	Jetson Xavier NX	Coral Dev Board	Coral USB Accelerator
# CPU Cores	4	6	4	6	6	4	-
CPU	4-core Cortex-A72 @ 1.5GHz	4-core Cortex A73 @ 1.8GHz + 2-core Cortex-A53 @ 1.9GHz	4-core Cortex-A57 @ 1.5GHz	2-core Denver 2 64bit CPU @ 2.0 GHz + 4-core Cortex-A57 @ 2.0GHz	6-core Carmel ARM v8.2 64bit CPU	4-core Cortex-A53, Cortex-M4F	-
GPU	-	Mali-G52	128-core Nvidia Maxwell	256-core Nvidia Pascal	384-core Nvidia Volta	Integrated GC7000 Lite	-
Co-Processor	-	-	-	-	-	Google EdgeTPU (4 TOPS)	Google EdgeTPU (4 TOPS)
Memory	4 GB LPDDR4	4 GB LPDDR4	4 GB LPDDR4	8 GB LPDDR4	8 GB LPDDR4	1 GB LPDDR4	-
Power	1.8–5.3W	2–5W	5–10W	7.5–15W	10–15W	4 TOPS @2W	4 TOPS @2W
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS	Mendel Debian 10	-
Price	$55.00	$79.00	$99.00	$399.00	$399.00	$129.99	$59.99
							

Table 2. Power Modes used in Nvidia Jetson Devices

	Mode	Num. Active CPU Cores	CPU Freq.	GPU Freq.	Power Usage
`J.Nano`	`mode-0`	4	1.5 GHz	up to 1 GHz	10 W
`J.TX2`	`mode-0`	6	2.0 GHz	1.3 GHz	15 W
`J.Xavier`	`mode-2`	6	1.4 GHz	1.1 GHz	15 W

and tensor cores, it is expected to show higher DL inference performance than the other two Jetson devices. `J.Xavier` can use different power modes having distinct power consumption and resource activation. We use power `mode-2` [14], which enables all six CPU cores at 1400 MHz and all GPU cores with 1100 MHz of speed. The `mode-2` allows faster processing of CPUs/GPUs and consumes 15 W of power.

All three Jetson devices allow altering the power modes using `nvpmodel`. We use the power mode that provides the highest performance (e.g., the highest frequency of GPU and CPU) to measure the upper bound of DL inference throughput and the performance of the devices. The power configurations for the devices are shown in Table 2.

EdgeTPU-based AI Accelerators. We use two EdgeTPU-based AI accelerators; Google's Coral Dev Board (`DevBoard`) [6] and Coral USB Accelerator (`USB-Accelerator`) [7]. `DevBoard` is an SBC equipped with a quad-core Cortex-A53 CPU (1.5 GHz) and 1 GB LPDDR4 RAM, as well as an onboard **Tensor Processor Unit** (**TPU**) co-processor (accelerator), which can perform 4 **trillion operations per second** (**TOPS**) at 2 W of power consumption.

Table 3. Overview of DL Models

DL Model	Year	Input Size	Num. Layers	Billion FLOPS	# Params (Millions)	Approx. File Size (MB)	DL Framework Availability			
							PyTorch	MXNet	TF	TFLite
AlexNet [46]	2012	224 × 224	8	0.7	61	244	✓	✓	✓	✗
DenseNet-161 [41]	2016	224 × 224	161	7.9	28.7	115	✓	✓	✓	✗
Inception-V3 [66]	2015	299 × 299	48	2.9	27.2	101, 25*	✓	✓	✓	✓
MobileNet-V1 [39]	2017	224 × 224	28	1.1	4.3	17, 4.5*	✓	✓	✓	✓
MobileNet-V2 [58]	2018	224 × 224	20	0.3	3.5	14, 4*	✓	✓	✓	✓
ResNet-18 [37]	2015	224 × 224	18	1.8	11.7	46	✓	✓	✓	✗
ResNet-50 [37]	2015	224 × 224	50	4.1	25.6	102	✓	✓	✓	✗
SqueezeNet-V1 [42]	2016	224 × 224	15	0.4	1.2	5	✓	✓	✓	✗
VGG-16 [62]	2014	224 × 224	16	15.4	138.36	553	✓	✓	✓	✗

✓ denotes that the pre-trained version of the models are available for the corresponding ML framework, ✗ denotes the unavailability of model for the ML framework, * means the model size for TF Lite.

USB-Accelerator is a USB-type TPU accelerator (co-processor) specialized for ML and DL. The performance of USB-Accelerator is equivalent (4 TOPS at 2 W) to that in DevBoard. USB-Accelerator can be connected with diverse host edge devices (e.g., RPi4 and J.Nano) via USB interface, and then it can enhance DL processing. Because USB-Accelerator has only 8 MB of SRAM to store model parameters temporarily, it relies on the host device's memory system to store and load the DL models and their parameters.

Both DevBoard and USB-Accelerator are designed to support TensorFlow Lite [21] and DL models that are fully 8-bit quantized and compiled specifically for EdgeTPU architecture.

2.2 Deep Learning Models

The accuracy of DL models keeps increasing along with the complexity of model dimension and the increased number of layers. However, huge models often do not fit into the resource-constrained, low-capacity edge devices like RPi4. Therefore, based on the capacity restriction, we select nine pre-trained DL models that can be deployed on resource-constrained edge devices to perform DL inference tasks (e.g., image classification). These nine DL models are **convolutional neural network** (**CNN**) models for image classifications with having different layers, the number of FLOPS, and the number of parameters. Such differences and the overview of the nine selected models are described in Table 3.

Moreover, each model has different characteristics and advantages. For instance, AlexNet [46] utilizes multiple convolutional, max-pooling, and fully-connected layers to improve the accuracy significantly. DenseNet-161 [41] uses a connection between one layer and the subsequent layers in a feed-forward network to reduce the number of model parameters. Inception-V3 [66] scales up networks by utilizing factorized convolutions and aggressive regularization to increase the model's accuracy with minimal computation cost. MobileNet-V1/V2 models [39, 58] employ depth-wise separable convolutions to reduce the model size and the inference latency. ResNet-18/50 models [37] use deep residual nets to increase accuracy with lower complexity. SqueezeNet-V1 [42] offers AlexNet-level accuracy with 50× fewer parameters by employing three design strategies (e.g., filter replacement, decreased input channel, and downsampling rate) and fire module composed of a squeeze (1 × 1 filters) and an expand layer (a mix of 1 × 1 and 3 × 3 convolution filters). VGG-16 [62] is an extensive convolution network with over 138 million parameters that improve over AlexNet in terms of accuracy and efficiency by introducing multiple smaller (3 × 3) kernel-sized filters instead of large kernel-sized filters.

2.3 Deep Learning Frameworks

We use four, widely-used open-source DL frameworks; PyTorch [53], MxNet [29], TensorFlow [24], and TensorFlow-Lite [21], which can be deployed on resource-constrained edge devices. PyTorch,

Table 4. Source of Pre-trained Models

Framework	Source Repository	Version
PyTorch	Torchvision [1]	0.2.2.post3
MxNet	GluonCV [33]	0.10.0
TensorFlow	Kerascv [13]	0.0.40
TensorFlow-Lite	TensorFlow Hub [23]	0.2.0

MxNet, and TensorFlow are used for performing CPU- and GPU-based DL inference tasks on edge devices (e.g., J.TX2, J.Nano, ODN2, and RPi4). TensorFlow-Lite is to run DL models on EdgeTPU accelerators (DevBoard and USB-Accelerator).

MxNet [29]. MxNet is a scalable open-source DL framework from Apache Software Foundation. It has been designed to support distributed training by leveraging a distributed parameter server. The performance of the framework is claimed to scale linearly with multiple GPUs (or CPUs) as well. It also allows mixing symbolic and imperative programming models enabling better efficiency and productivity for users. MxNet currently supports Python, Java, Scala, R, Julia, Go, Clojure, Perl, MATLAB, and JavaScript.

PyTorch [53]. PyTorch, developed by Facebook, is an open-source ML framework built on top of the Torch library and designed specifically for Python. One of the most prominent features of PyTorch is providing a *NumPy-like* tensor computing support (but only for CUDA-capable Nvidia GPUs). Unlike TensorFlow (version < 2.0.0) and MxNet, PyTorch adopts a dynamic computational graph-based approach to create the computational graph at runtime. This feature enables flexibility for developers when writing and debugging DL applications.

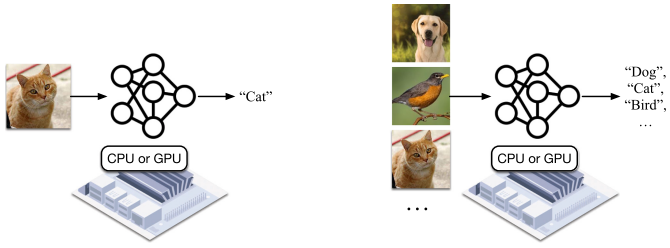
TensorFlow [24]. TensorFlow, from Google, is another open-source and widely popular ML framework. It uses a dataflow graph where nodes represent operations while the edges represent tensors. It can map the nodes of the graph across many machines in a distributed cluster and within a machine across multiple computational devices (CPUs, GPUs, or TPUs), thereby proving to be a scalable framework. Starting from version 2.0.0, TensorFlow supports *eager execution mode*, which emulates the behavior of PyTorch's dynamic computation graphs.

TensorFlow-Lite [21]. TensorFlow-Lite is a lightweight version of TensorFlow developed to support on-device DL inferencing. TensorFlow-Lite employs several techniques to optimize memory utilization, such as intermediate tensors, shared memory buffer objects, and memory offset calculation to run DL models on resource-constrained devices, including EdgeTPUs. TensorFlow-Lite has two primary parts, which are an interpreter and a converter. The interpreter runs DL models, and the converter converts TensorFlow models into TensorFlow-Lite ones.

Table 3 also shows DL frameworks' support for DL models. All nine DL models are available for PyTorch, MxNet, and TensorFlow for CPU-/GPU-based inferencing. However, Inception-V3, MobileNet-V1 and MobileNet-V2 are the only models that can be deployed on EdgeTPUs (DevBoard and USB-Accelerator) because these three models' *quantized* pre-trained versions are available for TensorFlow-Lite. Moreover, the pre-trained and compatible versions of the nine models for each of the three frameworks are readily available across various publicly accessible and maintained repositories. Table 4 lists the sources for the pre-trained models for each of the frameworks used in this study. All the models have been trained on the same ImageNet ILSVRC-2012 [56] dataset.

3 APPROACHES FOR DEEP LEARNING INFERENCE THROUGHPUT MAXIMIZATION

The goal of this study is to investigate different system approaches for maximizing the DL inference throughput on various edge devices and AI accelerators. For the DL single-tenancy use cases, we



(a) Single-Tenancy with Single Batch (b) Single-Tenancy with Batched Inferencing

Fig. 1. DL inference for AI single-tenancy on edge devices. Figure 1(a) shows single-batch inferencing. Figure 1(b) shows batched inferencing.

investigate the batching approach. For the AI multi-tenancy use cases, two approaches are studied; *CME* and *DMP*.

3.1 DL Throughput Maximization for AI Single-Tenancy on Edge Devices

For AI single-tenancy cases, where only one DL model is running on an edge device, we use *batched inferencing* (or multi-batch inferencing) to maximize the DL inference throughput. In the context of inference, *Batching* refers to enabling a single DL model to process multiple inputs simultaneously by implementing the concept of single instruction/multiple data operations. Figure 1 illustrates single batch inferencing and batched inferencing in single-tenancy on edge devices.

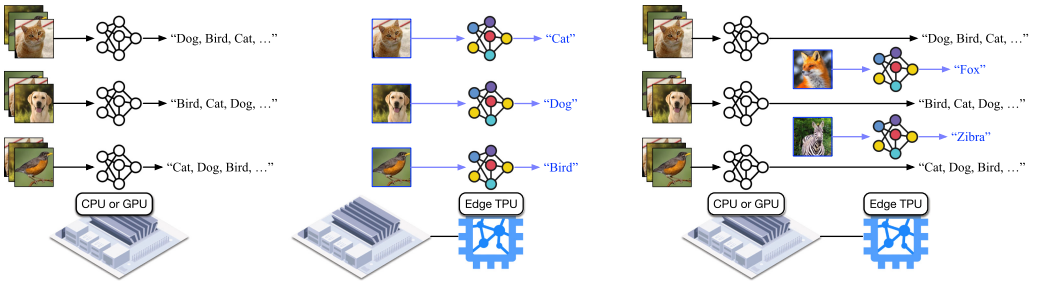
In real-world use cases, batched inferencing is beneficial as edge devices are often required to handle batches of data either from multiple IoT sensors (e.g., autonomous cars with multiple cameras) or from end devices that collect data over a period of time and send requests in batch (e.g., traffic monitoring and wearable devices). Therefore, we investigate the impact of batched inferencing on the overall throughput, specifically on GPU-enabled devices (`J.Nano`, `J.TX2`, and `J.Xavier`). The *batch size* (the number of input images to models) gradually increases in powers of two as GPUs can efficiently map virtual processing units onto physical processing units if the data to be parallelized is in the power of two [76]. Also, optimized libraries working with matrix operation can be effectively performed when processing batches of size in the order of powers of two [67].

3.2 DL Throughput Maximization for AI Multi-Tenancy on Edge Devices

We investigate techniques for maximizing DL inference throughput for AI multi-tenancy, in which multiple DL models are running simultaneously on the same edge devices (or with AI accelerators). In particular two techniques are investigated for AI multi-tenancy at the edge: (1) *CME* and (2) *DMP*.

Concurrent Model Executions (CMEs). CME leverages the idea of parallel processing and enables AI multi-tenancy by simultaneously executing multiple DL inference tasks (models) on edge devices' resources (either GPU or EdgeTPUs). Figure 2(a) and (b) illustrate the CME on edge devices and AI accelerators.

CME can provide two potential benefits to edge devices and EdgeTPUs; (1) improvement in the overall DL inference throughput and (2) ability to run multiple (often different) DL (e.g., inference) tasks. However, due to the resource-constrained nature of edge devices (e.g., limited memory sizes and the number of CPU cores), it is unclear how many DL models can be concurrently executed and which level of concurrency yields the maximum throughput. Therefore, it is important to empirically identify the upper bound of throughput improvement and the concurrency level



(a) Multi-Tenancy with CME on GPU (b) Multi-Tenancy with CME on TPU (c) Multi-Tenancy with DMP

Fig. 2. DL throughput improvement techniques for AI multi-tenancy on edge devices. Figure 2(a) and (b) illustrate CME on edge devices (using CPU or GPU) and on EdgeTPU, respectively. Figure 2(c) shows DMP on edge devices with an AI accelerator (EdgeTPU).

(the number of co-running DL models) on the devices by CME. To this end, we will measure throughput changes with different levels of concurrency. The concurrency level obtained from the last successful execution will be considered as the maximum concurrency level supported by the edge devices and EdgeTPUs. Furthermore, because CMEs provide software-level parallelism, we will also evaluate the impact of introducing the idea of *EdgeTPU-cluster* (e.g., running DL models on multiple `USB-Accelerators`) for maximizing the DL inferencing. Running concurrent models with more resources can lead to higher throughput, but other hardware bottlenecks like USB bandwidth can hinder the performance.

Dynamic Model Placements (DMPs). DMP is another approach for maximizing DL inference throughput for AI multi-tenancy with the idea of leveraging heterogeneous computing resources. DMP leverages the collective powers of edge devices and `USB-Accelerator` (EdgeTPU). In other words, it allows running multiple DL models simultaneously by placing DL models on an edge device's resource (CPU and/or GPU) and other DL models on EdgeTPUs. Figure 2(c) shows the DMP approach for AI multi-tenancy on edge devices and AI accelerators.

Furthermore, because `USB-Accelerator` relies on USB interfaces to connect edge devices, the potential benefits from DMP can be improved DL inference throughput using heterogeneous resources in both edge devices and `USB-Accelerator` and high resource utilization of both resources. However, DL inference tasks from both on-board edge resources and `USB-Accelerator` are managed by the host edge devices so that there can be a performance penalty from resource contention. Therefore, we will thoroughly investigate and analyze this penalty when employing DMP in Section 6. Moreover, similar to CME, we also study the performance impact of using the EdgeTPU cluster for DMP.

4 EVALUATION PROCESS AND BENCHMARKER DESIGN

This study investigates systematic approaches to maximize the DL inference throughput on edge devices and AI accelerators. The primary performance metric thus is the DL inference throughput. For the baseline performance (single-tenancy case), the *DL inference throughput* for single computing resource (T_{single}) is generally calculated by *DL inferences* per second, as expressed below.

$$T_{single} = \frac{The\ number\ of\ inferences}{Total\ execution\ time}. \tag{1}$$

However, the definition of *the number of inferences* varies with the type of experiment. For instance, when leveraging the single-tenancy with batched inferencing (feeding multiple input images into one DL model on a device), the number of inferences in Equation (1) is "batch size

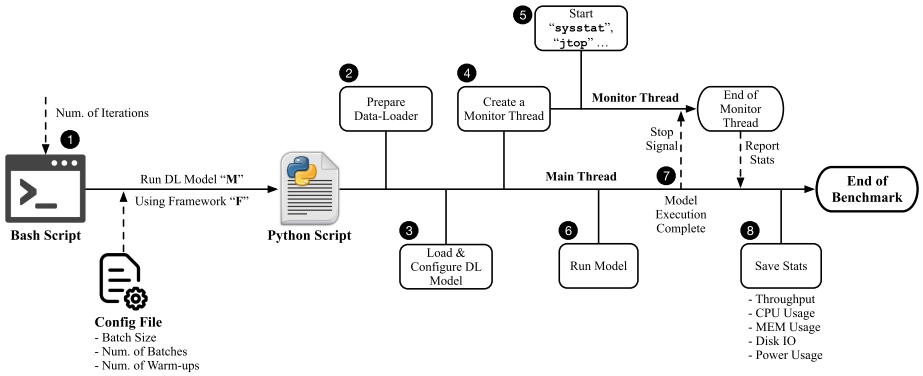


Fig. 3. Benchmark procedure.

(bs)" × "the number of batches (bc)." The equation for batched inferencing throughput (T_{batch}) is formulated below.

$$T_{batch} = \frac{bs \times bc}{Total\ execution\ time}. \tag{2}$$

On the other hand, when leveraging AI multi-tenancy with CME, the number of inferences will be calculated by "concurrency level (cc)" × "bs" × "bc". The DL inference throughput with CME (T_{cme}) is expressed in Equation (3).

$$T_{cme} = \frac{cc \times bs \times bc}{Total\ execution\ time}. \tag{3}$$

For the DMP evaluation, which uses multiple resources like both GPUs and TPUs, the total inference throughput (T_{dmp}) is the sum of the throughput of all used resources. The equation for DMP throughput is expressed as

$$T_{dmp} = \sum_{i=1}^{n} T_{cme_i}, \tag{4}$$

where i represents various computing resources for DL inference (e.g., CPU, GPU, and TPU), and n indicates the number of different resources employed by DMP.

Benchmarker Design. We develop a benchmarker that measures the DL inference throughput and collects other necessary system statistics together. We deploy it along with an image classification application on the edge devices and EdgeTPUs accelerators. The measurement procedure of the benchmarker is illustrated in Figure 3.

The benchmarker is invoked from a bash script (❶ in Figure 3) that takes parameters in a config file specific to a measurement. The config file specifies the DL model, framework, and the number of iterations to run the experiment. The config file also contains other parameters that are common across experiments, such as the number of warmup executions to perform, the input batch size, the number of batches, and resources (CPU, GPU, or EdgeTPU) used for the inference task. The bash script then runs the benchmarker (written in Python) with all these configurations. Invoking the Python interpreter using the bash script ensures that the cache constructed and maintained by the Python runtime gets cleared with each new iteration. The benchmarker then prepares a framework-specific data-loader (❷) that uses the validation dataset from `ImageNet ILSVRC-2012` to construct batches of inputs for the DL model. Next, the benchmarker initiates a DL framework as per the config file. It loads the DL model into the memory and configures the model to be executed on CPU, GPU, or EdgeTPU (❸). The next step (❹) is the warm-up phase, which ensures

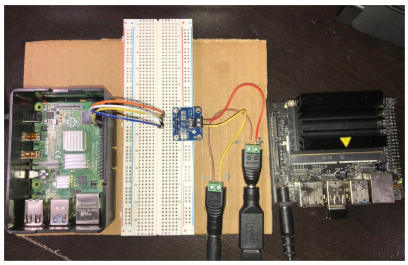


Fig. 4. Experimental setup for power measurement. Power consumption by a target edge device is being measured and transmitted to a computing board using I2C cables by INA-219 chip.

all the necessary components are loaded, and the DL framework configures suitable optimization strategies before performing the measurement. After the warm-up phase, the benchmarker creates a new background monitoring thread that captures system statistics while executing the model (❺). Simultaneously, the main thread of the benchmarker starts to run inference tasks using the DL model and the input data from the data-loader (❻). Once the pre-defined number of input batches is processed, the main thread instructs the monitoring thread to stop capturing system statistics (❼). Finally, a measurement report of throughput and system statistics is saved (❽). This entire benchmarking process is performed at least 30 times for each set of configurations to guarantee the statistical confidence of the measurement.

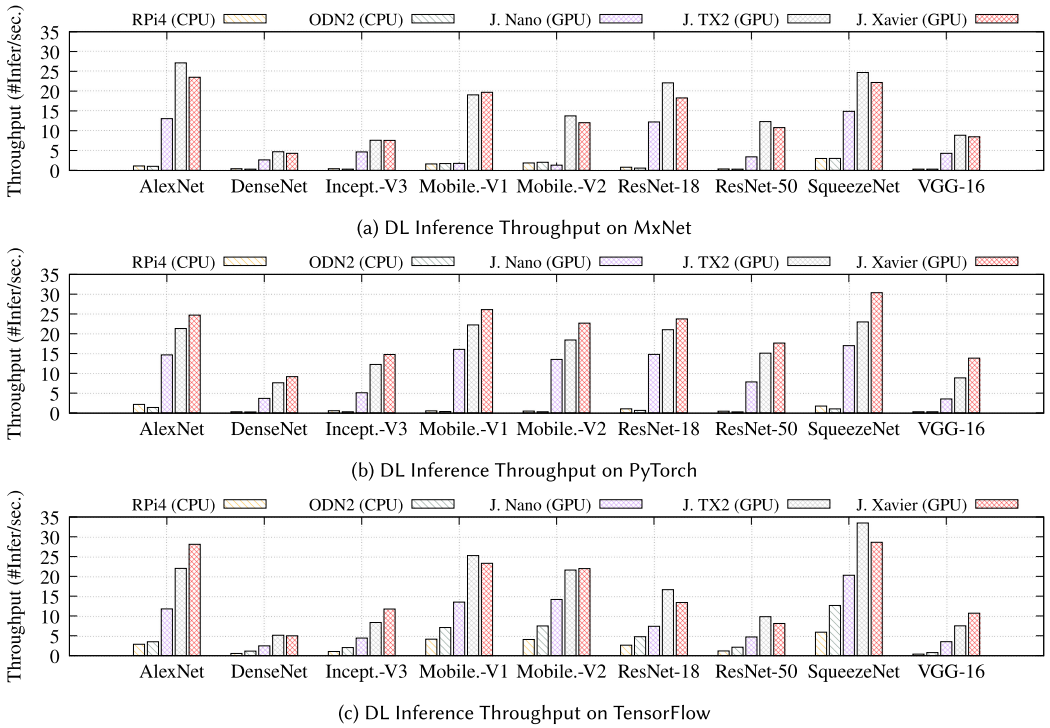
The monitoring thread is responsible for collecting diverse system statistics using `sysstat`, `usbtop`, and `jtop`. `sysstat` measures CPU and memory usage during the benchmark. `usbtop` measures USB IO bandwidth and `jtop` collects power consumption statistics. However, `jtop` is only available to measure power-related statistics for Nvidia's Jetson devices. So, we also employ `INA-219` [10], a voltage, current, and power measurement chip, for measuring the power usage of non-Nvidia devices (`RPi4` and `ODN2`). With a default resistance of 0.1 Ω, the chip allows measuring the power consumption with a current sensing range of ±3.2 A and a voltage range of 0–26 V. We use `pi-ina219` [19], a Python library to communicate with the `INA-219` chip. The experimental setup with `INA-219` is shown in Figure 4.

5 EVALUATION WITH DL SINGLE-TENANCY

We first report the evaluation results with single-tenancy on edge devices and EdgeTPU accelerators. As single-tenancy is a common use case of running AI tasks at the edge, it can be used on edge devices (with CPUs or GPUs) or EdgeTPUs. Regarding the single-tenancy on edge devices, Section 5.1 provides DL inference throughput with single-tenancy on edge devices. Moreover, as an approach for maximizing the inference throughput for single-tenancy, we evaluate the impact and performance of *batched inferencing*, where a DL model processes a batch of input images and outputs the classification results of all the images simultaneously. For single-tenancy on EdgeTPU, Section 5.2 discusses the evaluation results on EdgeTPUs. Finally, Section 5.3 thoroughly analyzes the experiment results and identifies the factors altering the DL inference throughput on edge devices and EdgeTPUs. The results reported in this section will serve as the baseline performance to evaluate throughput maximization approaches (CME and DMP) for AI multi-tenancy on edge devices.

5.1 DL Inference Throughput on Edge Devices (CPU or GPU) with Single-Tenancy

The first set of experiments measured the inference throughput of all the DL models on edge devices with a batch size of 1 i.e., single input image per model per iteration. Figure 5 reports the



(a) DL Inference Throughput on MxNet

(b) DL Inference Throughput on PyTorch

(c) DL Inference Throughput on TensorFlow

Fig. 5. DL inference throughput variations across models, edge devices, and DL frameworks with a batch size of 1.

average DL inference throughput for all the models using the three DL frameworks. Please note that the results of `J.Nano`, `J.TX2`, and `J.Xavier` are DL inference throughput on GPUs while the results from `RPi4` and `ODN2` are measured on CPUs.

DL Model Size. Figure 5 shows that the inference throughput varied significantly across different DL models. In particular, DL models with fewer parameters and floating-point operations (e.g., `AlexNet`, `MobileNet-V1`, and `SqueezeNet-V1`) showed higher throughput than DL models with a higher number of operations (e.g., `DenseNet-161`, `Inception-V3`, and `VGG-16`). We observed that the throughput differences between smaller and larger models were more prominent in CPU-based devices (`RPi4` and `ODN2`). The inference throughput with `SqueezeNet-V1` on `RPi4` and `ODN2` reported 10×–22× higher than the throughput with `DenseNet-161`. Both `RPi4` and `ODN2` had a throughput of less than 1 (one inference per sec.) for all the DL frameworks when running `DenseNet-161`, amplifying the difference over `SqueezeNet-V1`'s results on those devices. On the other hand, `SqueezeNet-V1`'s throughput on GPU-equipped edge devices (e.g., `J.Nano`, `J.TX2`, and `J.Xavier`) was only 3×–8× better than `DenseNet-161`'s throughput. These results indeed confirmed the benefits of using GPUs over CPUs for DL inferencing. Without the support for data parallelism in CPUs, very deep models (e.g., `DenseNet-161` and `Inception-V3`) that involved extensive floating-point operations, the CPU-only inferencing had to spend a significant amount of time on processing the models, hence significantly decreasing the inference throughput.

GPU vs. CPU. Figure 5 also confirms that, for single-batch inference, the GPU-based devices' DL inference throughput significantly outperformed the CPU-based devices' inference throughput.

The edge devices with GPUs (J.Nano, J.TX2, and J.Xavier) processed 4×–80× more inference requests than the devices without a GPU (RPi4 and ODN2) for all the models across all three frameworks. The advantage of using GPU is dominantly observed when DL inference using PyTorch. On average, J.Nano, J.TX2, and J.Xavier showed 17×, 30×, and 38× higher throughput than RPi4, respectively, and the DL inference throughput results on these devices were 38×, 62×, and 75× higher than ODN2. The results also showed the performance differences among the three GPU-based edge devices. Regarding the GPU performance, J.Nano has the least powerful GPU, which was clearly observed in the results. There was no clear winner between J.Xavier and J.TX2. But, J.TX2's GPU frequency (1.3 GHz) is slightly higher than that of J.Xavier's (1.1 GHz). This can be attributed as the reason why J.TX2's results were better, particularly when performing inferences on MxNet and TensorFlow. For PyTorch, however, we observed that J.Xavier outperformed J.TX2. This result was partly because of the difference in DL framework capability and partly due to the different GPU core numbers. PyTorch has better parallelization support than the other two frameworks. Also, J.Xavier has more GPU cores (384) than J.TX2 (256 GPU cores). As a result, PyTorch had better support in parallelizing its dynamic computation graph on more numbers of cores than the other two frameworks, hence producing higher DL inference throughputs.

DL Frameworks. Among the three DL frameworks, PyTorch showed the highest throughput on GPUs. On average, the throughput of DL models with PyTorch was 31% and 26% higher than MxNet and TensorFlow, respectively. PyTorch's superiority on GPUs was because of Torch library designed to make tensor operations on GPU faster and more efficient. On the other hand, TensorFlow significantly outperformed the other two on CPUs. The average throughput across all the models on CPUs using TensorFlow was about 5× the results from MxNet and 10× from PyTorch. This result was due to TensorFlow's design to support mapping nodes (from computational graph) across multicore CPUs [24]. Therefore, TensorFlow could enable faster computation, processing more DL inferences on CPUs than the other two frameworks.

MxNet, on all the devices, was the least performing framework. We observed that two MobilenNet models' throughput with MxNet on J.Nano was exceptionally lower than the throughput with other frameworks. For example, both models processed less than 5 inferences per second, which was even 4× less than the throughput of the same models with the other two frameworks. Such lower throughput is because of two reasons. First, when MxNet leveraged GPUs, the framework first performed an auto-tunning process that used Nvidia's cuDNN library to automatically tune convolution layers. In other words, MxNet tried to find the best performing convolution algorithm in its first run, enabling subsequent model executions to run faster. Unfortunately, this process is a highly memory-intensive operation, and J.Nano's 4 GB memory is not large enough to complete this process, frequently resulting in out-of-memory errors. Because of the above reason, the measurement had to be performed with disabling auto-tune, meaning that MxNet relied on sub-optimal convolution algorithm. Second, the two MobileNet models performed frequent small-size memory-bound element-wise operations, such as ReLU [39]. Without optimization strategies like operator fusion [28] enabled, the processing time of such models increased steeply. Such performance tuning strategies worked only well on edge devices with larger memory size but not on edge devices with smaller memory sizes like J.Nano.

Impact of Batched Inferencing. As discussed in Section 3.1, batched inferencing is the approach to maximize the throughput for single-tenancy. Figure 6 reports the DL inference throughput of *batched inferencing* with increasing batch sizes, on the three ML frameworks. Please note that Figure 6 includes the results of five models on four devices due to the page limitation, and other omitted results have similar patterns. We observed that there was a significant throughput improvement with increasing batch size for GPU-enabled devices. On average, a batch size of 32 showed 240% higher DL inference throughput. The impact of batching on J.Xavier,

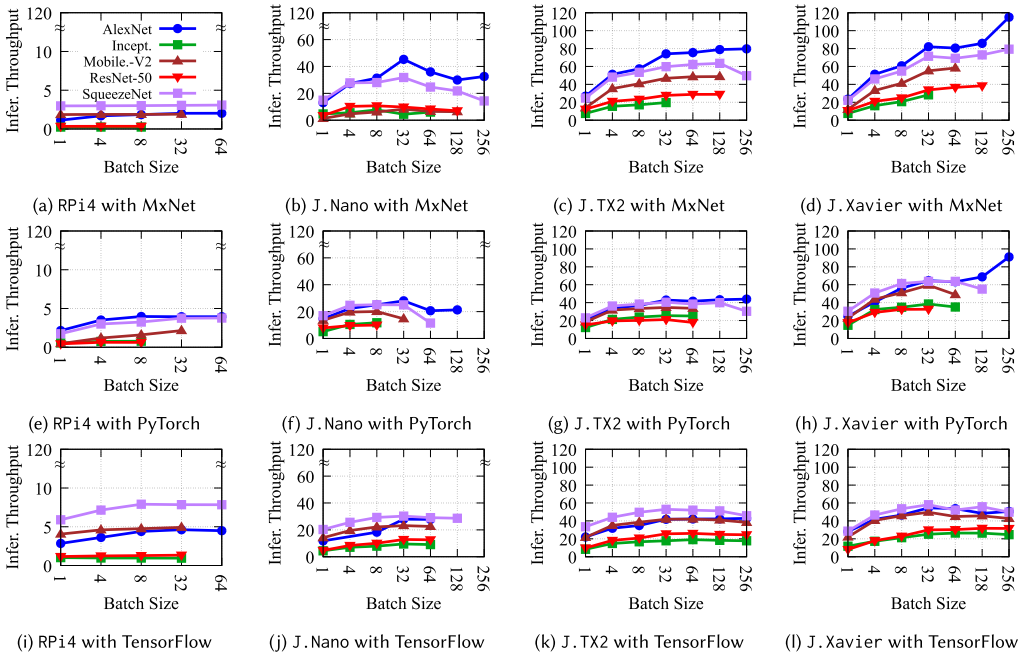


Fig. 6. Throughput variation across DL models, edge devices, and DL framework with different batch sizes.

with more GPU cores, was higher than in J.TX2 and J.Nano. Specifically, J.Xavier's results were 25% and 42% greater than J.TX2 and J.Nano, respectively. Another interesting observation is that, while a larger batch size appeared to increase the throughput, large batch sizes (> 128) may not always improve the throughput. For example, when using AlexNet, the batch sizes of 4, 8, or 32 often showed higher throughput than the throughput with the batch sizes of 128/256 on J.Nano (Figure 6(b) and (f)).

Another observation is that the inference throughput did not always increase as batch size increased. While inferencing without batching, there is an interval between computations in two activation layers in the model. With batched inferencing, the model can calculate other images' activation layers in the same batch and store the results in the memory for the next layer. The batch size is limited by an edge device's memory capacity [63]. If the memory is sufficient enough to store all the activations, that batch can be processed directly [32]. However, when the memory size is insufficient, batching would trigger the edge device's memory saving mechanism, such as swap space. The data in the memory will then be moved to the storage (SD card in edge devices). This mechanism can slow down the inference speed, hence decreasing inference throughput. Therefore, employing the right (or optimal) batch size is a critical factor for maximizing the DL inference throughput on edge devices.

5.2 DL Inference Throughput on EdgeTPU with Single-Tenancy

EdgeTPUs are designed to support faster processing of tensors (one of the primary components of CNNs), which in turn, can boost the DL inference throughput. Note that the USB-Accelerator requires a host device to run because it is a USB-type portable accelerator. We used all five edge devices with a USB-Accelerator to measure the throughput from the USB-Accelerator. Inception-V3, MobileNet-V1, and MobileNet-V2 were used for this evaluation.

Table 5. DL Inference Throughput of the Three Quantized DL Models on EdgeTPUs

Model	Host Device + Edge TPU	Avg. Infer. Through.	Std. Dev.
Inception -V3	RPi4 + USB-Acc	12.35	0.35
	ODN2 + USB-Acc	15.59	0.47
	J.Nano + USB-Acc	16.42	0.34
	J.TX2 + USB-Acc	18.54	0.48
	J.Xavier + USB-Acc	17.28	0.38
	DevBoard Only	13.26	0.19
MobileNet -V1	RPi4 + USB-Acc	54.65	4.03
	ODN2 + USB-Acc	58.84	6.73
	J.Nano + USB-Acc	63.60	5.58
	J.TX2 + USB-Acc	64.65	5.45
	J.Xavier + USB-Acc	64.01	2.73
	DevBoard Only	59.02	2.48
MobileNet -V2	RPi4 + USB-Acc	55.79	4.15
	ODN2 + USB-Acc	59.70	5.78
	J.Nano + USB-Acc	66.61	4.23
	J.TX2 + USB-Acc	64.01	6.57
	J.Xavier + USB-Acc	64.69	2.41
	DevBoard Only	60.67	5.23

Table 5 reports the DL throughput fluctuations on EdgeTPUs connected with different host devices, and the throughputs fluctuated across the different host devices. For example, when USB-Accelerator was connected with Jetson devices, its inference throughput could reach as high as 65 inferences per second for MobileNet-V1/V2 and 16 inferences per second for Inception-V3; however, the throughput decreases when using RPi4 and ODN2 as the host device. Several factors could result in such throughput fluctuations. Memory bandwidth on the (host) edge devices is one of the factors for such fluctuations. For example, the latency when swapping in/out of a DL model and its parameters between the host devices and USB-Accelerator rely on the memory bandwidth. Furthermore, USB IO can also change the DL inference throughput. Such factors will be analyzed in Section 5.3.

The benefits of using EdgeTPUs are confirmed by comparing the inference throughput against other edge devices' (CPU- and GPU-based) throughput results. As shown in Figure 7, USB-Accelerator and DevBoard showed significant improvement in DL inference throughput of all three models compared to the edge devices (RPi4 and ODN2) relying on CPU resources. On average, EdgeTPU processed 8.5× (DevBoard) and 9× (USB-Accelerator) more image classification tasks than CPU-based inferencing.

Compared to GPU-based inferencing on Jetson devices, EdgeTPUs outperformed J.Nano for all three models, and USB-Accelerator and DevBoard were able to manage similar performance

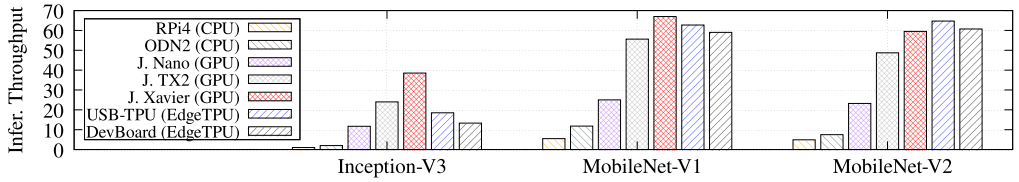


Fig. 7. Comparison of inference throughput in CPU, GPU, and EdgeTPU. The throughput results of CPU- and GPU-based inferences are the maximum throughput results of those devices amongst all three frameworks. Please note that `USB-Accelerator`'s throughput in this graph is the maximum throughput from the results reported in Table 5.

to `J.TX2` and `J.Xavier` when processing `MobileNet-V1/V2` models. However, both accelerators appeared to have lower performance for processing the Inception-V3 model. In particular, the throughput with `Inception-V3` reached only 35% (`DevBoard`) and 48% (`USB-Accelerator`) of that of `J.Xavier`. Our further analysis revealed that lower throughput with `Inception-V3` was related to the model size. Large models like `Inception-V3` have much more parameters (compared to smaller models like `MobileNet-V1/V2`) stored in the main memory, and the parameters have to be constantly swapped between the host memory and EdgeTPU. Unfortunately, `DevBoard` we used in this work has only 1 GB of main memory, which is not large enough to store all the parameters of `Inception-V3`. `USB-Accelerator` has only 8 MB of cache memory, requiring frequent parameter swap operations between the host edge devices and EdgeTPU, resulting in a considerable decrease in the DL inference throughput.

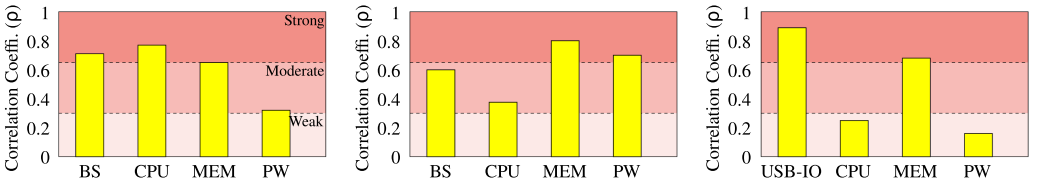
Moreover, we observed that `DevBoard` showed slightly lower throughput over `USB-Accelerator` even though both have the same co-processor. This was mainly due to the overhead associated with the management in process, memory, and other operating system-related tasks, which do not apply to `USB-Accelerator`. (Host edge devices performed such management tasks for `USB-Accelerator`).

5.3 Analysis of Factors for Influencing DL Inference Throughput with Single-tenancy

In this subsection, we further discuss the analysis of factors that can affect DL inference throughput on edge devices and EdgeTPUs when employing DL single-tenancy.

Correlation Analysis Between System Factors and DL Inference Throughput. We first performed correlation analysis to investigate the factors that change the DL inference throughput on edge devices and EdgeTPUs. The correlation analysis was performed by calculating the Pearson Correlation Coefficient, $\frac{cov(x,y)}{\sigma_x \sigma_y} = \frac{\sum_i^n (x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_i^n (x_i-\overline{x})^2 (y_i-\overline{y})^2}}$, between measured throughput results and resource usage statistics [26]. This coefficient represents the linear relationship between two variables, ranging from −1 to 1. Please note that the coefficient of 1 indicates an ideal positive correlation, negative values mean reverse correlation, and 0 means there is no correlation between two variables.

Figure 8 shows the correlated factors for the DL inference throughput when using CPUs, GPUs, and EdgeTPUs. For the CPU-based inferences on `RPi4`, `ODN2` (shown in Figure 8(a)), the batch size, CPU, and memory strongly correlated with the inference throughput changes. This is because the CPU is the main computing resource for performing the DL tasks, and memory resources are used to load and store the DL models. The inference tasks with larger batch sizes naturally increase the input data for processing so that an increase in the batch sizes can improve the throughput until the limit of device resources. Table 6 summarizes the impact on throughput with increasing batch size and the corresponding increment in CPU usage on `RPi4`. Heavier models like `DenseNet-161`



(a) CPU-based Inference (b) GPU-based Inference (c) TPU-based Inference

Fig. 8. Correlated factors that change the inference throughput. (BS: Batch Size, CPU: CPU usage, MEM: memory usage, PW: Power consumption, USB-IO: USB IO bandwidth usage).

Table 6. Change in CPU Usage and Inference Throughput (on TensorFlow) with Varying Batch Sizes in RPi4

Model	Batch Size	Avg. Throughput	Avg. CPU Usage(%)
AlexNet	1	2.85	53.9
	32	4.63	100.0
DenseNet-161	1	0.53	76.5
	32	0.56	100.0
Inception-V3	1	1.02	81.0
	32	0.95	100.0
MobileNet-V1	1	4.14	59.2
	32	5.51	93.0
MobileNet-V2	1	4.05	60.9
	32	4.92	87.4
ResNet-18	1	2.61	73.1
	32	2.90	98.3
ResNet-50	1	1.16	72.8
	32	1.34	98.1
SqueezeNet-V1	1	5.89	53.3
	32	7.86	85.8

and Inception-V3 did not show significant changes in throughput with increased batch size on CPUs because of the high processing demands, as highlighted by the 100% CPU usage. However, for the lighter models (AlexNet, MobileNet-V1/V2, SqueezeNet-V1), on average, there is a gain of 40% in throughput with a nearly 70% increase in CPU usage with increasing batch size.

For the GPU-based inference tasks on J.Nano, J.TX2, and J.Xavier (shown in Figure 8(b)), memory, power consumption, and inference batch sizes had a relatively stronger correlation with the DL inference throughput. Specifically, the power consumption showed a strong correlation with the throughput as the GPU module in edge devices consumed more power than typical CPUs in edge devices. We further observed that, on average, a 15–20% increase in power usage corresponded to a 90–100% gain in throughput. Regarding the batch sizes, as we discussed in Section 5.1, increasing batch size could significantly change the DL inference throughput, and it was clearly observed with the correlation analysis. On the other hand, CPU, as expected, showed a relatively weaker correlation with the DL inference throughput as CPU is primarily used for managing the device and non-DL tasks rather than performing GPU-based inference tasks.

For the inference tasks on EdgeTPU accelerators (especially USB-Accelerator), as shown in Figure 8(c), the USB bandwidth between a host edge device and the USB-Accelerator and memory usage on host edge devices had a strong correlation with the inference throughput. Both memory and USB IO were closely related to each other for executing DL models on USB-Accelerator. Because USB-Accelerator does not have main memory (RAM),[1] it relies on the host device's memory system to store models and uses context switching to swap models/parameters between the host device's RAM and EdgeTPU to perform DL inference tasks. Therefore, low USB IO bandwidth

[1]USB-Accelerator only have 8MB of cache memory (SRAM).

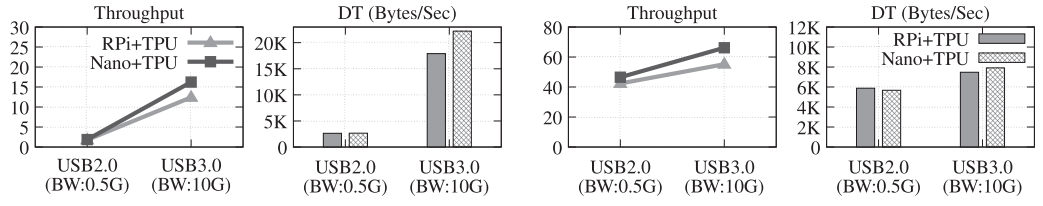


Fig. 9. Difference in DL inference throughput and data transfer with USB 2.0 and 3.0 interfaces. (DT: Data Transfer Rate).

between the host device and USB-Accelerator limited data rates for switching models/parameters so that the throughput could be decreased.

Impact of USB Bandwidth on USB-Accelerator's DL Inference Throughput. To investigate the impact of the USB IO bandwidth, we measured the inference throughput changes on USB-Accelerator by connecting it to two edge devices (RPi4 and J.Nano) with different USB interfaces; (a) USB 2.0 with up to 0.5 GB of bandwidth, (b) USB 3.0 with up to 10 GB of bandwidth. Figure 9 shows that USB's IO bandwidth could considerably change the inference throughput of EdgeTPUs. With larger IO bandwidth, RPi4 achieved 1.3× (MobileNet-V2) and 7× (Inception-V3) higher throughput when moving from USB 2.0 to 3.0. J.Nano also showed 1.4× (MobileNet-V2) and 8.7× (Inception-V3) higher throughput with USB 3.0 than that with USB 2.0. Larger USB IO bandwidth facilitated faster swapping of model parameters and input data between the host device and USB-Accelerator, enabling faster DL inferences.

6 EVALUATION WITH AI MULTI-TENANCY

CME and DMP are two approaches to maximizing the DL inference throughput with AI multi-tenancy. This section reports our measurement results with CME (Section 6.1) and DMP (Section 6.2) and discusses the benefits and limitations of both approaches.

6.1 Concurrent Model Executions (CME)

We first describe the evaluation procedure of CME and then report CME measurement results on GPUs on edge devices and EdgeTPUs. Finally, we will discuss CME results with a cluster of EdgeTPUs. In this evaluation, we seek answers to the following research questions:

(1) What is the maximum DL inference throughput of the edge devices and EdgeTPUs with CME?
(2) What is the maximum concurrency level on the edge devices and EdgeTPUs with CME?
(3) What is the concurrency level on edge devices and EdgeTPUs to maximize DL inference throughput?
(4) What are the benefits and limitations of leveraging multiple EdgeTPUs (a.k.a cluster of EdgeTPUs) with CME for maximizing the DL inference throughput?

For the rest of this study, we only use three DL models, Inception-V3, MobileNet-V1, and MobileNet-V2, because pre-trained versions of these models are *officially* available for all the edge devices, including EdgeTPUs. Furthermore, we also exclude TensorFlow from this CME evaluation due to software issues with kerascv [13] and tf.Graph [22] APIs, which do not fully support concurrent executions (e.g., not thread-safe). Finally, regarding the throughput calculation with CME, we use Equation (3) to calculate DL inference throughput in Section 4.

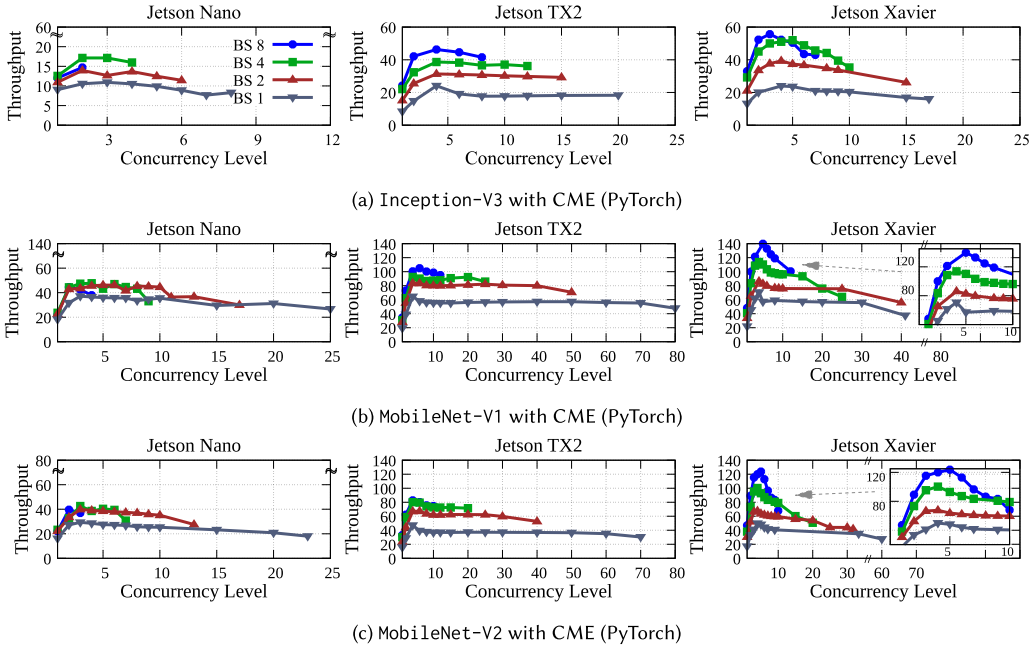


Fig. 10. Concurrency measurement results on J.Nano, J.TX2, and J.Xavier GPUs with PyTorch (BS: Batch Size).

Evaluation Procedure. Based on the measurement results from single-tenancy cases (Section 5), we gradually increase the number of co-running DL models ("*concurrency level*") on the devices and EdgeTPU to find the maximum level of concurrency and throughput improvement with CME. This process continues until the benchmarker fails to run for one of the following reasons. (1) the memory is fully saturated or (2) the device can no longer create more DL tasks. The concurrency level obtained from the last successful execution is considered as the maximum concurrency level supported by the edge devices and EdgeTPUs. In this measurement, we only report the results with leveraging CME on GPUs (J.Nano, J.TX2, and J.Xavier) and EdgeTPUs (DevBoard and USB-Accelerator). We omit the measurement results from CPU-based inferencing (e.g., RPi4 and ODN2) because, while we could find some benefits of CME on CPUs, e.g., six concurrent models could be executed on CPUs of RPi4 and ODN2, the throughput benefits were marginal. The measured throughput results were exceptionally lower than the results with CME on either GPUs or EdgeTPUs.

The benchmarking process described in Figure 3 (Section 4) is tweaked such that instead of running a model in the main thread (❻ in Figure 3), new threads are created to run models concurrently (i.e., separate copies of the model are created for each thread). The main thread then waits for all the models to finish execution and finally terminates the script, followed by steps similar to the previous workflow.

6.1.1 CME Evaluation Results on GPU in Edge Devices. The next evaluation is to measure the DL inference throughput of GPUs with CME and increasing concurrency levels using PyTorch (Figure 10) and MxNet (Figure 11), respectively, on J.Nano, J.TX2, and J.Xavier. As shown in the results, CME can significantly improve the overall DL inference throughput on GPUs by executing multiple DL inference tasks simultaneously. Similar to the results of batched inferencing, CME

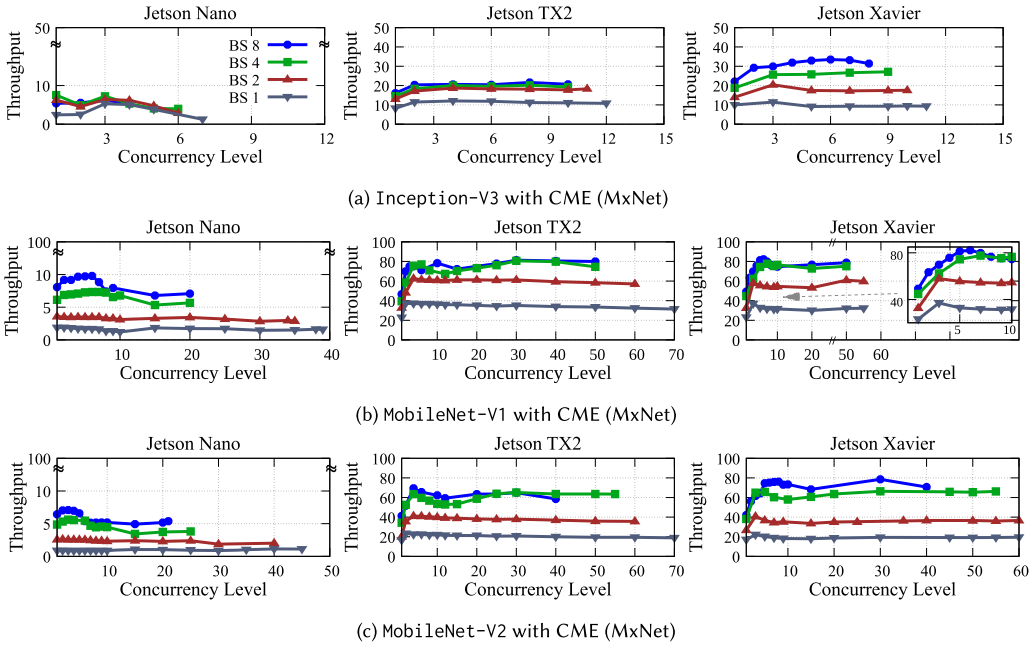
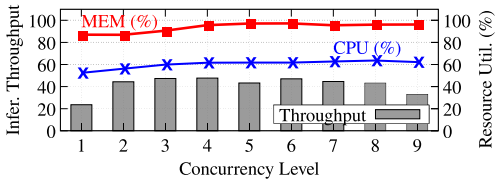


Fig. 11. Concurrency measurement results on J.Nano, J.TX2, and J.Xavier GPUs with PyTorch (BS: Batch Size).

of lighter models like MobileNet-V1/V2 (Figures 10(b)–(c) and 11(b)–(c)) yielded a higher gain in throughput while heavier models, e.g., Inception-V3 (Figures 10(a) and 11(a)), showed minor improvement. In particular, compared to single-tenancy with bached inferencing, CME resulted in 1.4×–2×, 1.8×–2.7×, 1.7×–3.0× increase in overall throughput on J.Nano, J.TX2, J.Xavier, respectively, across all the three models with PyTorch (Figure 10). The results with MxNet (Figure 11) were less impressive in that relatively lower throughput improvements were observed; 1.3×–1.5× on J.TX2 and 1.5×–1.8× on J.Xavier. J.Nano's results with MxNet were particularly low. The worst case we observed was 13% lower throughput than single-tenancy. The J.Nano's low throughput was because the J.Nano's experiments were performed by disabling MxNet's cuDNN auto-tune [9] process as J.Nano's memory size (4 GB) was not sufficient enough to perform such memory intensive optimization. Enabling or disabling auto-tune option could significantly impact DL inference throughput. If auto-tune was enabled, MxNet first ran a performance test to seek the best convolutional algorithm, and the selected algorithm was then used for further inference tasks.

Input batch size and level of concurrency complemented the performance gain as both the approaches rely on running multiple inferences simultaneously. However, due to memory and CPU usage constraints on edge devices, we could not indefinitely increase both to maximize performance. In our study, we observed that 5 to 6 of the concurrent level with a batch size of 8 resulted in the maximum empirical throughput improvement. After that, we observed increasing either of the two parameters (concurrency level and batch size) resulted in lower performance.

The level of concurrency was directly related to the size of the model and the available memory in edge devices. J.Nano could run 8 (Inception-V3) to 25 (MobileNet-V1/V2) models concurrently on GPU while J.TX2 and J.Xavier were able to run approximately 25 (Inception-V3) to 80 (MobileNet-V1/V2) models on their GPUs simultaneously when using a batch size of 1.



(a) MobileNet-V1 (J.Nano) (b) MobileNet-V2 (J.TX2)

Fig. 12. Resource utilization and inference throughput changes with CME (PyTorch). J.Nano uses a batch size of 4, and J.TX2 employs a batch size of 8.

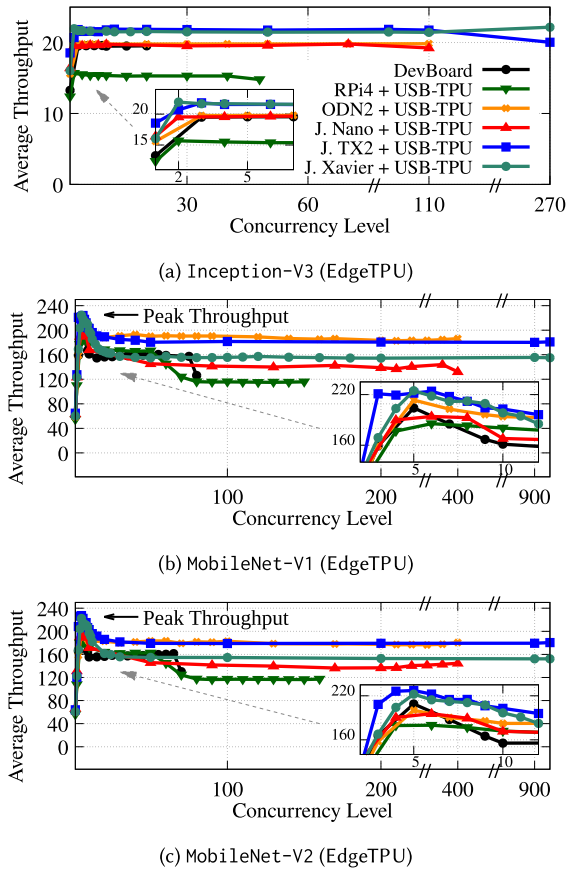
With increased batch sizes, the level of concurrency decreased as lesser memory became available. Figure 12 shows that the maximum throughput is highly correlated with memory utilization. As shown in the figures, after reaching the maximum throughput, the throughput was either decreased or stabilized with high memory utilization (or memory saturation). It is worth noting that the high correlation between memory utilization and throughput increase is consistent with our observation reported in Figure 8.

6.1.2 CME Evaluation Results on EdgeTPUs. The second evaluation for CME was to measure the DL inference throughput on EdgeTPUs, and Figure 13 shows CME results on TPUs (both DevBoard and USB-Accelerator). The result includes all the combinations of edge devices and USB-Accelerator as well as DevBoard.

Like the previous GPU results, CME on EdgeTPUs also increased throughput over the single-tenancy cases. For Inception-V3 (Figure 13(a)), DevBoard showed 1.3× higher throughput, and USB-Accelerator had 1.25× improved throughput over single-tenancy cases. For both MobileNet-V1 (Figure 13(b)) and MobileNet-V2 (Figure 13(c)), EdgeTPUs showed 3.3× higher throughput over the single-tenancy cases.

We found two interesting observations about the throughput improvement. One is that CME's throughput increase with Inception-V3 (1.3×) was much smaller than MobileNet-V1/V2 (3.3×). The other is that MobileNet-V1/V2 reached the maximum throughput with lower concurrency levels, and the throughput was decreased and stabilized with higher concurrency levels. Our further analysis revealed that the above two issues were related to the model size and EdgeTPU's small SRAM size (8 MB) used to cache the DL model's parameters. In particular, a smaller throughput increase with Inception-V3 was because 25 MB of Inception-V3 model size could not be fully loaded into the EdgeTPUs' cache (SRAM), and thus, model parameter swapping operations between the EdgeTPU's cache and the edge devices' memory were continuously being performed. Therefore, the increased concurrency level was not always increasing the inference throughput due to the high overhead in the model parameter swaps. On the other hand, if the model size was small, e.g., 4 MB of MobileNet-V1/V2, the model could be fully loaded into EdgeTPUs' cache and did not require frequent operations of model parameter swapping. As a result, EdgeTPUs with CME could significantly improve the DL inference throughput with low USB IO overhead.

The second observation (Figure 13(b) and 13(c))–*maximum throughput gain of* MobileNet-V1/V2 *was achieved with a lower concurrency level*–was because the EdgeTPU cache was enough to load even multiple smaller models simultaneously. While EdgeTPU could leverage only one model at a time, other loaded models in its cache could obtain data from the host device's memory, hence minimizing the delay when switching models in EdgeTPUs. On the other hand, if the concurrency level was high, frequent model swaps needed to be frequently performed in EdgeTPU's cache, resulting in increased data transfer between EdgeTPU and the host edge device's memory.



(a) `Inception-V3` (EdgeTPU)

(b) `MobileNet-V1` (EdgeTPU)

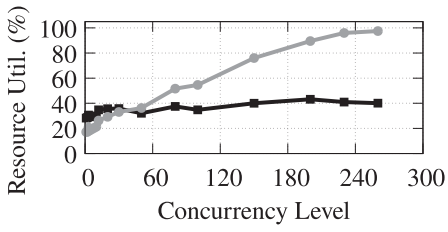
(c) `MobileNet-V2` (EdgeTPU)

Fig. 13. Results of CME measurement on EdgeTPU.

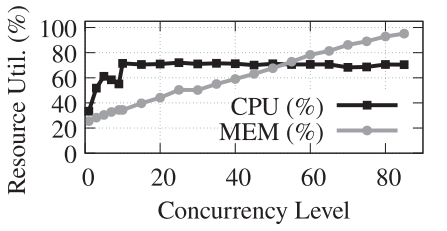
As a result, USB IO was quickly saturated, which in turn, decreased throughput. This was why both `MobileNet-V1` and `MobileNet-V2` reached the maximum throughput with a low concurrency level, and throughput could be decreased or stabilized with higher concurrency levels. This analysis suggests that, when using CME on EdgeTPU, model size and concurrency level should be carefully determined to maximize the throughput. Moreover, model compression techniques [27], e.g., quantization and parameter pruning, should be considered for optimizing model size for EdgeTPUs.

Furthermore, all three models reported much higher concurrency levels on EdgeTPUs than on GPUs. For example, `DevBoard` supported the concurrency level of 20 for `Inception-V3` and 80–85 for both `MobileNet-V1/V2` models. `USB-Accelerator` reached various maximum concurrency levels, and `USB-Accelerators`' concurrency levels vary considerably across different host edge devices. For example, for `Inception-V3`, the maximum concurrency level from `USB-Accelerator` with `RPi4` was 48, but the maximum concurrency level when it uses `J.TX2` or `J.Xavier` as the host device reached 270. For `MobileNet-V1/V2`, the maximum concurrency level from `USB-Accelerator` with `RPi4` was about 160, but it could be 1,100 when leveraging `J.TX2` or `J.Xavier` as the host device.

Regarding the varying concurrency levels, Figure 14 shows resource utilization changes with different concurrency levels measured on `USB-Accelerator` with `J.TX2` and `DevBoard`. We observed that memory utilization increased as the concurrency level increased. The maximum concurrency



(a) Inception-V3 (J.TX2 + USB-Accelerator)



(b) MobileNet-V2 (DevB)

Fig. 14. Resource utilization changes with increased concurrency level (EdgeTPUs).

level was determined when the memory utilization reached close to 100%, indicating that memory size and bandwidth often limited the supported concurrency level DL models when enabling CME on the USB-Accelerator.

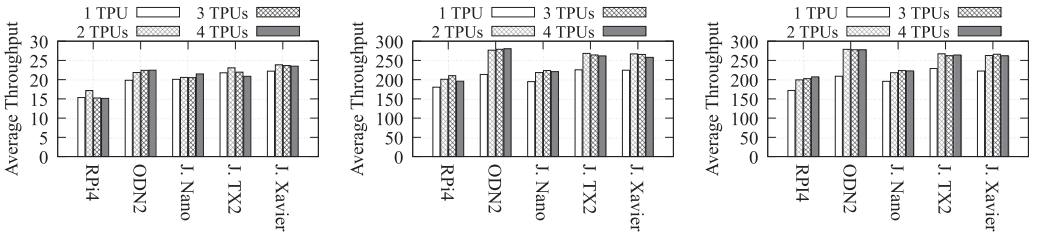
We also measured the changes in the host edge device's memory and USB bandwidth when throughput changes. Our observation is that the memory utilization kept increasing as the inference throughput degraded after the peak throughput. However, the USB bandwidth kept stable after reaching the peak throughput.

Finally, compared with CME on GPUs, the maximum throughput of EdgeTPUs was nearly 230 inferences per second when running concurrent MobileNet-V1/V2. And this maximum throughput was almost 2× higher throughput than CME on GPUs. However, this was not the case for larger models like Inception-V3. The maximum achievable throughput with Inception-V3 using EdgeTPUs was nearly half the value when using GPUs. These observations suggest that careful consideration of model (size) and device (GPU or EdgeTPU) is critical to maximizing the overall throughput.

6.1.3 CME Evaluation Results on EdgeTPU Cluster. As discussed in the previous subsection, USB-Accelerator could significantly improve DL inference throughput with CME. To maximize the performance of USB-Accelerator, we performed further evaluations using multiple USB-Accelerator (called *EdgeTPU cluster*) with CME. The idea of EdgeTPU cluster is to connect *more than one* USB-Accelerators with a host edge device and employ CME on each USB-Accelerator (shown in Figure 15). To this end, we used Edge TPU Python API [8] to load models in specific devices to ensure an equal number of models were running on all the USB-Accelerators.

We started the experiment by running each of three quantized models (Inception-V3, MobileNet-V1, and MobileNet-V2) on two USB-Accelerators simultaneously on each device. Similar to the previous CME evaluations, the experiment was repeated for multiple levels of concurrency to find one that produced the maximum throughput. Once we found the maximum throughput with two accelerators, we gradually increased the number of USB-Accelerator (as well as

Fig. 15. EdgeTPU-cluster composed of four EdgeTPUs (USB-Accelerator) connected with J.Xavier.



(a) Inception-V3 (b) MobileNet-V1 (c) MobileNet-V2

Fig. 16. DL inference throughput variation with multiple USB-Accelerators (EdgeTPU cluster).

the concurrency level) and repeated this procedure until the throughput reached the maximum with four USB-Accelerators with CME. The measurement results are reported in Figure 16. In general, EdgeTPU-cluster could increase the DL inference throughput over a single accelerator, but the throughput differences among two, three, and four USB-Accelerator were marginal. In particular, when using EdgeTPU-cluster composed of two accelerators, on average, we observed a 15–30% increase in throughput over a single accelerator for MobileNet-V1 (Figure 16(b)) and MobileNet-V2 (Figure 16(c)) across all devices. Such throughput increase was because two models could simultaneously be processed in two accelerators while, for one accelerator, the inference task had to wait for the single EdgeTPU to finish the current task before moving on to the next task. Moreover, with two USB-Accelerators, the throughput improvement could not reach 2× as desired because the USB ports in all devices have an *internal shared hub* for the USB interface. This shared USB hub caused a certain delay (due to its serial processing characteristics) in data transfer, hence increasing the overall latency. For Inception-V3 with two accelerators (Figure 16(a)), the throughput improvement was smaller than the improvement observed with MobileNet-V1/V2. As mentioned earlier, Inception-V3 was too large to fit in a USB-Accelerator and thus requires constant swapping of model parameters with the host device. This, along with the serial nature of USB ports due to the presence of the shared hub, limited the parallelism that could have been achieved from multiple EdgeTPU-clusters with USB-Accelerators.

The EdgeTPU-cluster with three or four USB-Accelerators did not show meaningful throughput improvement over the cluster with two accelerators. Such limited throughput improvement was also due to the internal shared hub for the USB interface, which limited the total USB bandwidth while the device had the increased number of accelerators. Figure 17 shows the overall bandwidth as well as the bandwidth consumed by each accelerator with J.TX2 when running MobileNet-V2. It is observed that the bandwidth available to each accelerator decreases with every addition of a USB-Accelerator. This reduced bandwidth and data transfer rate directly degrade

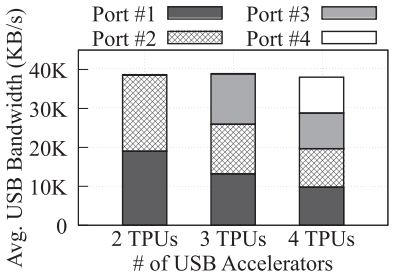


Fig. 17. Total USB bandwidth usage and bandwidth consumed by each USB-Accelerator when using EdgeTPU-cluster with J.TX2.

the overall performance of the USB-Accelerators and negate the benefits of having extra processing powers (EdgeTPU). Based on this evaluation, using two accelerators with CME appears to be the most effective approach to maximize DL inference throughput when using EdgeTPU-cluster.

6.2 Dynamic Model Placements (DMP)

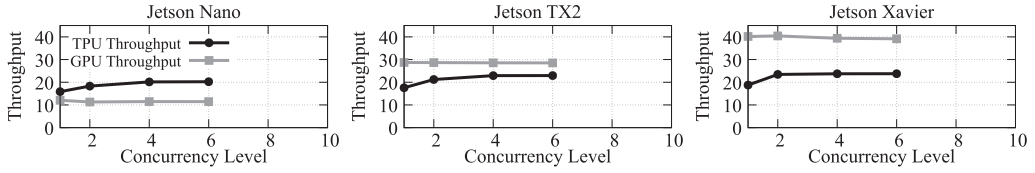
This subsection evaluates the DMP technique for AI multi-tenancy on edge devices and EdgeTPUs. DMP allows running multiple DL models simultaneously by placing DL models on an edge device's resource (CPU and/or GPU) and other DL models on EdgeTPUs. Because USB-Accelerator can be connected to edge devices via USB interfaces, the potential benefits from DMP can be improved DL inference throughput using heterogeneous resources in both edge devices and USB-Accelerator. However, DL inference tasks running on both on-board edge resources and USB-Accelerator are managed by the host edge devices so that there can be a performance penalty from resource contention. Therefore, in this evaluation, we focus on seeking answers to the following research questions:

(1) What are the performance benefits (e.g., DL inference throughput) from DMP on heterogeneous resources?
(2) What are the actual performance penalties of using DMP, compared to CME for AI multi-tenancy?
(3) What are the performance benefits and limitations of using EdgeTPU-cluster for DMP?

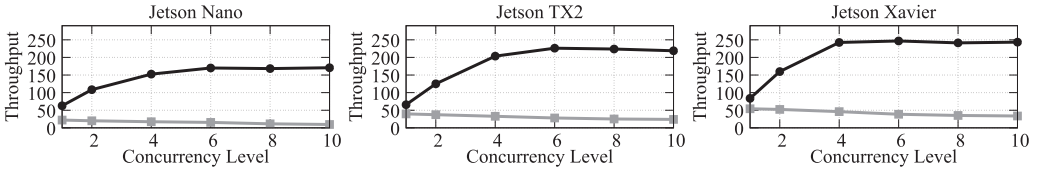
Similar to the previous CME evaluations, we used three DL models (Inception-V3, MobileNet-V1, and MobileNet-V2) because these models could perform inference tasks on all resource types in edge devices and EdgeTPUs. Moreover, as CME showed significant throughput improvement in our previous evaluation, we enabled CME on both edge devices and USB-Accelerator when measuring the inference throughput with DMP. To calculate the throughput, we used Equation (4) in Section 4.

We initially used all edge devices connected with USB-Accelerators and deployed DL models on both edge resources and EdgeTPUs. However, we omit the evaluation results of RPi4 and ODN2 because we could not observe the benefits of using DMP on such devices. Specifically, CPU resources on RPi4 and ODN2 were quickly saturated by both CPU-based and EdgeTPU-based DL inference tasks, and the overall inference throughput results with DMP on RPi4 and ODN2 were even lower (about 10%) than EdgeTPU-only inference throughput.

6.2.1 DMP Evaluation Results on Edge Device and a Single USB-Accelerator. As described above, CME was also enabled for DMP. The first step of this evaluation was to find an empirically optimal concurrency level that could produce the maximum DL inference throughput. While



(a) Inception-V3 on DMP (GPU: PyTorch, EdgeTPU: TensorFlow Lite)



(b) MobileNet-V1 on DMP (GPU: PyTorch, EdgeTPU: TensorFlow Lite)

Fig. 18. Throughput changes with different concurrency levels on both GPU and EdgeTPU when enabling DMP. We omit the results of MobileNet-V2 because the results are similar to the results of MobileNet-V1 (Figure 18(b)).

we reported the throughput changes with different concurrency levels on either GPU or EdgeTPU in Section 6.1, such high concurrency levels *may* not be achievable for DMP. This is because the edge device needs to manage multiple inference tasks on both GPU and EdgeTPU, and there will be a contention of the edge device's resources (e.g., memory). Therefore, we re-measured the throughput changes with different concurrency levels on both GPU and EdgeTPU for DMP. The evaluation results are shown in Figure 18. As expected, much lower levels of concurrency were supported by edge devices and USB-Accelerator. We further observed that memory resources on the edge device were quickly saturated with lower levels of concurrency because the device needed to handle more inference tasks and frameworks for DMP.

Then, we measured the overall throughput with DMP, and the evaluation used concurrency levels for GPU and EdgeTPU that could produce overall (accumulated) maximum DL inference throughput. Figure 19 shows DMP's DL inference throughput improvement against the single-tenancy cases. All J.Nano, J.TX2, and J.Xavier showed significantly increased throughput compared to the single-tenancy GPU or EdgeTPU-based inferences. On average (across all three models), J.Nano had 6.2× improved throughput over the single-tenancy on GPU and 2× increased throughput over the single-tenancy on EdgeTPU. Both J.TX2 and J.Xavier also showed significant throughput improvement for all three models ranging from 2×–9.9× over GPU-only or EdgeTPU-only inferencing with single-tenancy (with bached inferencing). However, this improved throughput can be in part due to leveraging both CME and DMP. Therefore, we also compare the DL inference throughput between the ideal throughput upper bound based on CME results (Section 6.1) and DMP. Please note that the ideal throughput upper bound is calculated by accumulating GPU throughput with CME and EdgeTPU throughput with CME measured separately.

Figure 20 reports the throughput comparison between (ideal) CME results and DMP results. The figure contains the results measured from J.TX2 when using PyTorch/MXNet (for GPU) and TFLite (for EdgeTPU). Please note that we omit the results from J.Nano and J.Xavier, but those omitted results show similar patterns to Figure 20. As shown in the figure, while the differences between the ideal throughput and DMP's throughput varied with DL models and DL frameworks, J.TX2 with DMP showed 34.6% and 25.3% lower DL inference throughput than the ideal throughput with CME (on both GPU and EdgeTPU). Such differences were mainly due to the resource contention and resource limits in the edge devices.

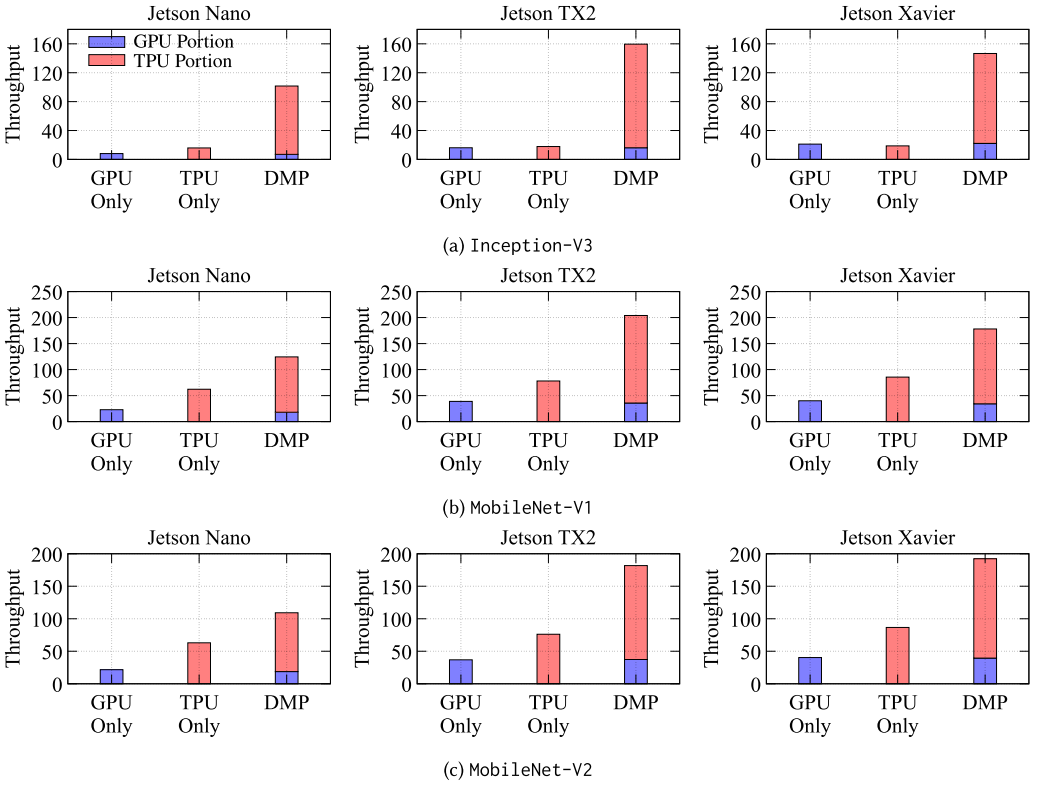


Fig. 19. Comparison of DL inference throughput between DMP and single-tenancy.

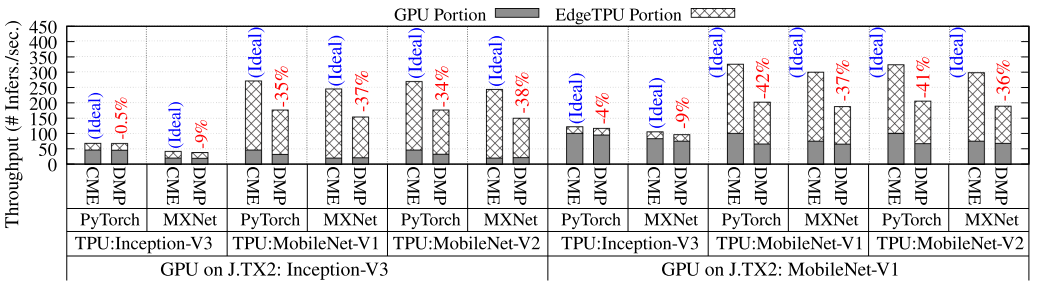
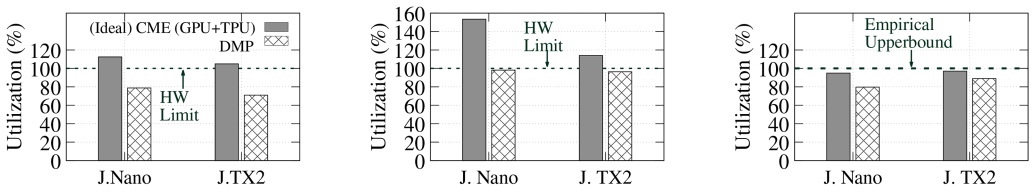


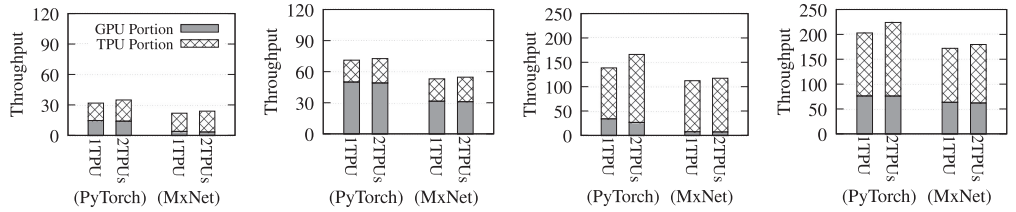
Fig. 20. J.TX2's DL inference throughput comparison between (ideal) results from CME and DMP. The (ideal) results from CME are calculated by the sum of separately measured CME throughput on GPU and EdgeTPU.

To understand the gap between the DMP's throughput and ideal throughput, we performed further analysis on resource consumption. Figure 21 shows the resource utilization (CPU, memory, and USB IO) between the ideal sum of CME on GPU/EdgeTPU (measured in Section 6.1) and DMP. The figure shows that the ideal throughput often could not be achievable with current HW specifications. Specifically, CPU (Figure 21(a)) and memory (Figure 21(b)) utilization should exceed the HW limits of the edge devices (more than 100%) to reach such high throughput. Moreover, similar to the CME analysis, memory was identified as a critical resource when enabling DMP. Specifically, we observed that memory utilization reached 100% with DMP while CPU utilization did not reach



(a) CPU Usage (b) Memory Usage (c) USB-IO Usage

Fig. 21. Resource usage comparison between (ideal) sum of CME on GPU/EdgeTPU and DMP.



(a) Nano (GPU:Inception-V3, TPU:Inception-V3) (b) Xavier (GPU:Inception-V3, TPU:Inception-V3) (c) Nano (GPU:MobileNet-V1, TPU:MobileNet-V1) (d) Xavier (GPU:MobileNet-V1, TPU:MobileNet-V1)

Fig. 22. Throughput comparison between DMP with 1 EdgeTPU and DMP with 2 EdgeTPUs (EdgeTPU-Cluster).

100%. Based on this observation, the DL inference throughput, when the memory resource is saturated, can be the empirical performance upper bound when enabling DMP. We also observed that resource contention could impact the DL inference throughput because the shared resources, such as memory and CPU, were needed to manage multiple DL models running on different resources. The decreased USB IO utilization (about 8% to 15%) with DMP (Figure 21(c)) was because such resource contentions and the reduced USB IO utilization could decrease DL inference throughput from in USB-Accelerator.

6.2.2 DMP Evaluation Results on Edge Device and EdgeTPU-cluster. The next evaluation is to measure the throughput improvements of DMP when leveraging EdgeTPU-cluster. As shown in Section 6.1.3, EdgeTPU-cluster with two accelerators showed almost maximum performance improvement due to the limitation of USB bandwidth. Therefore, in this evaluation, we used an EdgeTPU-cluster with two accelerators.

Figure 22 reports DL inference throughput changes from DMP employing EdgeTPU-cluster. Please note that the figure only shows the results from `J.Nano` and `J.Xavier` due to the page limitation. Also, we only report the combination of `Inception-V3` models on GPU and EdgeTPU-cluster (Figure 22(a) and (b)) and `MobileNet-V1` on GPU and EdgeTPU-cluster (Figure 22(c) and (d)). Other omitted results showed similar results to Figure 22. As shown in the figure, when using EdgeTPU-cluster on DMP, we observed a marginal improvement in DL inference throughput. On average, `J.Nano` and Xavier showed a 11% and a 5% increase in DL inference throughput over DMP with a single EdgeTPU accelerator. More specifically, EdgeTPU cluster could increase 11% (`J.Xavier`) to 20% (`J.Xavier`) of DL inference throughput with multiple accelerators, but at the same time, `J.Nano`'s GPU showed a 12% decreased throughput and `J.Xavier`'s GPU showed 3% decreased throughput. This decreased GPU throughput was mainly due to the resource contention and the resource-constrained nature of edge devices. Because the edge devices do not have sufficient CPU and memory resources (of course limited USB bandwidth) to process many tasks for

multiple accelerators, the devices naturally allocated less computing power. As a result, the DL inference throughput of GPUs was directly impacted (and decreased).

7 RELATED WORK

Several studies have been conducted quantifying the performance of various edge devices for DL and ML inference tasks [25, 34, 48, 49, 54, 57, 73]. However, most studies have focused on characterizing performance (e.g., latency and throughput) and efficiency (e.g., energy consumption) of edge devices and AI accelerators with single DL tasks.

pCamp [73] evaluated ML packages and frameworks' performance when executing image classification tasks on edge platforms, including `J.TX2`, `RPi4`, and `Nexus-6p`. This work reported latency (including model loading time), memory usage, and energy consumption from different ML packages. Hadidi et al. [34] have characterized various edge devices and AI accelerators (e.g., EdgeTPUs) with DL inference tasks. The authors analyzed the impact of DL frameworks and SW stacks and measured energy consumption and temperature when performing DL inference tasks. Moreover, several studies [25, 57, 69] have focused on characterizing the performance of DL inference tasks on different HW architectures and resource models (CPU, GPU, and portable AI accelerator). EmBench [25] performed a per-layer analysis of DL inference tasks to identify performance bottlenecks. Libutti et al. [49] conducted performance evaluations of DL inference tasks with portable, USB-based edge accelerators, including `USB-Accelerator` and Intel Neural Compute Stick [11].

More recently, Liang et al. [48] have conducted an experimental study to evaluate model splitting and compression techniques on edge devices and accelerators when performing co-inference tasks with clouds. Network latency, bandwidth usage, and resource utilization with various configurations were also reported when applying model splitting and compression to cloud-edge co-inference use-cases. Additionally, the authors have evaluated the concurrency model executions for multi-tenancy use cases. However, the concurrency evaluation is narrowly performed with only one model having a single batch size. Moreover, in addition to evaluating the CME strategy, our work also evaluates and characterizes the DMP strategy for AI multi-tenancy that leverages heterogeneous resources in edge and EdgeTPU.

8 CONCLUSION

This study investigated system approaches to maximize the DL inference throughput on *resource-constrained* edge devices and EdgeTPU accelerators with AI multi-tenancy.

We first evaluated various DL models' performance with image classification tasks on edge devices and AI accelerators, including CPU, GPU, and EdgeTPU. Based on the evaluation, we further investigated three system approaches for maximizing DL inference throughput. *Batched inferencing* is the approach for maximizing the throughput with DL single-tenancy use cases. With batched inferencing, GPU-equipped devices showed significant throughput improvement as multiple images could be processed in parallel on the GPU resources. We then explored the feasibility and effectiveness of AI multi-tenancy at the edge. Notably, two approaches were applied—*CME* and *DMP*. CME exploits the available system resources (CPU, memory, and GPU) to load more models into the system and process multiple inference tasks in parallel. DMP, on the other hand, leverages available, heterogeneous computing resources by placing models on different processors (GPU and EdgeTPU) and processes DL inference tasks on both the processors/accelerators simultaneously. Our evaluation results confirmed that CME and DMP were viable and successfully improved the system's overall throughput, including GPU and EdgeTPU, by a significant factor.

However, we also observed the limitations of the three approaches that will be future research explorations. For batched inferencing, the performance improvements start decreasing once the batch size exceeds a certain threshold (e.g., the number of GPU cores). Besides, due to the limited

memory size, there was a limit to the number of input images that could be simultaneously loaded into the memory. For CME and DMP with multi-tenancy, we started getting diminishing returns once the number of concurrently processed models exceeded the number of concurrent threads (or cores) supported by the CPUs. System memory also turned out to be a bottleneck as we increased the number of concurrent models. Finally, since USB bandwidth drived the rate at which `USB-Accelerators` could process models, multi-tenancy on EdgeTPUs showed performance gain only when fewer data transfers (because of model parameters swapping) were involved. Inherently sequential hardware design, such as shared internal USB hubs, was also a restricting factor when using multiple `USB-Accelerators` simultaneously.

This study confirmed that AI multi-tenancy on edge devices is a promising technique to improve the performance of DL tasks. Further study on strategic placement of models to minimize resource contention and isolation mechanism for dynamic control of DL inference throughput can push the performance boundaries of DL inferencing. In addition, since the multi-tenant applications share the same system memory, a thorough analysis of the security of individual applications (i.e., isolation from other models) is necessary for techniques like CME or DMP to be suitable for deployment.

REFERENCES

[1] 2020. Torchvision 0.5.0. Retrieved from https://pytorch.org/vision/. Accessed 9/12/2020.
[2] 2021. Azure AI. Retrieved from https://azure.microsoft.com/en-us/overview/ai-platform/. Accessed 2/8/2021.
[3] 2021. Cloud AI – Google Cloud. Retrieved from https://cloud.google.com/products/ai/. Accessed 2/12/2021.
[4] 2021. IBM Watson Machine Learning. Retrieved from https://www.ibm.com/cloud/machine-learning. Accessed 2/12/2021.
[5] 2021. Machine Learning on AWS. Retrieved from https://aws.amazon.com/machine-learning/. Accessed 2/13/2021.
[6] 2022. Coral Dev Board datasheet. Retrieved from https://coral.ai/docs/dev-board/datasheet/. Accessed 2/16/2022.
[7] 2022. Coral USB Accelerator Datasheet. Retrieved from https://coral.ai/docs/accelerator/datasheet/. Accessed 1/27/2022.
[8] 2022. Edge TPU Python API overview. Retrieved from https://coral.ai/docs/edgetpu/api-intro/. Accessed 1/27/2022.
[9] 2022. Environment Variables – MXNet v1.7.0. Retrieved from https://mxnet.apache.org/versions/1.7.0/api/faq/env_var. Accessed 2/2/2022.
[10] 2022. INA219 – 26V, 12-bit, i2c output current/voltage/power monitor. Retrieved from https://www.ti.com/product/INA219. Accessed 2/2/2022.
[11] 2022. Intel Neural Compute Stick. Retrieved from https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. Accessed 2/4/2022.
[12] 2022. Jetson Nano | Nvidia Developer. Retrieved from https://developer.nvidia.com/embedded/jetson-nano. Accessed 2/3/2022.
[13] 2022. kerascv 0.0.40. Retrieved from https://pypi.org/project/kerascv/. Accessed 2/3/2022.
[14] 2022. NVIDIA Jetson Linux Developer Guide : Clock Frequency and Power Management. Retrieved from https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/clock_power_setup.html#. Accessed 2/5/2022.
[15] 2022. Nvidia Jetson TX2. Retrieved from https://developer.nvidia.com/embedded/jetson-tx2. Accessed 2/5/2022.
[16] 2022. NVIDIA Jetson Xavier NX. Retrieved from https://developer.nvidia.com/embedded/jetson-xavier-nx. [ONLINE].
[17] 2022. NVPModel – Nvidia Jetson TX2 Dev. Kit. Retrieved from https://www.jetsonhacks.com/2017/03/25/nvpmodel-nvidia-jetson-tx2-development-kit/. Accessed 2/5/2022.
[18] 2022. ODROID-N2. Retrieved from https://wiki.odroid.com/odroid-n2/odroid-n2. Accessed 2/5/2022.
[19] 2022. pi-ina219 1.4.0. Retrieved from https://pypi.org/project/pi-ina219/. Accessed 2/5/2022.
[20] 2022. Raspberry Pi 4. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. Accessed 2/5/2022.
[21] 2022. TensorFlow Lite – ML for Mobile and Edge Devices. Retrieved from https://www.tensorflow.org/lite. Accessed 2/3/2022.
[22] 2022. tf.Graph – TensorFlow v2.4.1. Retrieved from https://www.tensorflow.org/api_docs/python/tf/Graph. Accessed 2/3/2022.
[23] 2022. tf.hub – TensorFlow Hub. Retrieved from https://www.tensorflow.org/hub. Accessed 2/3/2022.

[24] 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 G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*.

[25] Mário Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. 2019. EmBench: Quantifying performance variations of deep neural networks across modern commodity devices. *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*. 1–6.

[26] Jacob Benesty, Jingdong Chen, and Yiteng Huang. 2008. On the importance of the pearson correlation coefficient in noise reduction. *IEEE Transactions on Speech and Audio Processing* 16, 4 (2008), 757–765.

[27] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proceedings of the IEEE* 107, 8 (2019), 1655–1674.

[28] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*.

[29] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015).

[30] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282 (2017).

[31] Koustabh Dolui and Soumya Kanti Datta. 2017. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *Proceedings of the Global Internet of Things Summit*. IEEE, Geneva, Switzerland, 1–6.

[32] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling deep contrastive learning batch size under memory limited setup. arXiv preprint arXiv:2101.06983 (2021).

[33] He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, Shuai Zheng, and Yi Zhu. 2020. GluonCV and GluonNLP: Deep learning in computer vision and natural language processing. *Journal of Machine Learning Research* 21, 23 (2020), 23:1–23:7.

[34] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. 2019. Characterizing the deployment of deep neural networks on commercial edge devices. In *Proceedings of the IEEE International Symposium on Workload Characterization*. IEEE, Orlando, FL, 35–48.

[35] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the 4th International Conference on Learning Representations*.

[36] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*. 620–629.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Las Vegas, NV, 770–778.

[38] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the 15th European Conference Computer Vision*. Springer, Munich, Germany, 815–832.

[39] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

[40] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *Proceedings of the IEEE Conference on Computer Communications*. IEEE, Paris, France, 1423–1431.

[41] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[42] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv preprint arXiv:1602.07360 (2016).

[43] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lyu, and Zhihua Wu. 2020. MNN: A universal and efficient inference engine. In *Proceedings of the 3rd Conference on Machine Learning and Systems*.

[44] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor N. Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

[45] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computing Systems* 97 (2019), 219–235.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[47] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2020. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions Wireless Communications* 19, 1 (2020), 447–457.

[48] Qianlin Liang, Prashant J. Shenoy, and David E. Irwin. 2020. AI on the edge: Characterizing AI-based IoT applications using specialized edge architectures. In *Proceedings of the IEEE International Symposium on Workload Characterization*.

[49] Leandro Ariel Libutti, Francisco D. Igual, Luis Piñuel, Laura De Giusti, and Marcelo Naiouf. 2020. Benchmarking performance and power of USB accelerators for inference with MLPerf. In *Proceedings of the International Workshop on Accelerated Machine Learning*.

[50] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of IEEE* 107, 8 (2019), 1697–1716.

[51] Marcia Sahaya Louis, Zahra Azad, Leila Delshadtehrani, Suyog Gupta, Pete Warden, Vijay Janapa Reddi, and Ajay Joshi. 2019. Towards deep learning using tensorflow lite on RISC-V. In *Proceedings of the 3rd Workshop on Computer Architecture Research with RISC-V*. Phoenix, AZ.

[52] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. 2020. Distributed inference acceleration with adaptive DNN partitioning and offloading. In *Proceedings of the IEEE Conference on Computer Communications*. IEEE, Toronto, ON, Canada, 854–863.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[54] Kaustubh Rajendra Rajput, Chinmay Dilip Kulkarni, Byungjin Cho, Wei Wang, and In Kee Kim. 2022. EdgeFaaSBench: Benchmarking edge devices using serverless computing. In *Proceedings of the IEEE International Conference on Edge Computing*.

[55] Ju Ren, Hui Guo, Chugui Xu, and Yaoxue Zhang. 2017. Serving at the edge: A scalable IoT architecture based on transparent computing. *IEEE Network* 31, 5 (2017), 96–105.

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252. DOI: https://doi.org/10.1007/s11263-015-0816-y

[57] Colin Samplawski, Jin Huang, Deepak Ganesan, and Benjamin M. Marlin. 2019. Resource characterisation of personal-scale sensing models on edge accelerators. In *Proceedings of the International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*.

[58] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[59] Omid Setayeshfar, Karthika Subramani, Xingzi Yuan, Raunak Dey, Dezhi Hong, Kyu Hyung Lee, and In Kee Kim. 2021. ChatterHub: Privacy invasion via smart home hub. In *Proceedings of the IEEE International Conference on Smart Computing*. IEEE, 1–8.

[60] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. DOI: https://doi.org/10.1109/JIOT.2016.2579198

[61] Weisong Shi and Schahram Dustdar. 2016. The promise of edge computing. *IEEE Computer* 49, 5 (2016), 78–81.

[62] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*.

[63] Leslie N. Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820 (2018).

[64] Ion Stoica, Dawn Song, Raluca Ada Popa, David A. Patterson, Michael W. Mahoney, Randy H. Katz, Anthony D. Joseph, Michael I. Jordan, Joseph M. Hellerstein, Joseph E. Gonzalez and others. 2017. A berkeley view of systems challenges for AI. arXiv preprint arXiv:1712.05855 (2017).

[65] Piyush Subedi, Jianwei Hao, In Kee Kim, and Lakshmish Ramaswamy. 2021. AI multi-tenancy on edge: Concurrent deep learning model executions and dynamic model placements on edge devices. In *Proceedings of the 14th IEEE International Conference on Cloud Computing*.

[66] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[67] Martin Takác, Avleen Singh Bijral, Peter Richtárik, and Nathan Srebro. 2013. Mini-batch primal and dual methods for SVMs. *International Conference on Machine Learning*, PMLR, 1022–1030.

[68] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, Long Beach, CA, 8612–8620.

[69] Yu Wang, Gu-Yeon Wei, and David Brooks. 2020. A systematic methodology for analysis of deep learning hardware and software platforms. In *Proceedings of the Conference on Machine Learning and Systems*.

[70] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. February, 2019. Machine learning at facebook: Understanding inference at the edge. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*. Washington DC, 331–344.

[71] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward A. Lee, and John Kubiatowicz. 2015. The cloud is not enough: Saving IoT from the cloud. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing*. USENIX Association, Santa Clara, CA.

[72] Haotian Zhang, Gaoang Wang, Zhichao Lei, and Jenq-Neng Hwang. 2019. Eye in the sky: Drone-based object tracking and 3D localization. In *Proceedings of the ACM International Conference on Multimedia*.

[73] Xingzhou Zhang, Yifan Wang, and Weisong Shi. 2018. pCAMP: Performance comparison of machine learning packages on the edges. In *Proceedings of the USENIX Workshop on Hot Topics in Edge Computing*.

[74] Serena Zheng, Noah J. Apthorpe, Marshini Chetty, and Nick Feamster. 2018. User perceptions of smart home IoT privacy. *ACM on Human-Computer Interaction* 2, CSCW (2018), 200:1–200:20.

[75] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of IEEE* 107, 8 (2019), 1738–1762.

[76] Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. 2017. Understanding the security of discrete GPUs. In *Proceedings of the General Purpose GPUs*. 1–11.

Received 12 February 2022; revised 15 June 2022; accepted 21 June 2022