Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices

by

Piyush Subedi

(Under the Direction of In Kee Kim)

Abstract

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 (i.e., number of inferences per second) by exploiting various system techniques. This study investigates system techniques that enhance the overall inference throughput on edge devices with DL models for image classification tasks. We present various approaches, such as batched inferencing and multi-tenancy to utilize 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 up to 4× more inferences per second in devices equipped with high-performance GPUs like Jeston Xavier NX. Moreover, with multi-tenancy approaches, e.g., concurrent model executions and dynamic model placements, a throughput of more than 200 inferences per second can be achieved, which is 2× higher than the maximum throughput when running the models on a single tenant with batching enabled. Furthermore, a detailed analysis of the factors (hardware and software) that affect the throughput of

the systems is presented, thereby shedding light on areas that could be further improved to achieve high-performance DL inference at the edge.

INDEX WORDS: [edge computing, deep learning, machine learning, multi-tenancy, performance evaluation, characterization]

Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices

by

Piyush Subedi

BS, Sona College of Technology, India 2016

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree

Master of Science

Athens, Georgia

2021

Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices

by

Piyush Subedi

Major Professor: In Kee Kim

Committee: Lakshmish Ramaswamy

Jaewoo Lee

Electronic Version Approved:

Ron Walcott

Vice Provost for Graduate Education and

Dean of the Graduate School

The University of Georgia

August 2021

Acknowledgments

I would like to extend my sincere gratitude to some of the people who have provided me great support and guidance in completing this research. First and foremost, I want to thank my major advisor, Dr. In Kee Kim for providing me the opportunity to work on this research. He was always helpful and never stopped encouraging me to explore and exploit my full potential. I also like to thank Dr. Lakshmish Ramaswamy and Dr. Jaewoo Lee for their invaluable inputs and constructive suggestions throughout the research. I am also very grateful to my colleague Jianwei Hao, who was always ready to help me with the work and always encouraged insightful discussions. Finally, I must thank my family and friends who have directly or indirectly helped me complete this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High network latency and privacy concerns have hindered the adoption of cloud computing for IoT and CPS (Cyber-Physical System) applications that require prompt responses and collect sensitive information [6, 40]. Wide-area data transmission from IoT sensors to cloud data centers often causes intolerable network latency [76, 34]. Moreover, IoT systems that monitor user activity (e.g., smart homes, activity tracker) have the risk of a privacy breach because all sensing data need to be stored and processed in cloud data centers. [79, 59]. *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) [60, 61, 36]. The proliferation of IoT sensors and SBCs (Single-Board Computers), e.g., Raspberry Pi [54], enable the adoption of edge computing paradigm across a wide range of latency-sensitive applications, including vehicle to vehicle communication, emergency alert services, traffic monitoring, proactive caching for faster web pages loading, wearable cognitive-assistance systems, and augmented reality [14, 60].

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 various 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. [34, 19, 75]. 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 [11, 45, 3, 29, 64]. Therefore, emerging edge applications, such as autonomous driving, disaster response systems, drone-based surveillance [55, 80, 43], *may* offer poor QoS (Quality of Services) 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 [9, 22] and quantization [18, 73] techniques are developed to downsize DL models without significant degrade in their accuracy [28, 23, 58]. Similarly, studies involving efficient partitioning of heavier models and subsequent offloading to the cloud [34, 24, 46] show how the edge-cloud hybrid model could be leveraged for DL inference tasks. Moreover, light-weight ML (Machine Learning) frameworks, e.g., Tensorflow Lite [44], MNN (Mobile Neural Network) [33], are developed for enabling on-device inference. Finally, the advent of edge devices with GPUs, e.g., NVIDIA's Jetson Series [32, 48, 49], and AI accelerators, e.g., Google's Coral Accelerator [13] and Intel's Neural Compute Stick [31] 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 [78, 2, 57, 42, 17, 41] 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, DL 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 [77]. Furthermore, system approaches to maximize DL inference throughputs (e.g., the number of inferences per second) on edge devices have not been deeply investigated.

This study starts 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 edge devices. Then, three techniques to maximize the DL inference throughput on the devices: *batched inferencing*, *concurrent model executions*, and *dynamic model placements* are employed. Batched inferencing exploits the parallel computing capabilities of GPU resources on edge devices and maximizes the DL inference throughput for single DL tasks. Both concurrent model executions (CME) and dynamic model placements (DMP) are techniques for maximizing DL inference throughput with DL 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 DL multitenancy 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, this work evaluates DL deployment strategy on edge devices with multiple AI accelerators (e.g., EdgeTPU cluster) and reports the benefits and limitations of the deployment scenarios with EdgeTPU clusters.

The extensive evaluation results confirm a DL inference throughput improvement on edge devices and AI accelerators by leveraging three approaches. For the DL single-tenancy use cases, batched inferencing results in up to 4× more inferences per second in devices equipped with high-performance GPUs, including Jeston Nano, TX2, and Xavier NX. With CME on edge devices' GPU (for the DL multi-tenancy cases), the result shows that a maximum throughput of 140 inferences per second can be achieved, which is more than 2× higher inference throughput than the maximum throughput with batched inferencing. Using a single `USB-Accelerator`, CME results in throughput as high as 230 inferences per second. With two `USB-Accelerators`, the throughput rises to nearly 260 inferences per second, which is more than 4× the throughput achieved with single-tenancy. Additionally, DMP combined with concurrency and batched inferencing outperformed GPU-only and TPU-only performance on DL models by a factor of 8 and 3, respectively. DMP with multiple TPUs further improves this throughput by 25%. Finally, a de-

tailed analysis of the factors (hardware and software) that affect the throughput of the systems 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:

- 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 and a thorough analysis of the factors that affect the DL inference throughput.
- 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 DL multi-tenancy.
- With these three techniques, discover the empirical upper-bound of batch size, throughput, and model concurrency on edge devices and AI accelerators for DL inference tasks.
- Identify the performance benefits and limitations when adopting DMP to leverage heterogeneous resources on edge resources and EdgeTPUs (AI accelerators).
- Investigate the benefits and limitations of deploying DL models on edge devices with EdgeTPU clusters.

In particular, to the best of my 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.

1.1 Background

This section provides the background of edge devices, AI accelerators, DL models, and DL frameworks used in this study.

1.1.1 Edge Devices and EdgeTPU Accelerators

We use three categories of devices in this study, which are 1) general-purpose edge devices, 2) edge devices with GPU accelerator, and 3) EdgeTPU-based AI Accelerators. We choose Raspberry Pi 4 and Odroid-N2 for the general-purpose edge device category, Jetson Nano, TX2, and Xavier for the edge device with GPU accelerator category, and Coral Dev Board and USB Accelerator for EdgeTPU-based AI Accelerators. The summary of these devices is shown in Table 1.1.

General-purpose Edge Devices.

Raspberry Pi 4 [54] and Odroid-N2 [51] are chosen for this category. Raspberry Pi 4 (`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 CPU resources for processing DL inference task and it neither has a GPU nor specialized HW accelerators for DL processing.

Odroid-N2 (`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 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.

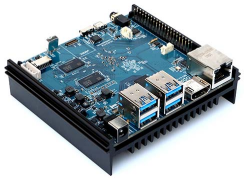
Edge Devices with GPU Accelerators.

Three Jetson devices developed by NVIDIA are chosen for this category. Jetson Xavier NX (`J.Xavier`) [49] is one of the high-end edge devices developed by NVIDIA. `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 and tensor processing units, it is expected to show higher DL inference performance than other two Jetson devices. `J.Xavier` can use different power modes with different power consumption and resource activation. In this study, we use power `mode-2` [47] that enables all six CPU cores at 1400 MHz of speed and all GPU cores with 1100 MHz of speed. The `mode-2` allows faster processing of CPUs/GPUs and consumes 15W of power.

Table 1.1: Specification of Edge Devices and AI Accelerators used in this study.

	Raspberry Pi 4 (Model B)	**Odroid N2**	**Jetson Nano**	**Jetson TX2**	**Jetson Xavier NX**	**Coral Dev Board**	**Coral USB Accelerator**
CPU	Quad-core Cortex-A72 @1.5GHz	Quad-core Cortex-A73 @ 1.8GHz + Dual-core Cortex-A53 @ 1.9GHz	Quad-Core ARM Cortex-A57 MPCore @1.5 GHz	Dual-Core NVIDIA Denver 2 64-Bit CPU @2GHz + Quad-Core ARM Cortex-A57 MPCore processor @2GHz	6-core NVIDIA Carmel ARMv8.2 64-bit CPU @1.5-1.9GHz	Quad Cortex-A53, Cortex-M4F	✗
GPU	✗	Mali-G52	128-core NVIDIA Maxwell	256-core NVIDIA Pascal	384-core NVIDIA Volta	Integrated GC7000 Lite Graphics	✗
Co-processor	✗	✗	✗	✗	✗	Google Edge TPU (4 TOPS)	Google Edge TPU (4 TOPS)
Memory	4GB LPDDR4	4GB LPDDR4	4GB LPDDR4	8GB LPDDR4	8GB LPDDR4	1GB LPDDR4	✗
Storage[1]	External SD card	8 MB eMMC	16 GB eMMC	32 GB eMMC	16 GB eMMC	8 GB eMMC	✗
LAN	✓	✓	✓	✓	✓	✓	✗
WiFi	✓	✗	✗	✓	✗	✓	✗
Power	1.8-5.3W	2-5W	5/10W	7.5W/15W	10W/15W	4 TOPS @ 2 W	4 TOPS @ 2 W
USB (*Type A*)	2 × USB 2.0 + 2 × USB 3.0	4 × USB 3.0	4 × USB 3.0	1 × USB 3.0	4 × USB 3.0	1 × USB 3.0	1 × USB 3.0 *Type C*
OS	Raspbian GNU/Linux 10	Ubuntu 18.04.4 LTS	Ubuntu 18.04.5 LTS	Ubuntu 18.04.5 LTS	Ubuntu 18.04.5 LTS	Mendel Debian 10	✗
Price	$55.00	$79.00	$99.00	$399.00	$399.00	$129.00	$59.00

(a) Raspberry Pi 4

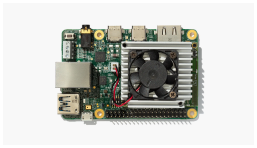


(b) Odroid N2

(c) Jeston Nano

(d) Jeston TX2

(e) Jetson Xavier NX



(f) Coral Dev Board

(e) Coral USB Accelerator

Figure 1.1: Images of edge devices and EdgeTPUs used.

Jetson TX2 (`J.TX2`) [48] is another 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 a 8 GB LPDDR4 RAM, which is shared by CPU and GPU. Among five different power modes in `J.TX2` [50], 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 Nano (`J.Nano`) [32] is a small yet powerful single board computer specialized in DL processing. 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 default mode for maximizing the device performance.

All three Jetson devices allow users to alter the power modes using *nvpmodel*. We decide to configure the power mode that can provide the highest performance (e.g., the highest frequency for GPU and CPU) to understand the upper bound of DL inference throughput and the performance of such devices. The power configurations for three devices are shown in Table 1.2.

Table 1.2: Summary of Power Modes used in NVIDIA Jetson Devices

	Mode	# Of Active CPU Cores	CPU Frequency	GPU Frequency	Power Usage
Jetson Xavier	Mode-2	6	1.4 GHz	1.1 GHz	15W
Jetson TX2	Mode-0	6	2.0 GHz	1.30 GHz	15W
Jetson Nano	Mode-0	4	1.5 GHz	ĩ GHz	10W

EdgeTPU-based AI Accelerators.

We use two EdgeTPU-based AI accelerators, which are Google's Coral Dev Board (`DevBoard`) [12] and Coral USB Accelerator (`USB-Accelerator`) [13]. `DevBoard` is a single-board computer equipped with a quad-core Cortex-A53 CPU (at 1.5GHz) and 1GB LPDDR4 RAM[2], as well as onboard TPU (Tensor Processor Unit) co-processor (accelerator), which can perform 4 trillion operations per second (TOPS) at 2W of power consumption.

`USB-Accelerator` is a USB-type TPU accelerator (co-processor) for ML/DL. The performance of `USB-Accelerator` is equivalent (4 TOPS at 2W) 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 hold 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 [69] and DL models that are fully 8-bit quantized and compiled specifically for EdgeTPU architecture.

[2]Newer version of `DevBoard` have 2G or 4G of LPDDR4 RAM, but we use `DevBoard` with 1GB RAM.

1.1.2 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 do not fit into the resource-constrained, low-capacity edge devices like `RPi4`. Based on the capacity restriction, we select 9 pre-trained DL models[3] that can be deployed on resource-constrained edge devices to perform DL inference tasks (e.g., image classification). These 9 DL models have different number of layers, number of FLOPS (Floating Point Operation Per Second), and the number of parameters. Such differences and the overview of the 9 selected models are described in Table 1.3.

Moreover, each model has different characteristics and advantages. For instance, AlexNet [38] utilizes multiple convolutional, max-pooling, and fully-connected layers with GPU to improve the accuracy significantly. DenseNet [25] uses a connection between one layer and the subsequent layers in a feed-forward network to reduce the number of model parameters. Inception [66] scales up networks by utilizing the factorized convolutions and aggressive regularization to increase the model's accuracy with minimal increase in computation cost. MobileNet models [23, 58] employ depthwise separable convolutions to reduce the model size and the inference latency. ResNet [20] uses deep residual nets to increase accuracy but still keeps lower complexity. SqueezeNet [27] manages AlexNet-level accuracy with 50$\times$ fewer parameters by employing three design strategies (e.g., filter replacement, decreased input channel, downsampling rate) and fire module composed of a squeeze (1$\times$1 filters) and a expand layer (a mix of 1$\times$1 and 3$\times$3 convolution filters). VGG-16 [62] is a large convolution network with over 138 million parameters that improves on AlexNet in terms of accuracy and efficiency by introducing multiple smaller(3$\times$3) kernel-sized filters instead of large kernel-sized filters.

[3] These 9 DL models are CNN (Convolutional Neural Network) models for image classifications.

Table 1.3: Overview of DL models used in this study.

DL Model	Year	Input Size	Num. Layers	Billion FLOPS	# Params (Million)	Approx. File Size (MB)	DL Framework (FW) Availability			
							PyTorch	MXNet	TF	TFLite
AlexNet [39]	2012	224×224	8	0.7	61	244	✓	✓	✓	✗
DenseNet-161 [26]	2016	224×224	161	7.9	28.7	115	✓	✓	✓	✗
Inception-V3 [67]	2015	299×299	48	2.9	27.2	101, 25*	✓	✓	✓	✓
MobileNet-V1 [23]	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 [21]	2015	224×224	18	1.8	11.7	46	✓	✓	✓	✗
ResNet-50 [21]	2015	224×224	50	4.1	25.6	102	✓	✓	✓	✗
SqueezeNet-V1 [28]	2016	224×224	15	0.4	1.2	5	✓	✓	✓	✗
VGG-16 [63]	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 information for TF Lite.

1.1.3 Deep Learning Frameworks

We use four, widely-used open-source DL frameworks, which are PyTorch [52] [1.4.0], MxNet [7][1.4.0], TensorFlow [1][2.0.0], and TensorFlow-Lite [69] for this study. PyTorch, 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 used to run DL models on EdgeTPU of `DevBoard` and `USB-Accelerator`.

MxNet [7]: MxNet is a scalable open-source deep learning 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. With the backend developed using C++, it currently supports Python, Java, Scala, R, Julia, Go, Clojure, Perl, MATLAB and JavaScript for frontend developers. Finally, MxNet allows for the deployment of pre-trained models to resource-constrained devices like mobile devices or edge devices.

PyTorch [52]: Developed by Facebook's AI Research Lab, PyTorch is an open-source machine learning framework built on top of the Torch[4] library and designed specifically for Python. One of the most prominent features of PyTorch is that it provides a *NumPy-like* tensor computing support but with an added capability of operating tensors on a CUDA-capable NVIDIA GPU. Unlike TensorFlow (< 2.0.0) and MxNet, PyTorch adopts a dynamic computational graph based approach which means that the computational graph is created on the fly (runtime). This feature enables flexibility for developers when writing and debugging deep learning applications. Like MxNet, PyTorch also supports scalable distributed training and performance optimization and is light enough to be deployed to low-end devices.

TensorFlow [1]. TensorFlow, from Google Brain team, is another open-source and widely popular machine learning framework. It uses a dataflow graph in which nodes represent operations while the edges represent tensors. It can map the nodes of the graph across many machines in a cluster (distributed), and within a machine across multiple computational devices (CPUs, GPUs or TPUs) thereby proving to be a

[4]Currently, the library is not in active development (https://github.com/torch/torch7).

Table 1.4: Source of pre-trained models

Framework	**Source Repository**	**Version**
PyTorch	`Torchvision` [72]	0.2.2.post3
MxNet	`GluonCV` [16]	0.10.0
TensorFlow	`Kerascv` [35]	0.0.40
TensorFlow-Lite	`TensorFlow Hub` [71]	0.2.0

scalable framework. Starting from version 2.0.0, TensorFlow supports *eager execution mode* (which is also the default mode of execution) which emulates the behavior of PyTorch's dynamic computation graphs.

TensorFlow-Lite [69]: 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. TensorFlow-Lite has two primary parts, which are interpreter and converter. The interpreter run DL models, and the converter converts TensorFlow models into TensorFlow-Lite models.

The pre-trained and compatible versions of the nine models described in Section 1.1.2 for each of the three frameworks are readily available across various publicly accessible and maintained repositories. Table 1.4 lists the sources for the pre-tained models for each of the frameworks used in this study. All the models have been trained on the same `ImageNet ILSVRC-2012` [56] dataset.

Chapter 2

Literature Review

Several studies have been conducted to quantify the performance of various edge devices for DL and Machine Learning (ML) inference tasks [78, 2, 57, 42, 17, 41]. 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 [78] 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. [17] 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 as well as measured energy consumption and temperature when performing DL inference tasks. Moreover, several studies [57, 2, 74] have focused on characterizing the performance of DL inference tasks on different HW architectures and resource models (CPU, GPU, EdgeTPU). EmBench [2] has performed a per-layer analysis of DL inference tasks to identify performance bottlenecks. Libutti et al. [42] conducted performance evaluations of DL inference tasks with portable, USB-based edge accelerators, including Coral USB Accelerator and Intel Neural Compute Stick [31].

More recently, Liang et al. [41] 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 performs the evaluation and characterization of the DMP strategy for AI multi-tenancy that leverages heterogeneous resources in edge and EdgeTPU.

Chapter 3

Evaluation Methodology

3.1 Approaches for Deep Learning Inference Throughput Maximization

The goal of this study is to investigate different approaches for maximizing the inference throughput for DL applications on various edge devices and AI accelerators. For the DL single-tenancy use cases, batched inferencing approach is explored. FOr the DL multi-tenancy use cases, two approaches are employed, which are *concurrent model executions* and *dynamic model placements*.

3.1.1 Batching

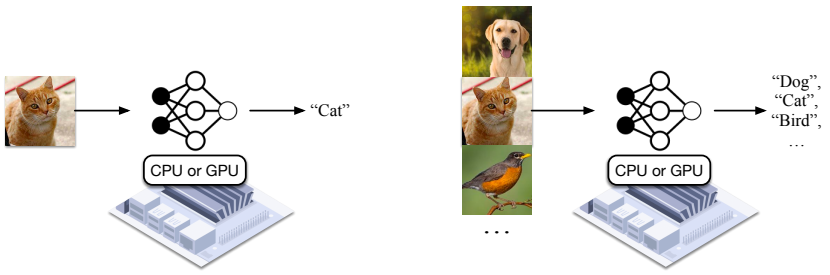
The integration of GPUs on modern devices has made it possible to train and run deep learning models significantly faster primarily because of the ability of GPUs to run parallel tasks efficiently. *Batching*, in the context of inference, refers to the idea of having the model process multiple inputs at the same time (Single Instruction Multiple Data), which aligns perfectly with GPU's design. Edge devices can be required to handle batches of data either from multiple 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, wearable devices). Therefore, by exploiting the capabilities of ML frameworks to run DL models on

GPUs, the impact of batched inferencing on the overall throughput, specifically on GPU-embedded devices (`J.Nano`, `J.TX2`, `J.Xavier`) is studied. In the experiments, the *batch size* (the number of images provided as input to the models) is gradually increased in powers of two as GPUs can efficiently map virtual processors onto physical processors if the data to be parallelized is in power of two [81]. Also, optimised libraries that work with matrix operation tend to be the most effective when processing batches of size in the order of powers of two [68].

3.1.2 Multi-tenancy through Concurrent Model Executions (CME)

Multi-processing and multi-threading paradigms have been extensively applied to general purpose systems in order to boost up the overall performance of the system. As most edge devices are now equipped with multi-core CPUs and have operating systems that facilitate multi-threading and multi-processing, it has become feasible to deploy such techniques at the edge. Naturally, without having to wait for the single model to process previous request, newer models can be executed *in parallel* to process newer requests thereby improving the overall throughput of the system. Besides, multi-tenancy also enables maximum resource utilization leveraging largely unused GPUs and other system resources (memory, CPU).

Concurrent Model Execution refers to the process of executing multiple models on the same device. 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 services (e.g., inference tasks). Due to hardware limitations with memory and CPU usage, it is unclear as to how many number of models can be concurrently executed (i.e level/degree of concurrency) and which level of concurrency yields the maximum performance before it invariably starts declining (Amdahl's law [65]). Therefore, it is important to empirically identify the upper bound of throughput improvement and the concurrency level (the number of co-running DL models) on the devices' resources by CME. In this study, the change in throughput with gradually increasing levels of concurrency is recorded. The concurrency level obtained from the last successful execution is considered as the maximum concurrency level supported by the edge devices and EdgeTPUs. With concurrent execution of models providing software level parallelism, the



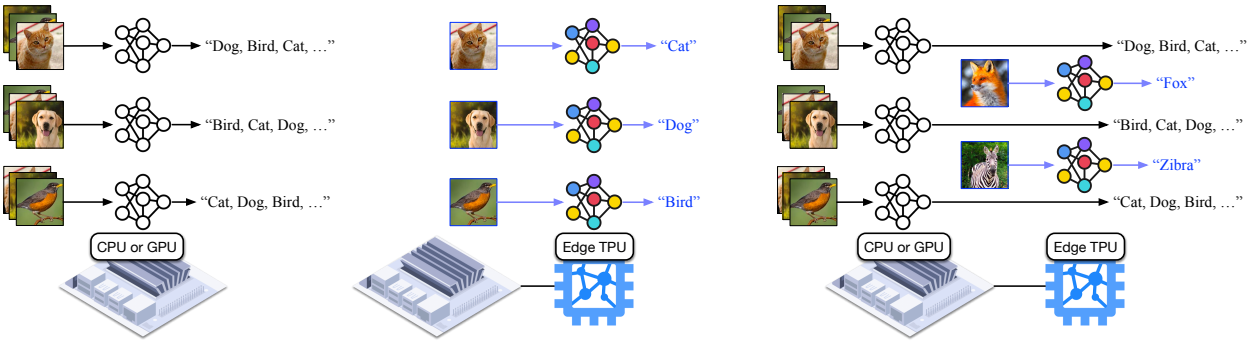
(a) Single-Tenancy with Single Batch (b) Single-Tenancy with Multi Batch

Figure 3.1: Single-Tenancy on Edge Device for DL Inferencing

impact of introducing redundant hardware (e.g multiple `USB-Accelerators`) for DL inferencing is also evaluated. Running concurrent models with more resources can lead to higher throughput but other hardware bottlenecks like USB bandwidth can hinder the performance.

3.1.3 Multi-tenancy through Dynamic Model Placements (DMP)

DMP is another strategy for enabling multi-tenancy at the edge with the idea of increasing the overall throughput of the system. Dynamic placement of tasks according to the availability of resources is a commonly studied problem, particularly in the realm of cloud computing [37, 5, 10]. DMP at the edge refers to a deployment strategy wherein DL models are *placed* in different processors while sharing other system resources (memory, power, and so on). Specifically, models are simultaneously run on GPU (host device) and TPU (`USB-Accelerator`) and the overall improvement in the throughput is recorded. Such a strategy can provide the edge devices with the ability to run heavier models on more powerful resources (e.g GPUs) for higher accuracy while delegating latency-critical tasks to the less powerful processor such as TPU. Building on this, we can also design a scheduler that can orchestrate the placement of inference tasks on available (or less busier) resource/processor. Similar to CME, the performance impact of using TPU cluster for DMP is also studied. In other words, more number of processors are introduced to place the models on.



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

Figure 3.2: Multi-Tenancy on Edge Device and EdgeTPU for DL Inferencing

3.2 Evaluation Process and Benchmarker Design

Because the goal of this study is to investigate systematic approaches to maximize the DL inference throughput on edge devices and AI accelerators, the primary metric considered for all the experiments is the DL inference throughput. In general, the DL inference throughput is calculated by *inferences per second* as shown in equation-(3.1). The definition of *number of inferences* varies with the type of experiment. For instance, when leveraging the single tenancy (running one DL model at a time on a device), the number of inferences in equation-(3.1) is calculated as "batch size" × "the number of batches." On the other hand, when leveraging DL multi-tenancy (running multiple models concurrently), the number of inferences will be calculated by "concurrency level" × "batch size" × "the number of batches."

$$DL\ Inference\ Throughput = \frac{Number\ of\ Inferences}{Total\ Execution\ Time} \tag{3.1}$$

To correctly measure the DL inference throughput, a benchmarker has been developed that measures the DL inference throughput and collects other necessary system statistics together. It is deployed along with an image classification application on the edge devices and EdgeTPUs accelerators. The measurement procedure of the benchmarker is illustrated in Figure 3.3.

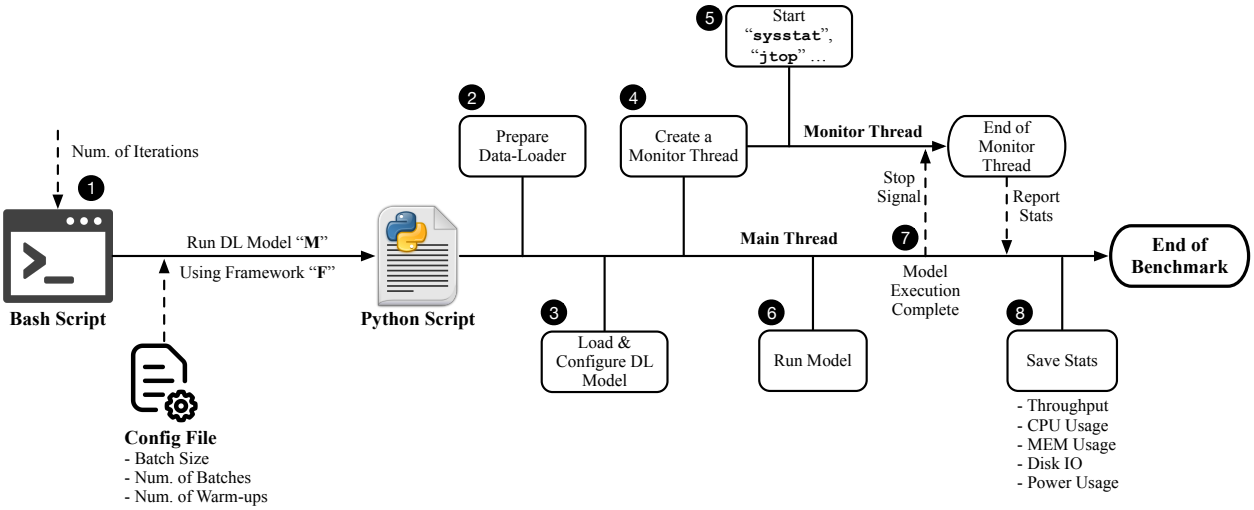


Figure 3.3: Throughput Evaluation Process of Benchmarker

The benchmarker is invoked from a bash script (❶ in Figure 3.3) that takes in parameters in a config file specific to a measurement. The config file includes a DL model to run, a framework to use, 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 of input to run per iteration, and resources (CPU, GPU, or EdgeTPU) used for the inference task, etc. The bash script then runs the benchmarker (written in Python) with all of 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 ready to be fed into the DL model. Next, the benchmarker initiates a DL framework as per the config file, and it loads the DL model into the main memory and configures the model to be executed on CPU, GPU, or EdgeTPU based on the configuration file (❸). The next step (❹) is the warm-up phase, which ensures 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 the model is being executed (❺). Simultaneously, the main thread of the bench-

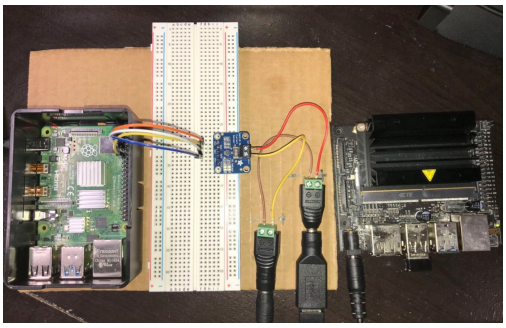


Figure 3.4: Experimental setup for power measurement. Power consumption by a target edge device is being measured and transmitted to a computing board using I2C cables by `INA-219` chip.

marker starts to run inference tasks using the DL model and the input data from the data-loader (❻). Once the pre-defined number of batches of inputs have been processed, the main thread instructs the monitoring thread to stop capturing system statistics (❼). Finally, a measurement report consisting 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 measured data.

The monitoring thread is responsible for collecting diverse system statistics using `sysstat`, `usbtop`, and `jtop`. `sysstat` is used to measure CPU usage and Memory usage during the benchmarking is running. `usbtop` is for measuring USB IO bandwidth and `jtop` is leveraged for measuring power consumption. However, `jtop` is available to measure power-related statistics for NVIDIA's Jetson devices. So, `INA-219` [30], a voltage, current and power measurement chip, has been employed for measuring the power consumption of other edge devices like `RPi4` and `ODN2`. With a default resistance of $0.1\,\Omega$, the `INA-219` chip allows measuring the power consumption with a current sensing range of ± 3.2 A and a voltage range of 0 V to 26 V. `pi-ina219` [53], a Python library is used to communicate with the `INA-219` chip. The experimental setup with `INA-219` for power measurement is shown in Figure 3.4.

Chapter 4

Evaluation Results

4.1 Evaluation with DL Single-Tenancy

Since DL single-tenancy (running a single DL model at a time on devices) is a common deployment model serving DL inference tasks at the edge, this section evaluates the throughput of DL single-tenancy on edge devices (Section 4.1.1) and EdgeTPUs (Section 4.1.3). Moreover, as an approach for maximizing the inference throughput for single-tenancy on GPUs, the impact and performance of *batched inferencing* is evaluated, in which a DL model processes a batch of input images and outputs the classification results of all the images simultaneously (Section 4.1.2). Finally, a thorough analysis of the experiment results and the factors that affect the DL inference throughput on edge devices and AI accelerators (Section 4.1.4) is presented . The results reported in this section will serve as the baseline performance to evaluate further throughput maximization approaches such as DL multi-tenancy.

4.1.1 Single Input At A Time

The first set of experiments are to measure 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 4.1 reports the 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 GPU resource while the results from RPi4 and ODN2 are measured on CPU resources.

The results show that the inference throughput varies significantly across different DL models. In particular, DL models with fewer parameters and floating-point operations (e.g., AlexNet, MobileNet-V1, SqueezeNet-V1) show higher throughput than DL models with a relatively higher number of operations (e.g., DenseNet-161, Inception-V3, VGG-16). The throughput differences between smaller and larger models are more prominent in CPU-based devices, e.g., RPi4 and ODN2. For example, the inference throughput with SqueezeNet-V1 on RPi4 and ODN2 reports $10\times - 22\times$ higher than the throughput with DenseNet-161. Both RPi4 and ODN2 have a throughput of less than 1 (one inference per second) for all the DL frameworks when running DenseNet-161, which amplifies the difference against results from SqueezeNet-V1 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) is only $3\times - 8\times$ better than DenseNet-161's throughput. This further highlights the benefits of using a GPU over CPU for DL inferencing. Without the support for data parallelism in CPUs, very deep models like DenseNet-161, Inception-V3, and VGG-16 that involve very high number of floating point operations, the CPU has to spend significantly large amount of time processing the models. This increases the latency which in turn degrades the throughput.

The results also confirm that, for single-batch inference, the DL inference throughput on the GPU-based devices significantly outperforms the throughput on the CPU-based devices. This result shows the advantage of GPUs over CPUs, particularly in processing DL workloads on edge devices. The edge devices with GPU (J.Nano, J.TX2, and J.Xavier) are capable of process $4\times - 80\times$ 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 with using PyTorch. On average, J.Nano, J.TX2 and J.Xavier show $17\times$, $30\times$, and $38\times$ higher throughput than RPi4, respectively, and the DL inference throughput results on these devices are $38\times$, $62\times$, and $75\times$ higher than ODN2. The results also confirm the performance differences among the three GPU-based edge devices. In terms of GPU frequency, J.Nano has the least powerful GPU which is evident in the results as well. There's

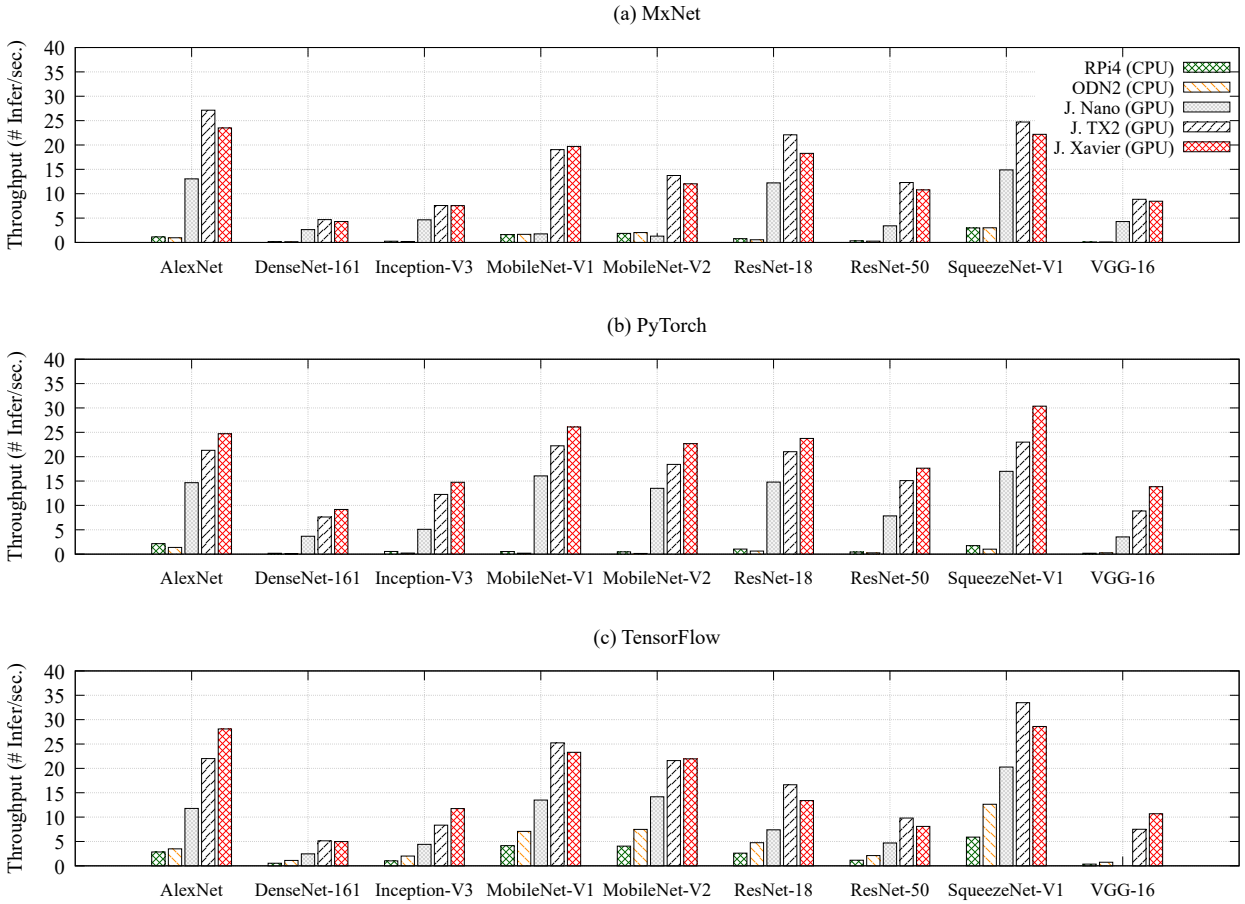


Figure 4.1: DL Inference Throughput Variations across Models, Edge Devices and DL Frameworks with a Batch Size of 1

no clear winner when comparing `J.Xavier` and `J.TX2`. `J.TX2`'s GPU frequency (1.3 GHz) is higher than that of `J.Xavier`'s (1.1 GHz). This can be attributed as the reason why `J.TX2`'s results are better, particularly when using MxNet and TensorFlow. For PyTorch, however, `J.Xavier` seems to perform better than `J.TX2`. One possible reason behind this deviation could be that since `J.Xavier` has higher number of GPU cores (384) compared to `J.TX2` (256), PyTorch *may* be better at parallelizing its dynamic computation graph on more number of cores than the other two frameworks. A detailed inspection on the behavior of these ML frameworks could help better understand this difference.

Amongst the three DL frameworks, PyTorch reports the highest throughput on GPUs. On an average, the throughput of DL models when using PyTorch is 31% and 26% higher than using MxNet and using TensorFlow respectively. PyTorch's superiority on GPUs can be justified by the fact that it has been built on top of Torch library which has been primarily designed to make tensor operations on GPU fast and efficient, as described in Section 1.1. On CPUs, on the other hand, TensorFlow significantly outperforms the other two frameworks. More specifically, the average throughput across all the models on CPUs using TensorFlow is almost 5× the results from MxNet and 10× from PyTorch. As TensorFlow has been designed to support mapping of nodes (from computational graph) across multicore CPUs [1], it enables faster computation in CPUs. This design emphasis *could* be the reason why TensorFlow showed better results in CPUs than the other two frameworks.

As we can see, MxNet, on all the devices, is the least performing framework. However, the results from MxNet on `J.Nano`, particularly for models MobileNet-V1 and MobileNet-V2 is exceptionally poor (less than 5 inferences per second which is more than 4× lower than the other two frameworks). My reasoning behind this behavior is two folds. Firstly, when MxNet leverages GPU resources, the default behavior of the framework is to use NVIDIA cuDNN library to auto tune convolution layers. Simply put, it searches for the best performing convolution algorithm in the first run, which allows subsequent model executions to run faster. However, this process is highly memory-intensive, and further analysis reveals that `J.Nano`'s 4GB memory is not large enough to complete this process, and the algorithm search process on `J.Nano` often results in out-of-memory errors. Hence, this auto-tuning mechanism had to be turned off so that the models could complete the inference. Further details of this problem will be provided when discussing the results of DL multi-tenancy in Section 4.2.1. Secondly, the design of the two MobileNets is such that it involves many small memory-bound element-wise operations such as ReLU [23]. Without optimization strategies like operator fusion [8] enabled, the processing time of such models can rise steeply. Such performance tuning strategies work well on `J.TX2` and `J.Xavier` because of larger memory size. This is why the results from `J.TX2` and `J.Xavier` are comparable to the other frameworks.

4.1.2 Impact of Batched Inferencing on DL Inference Throughput

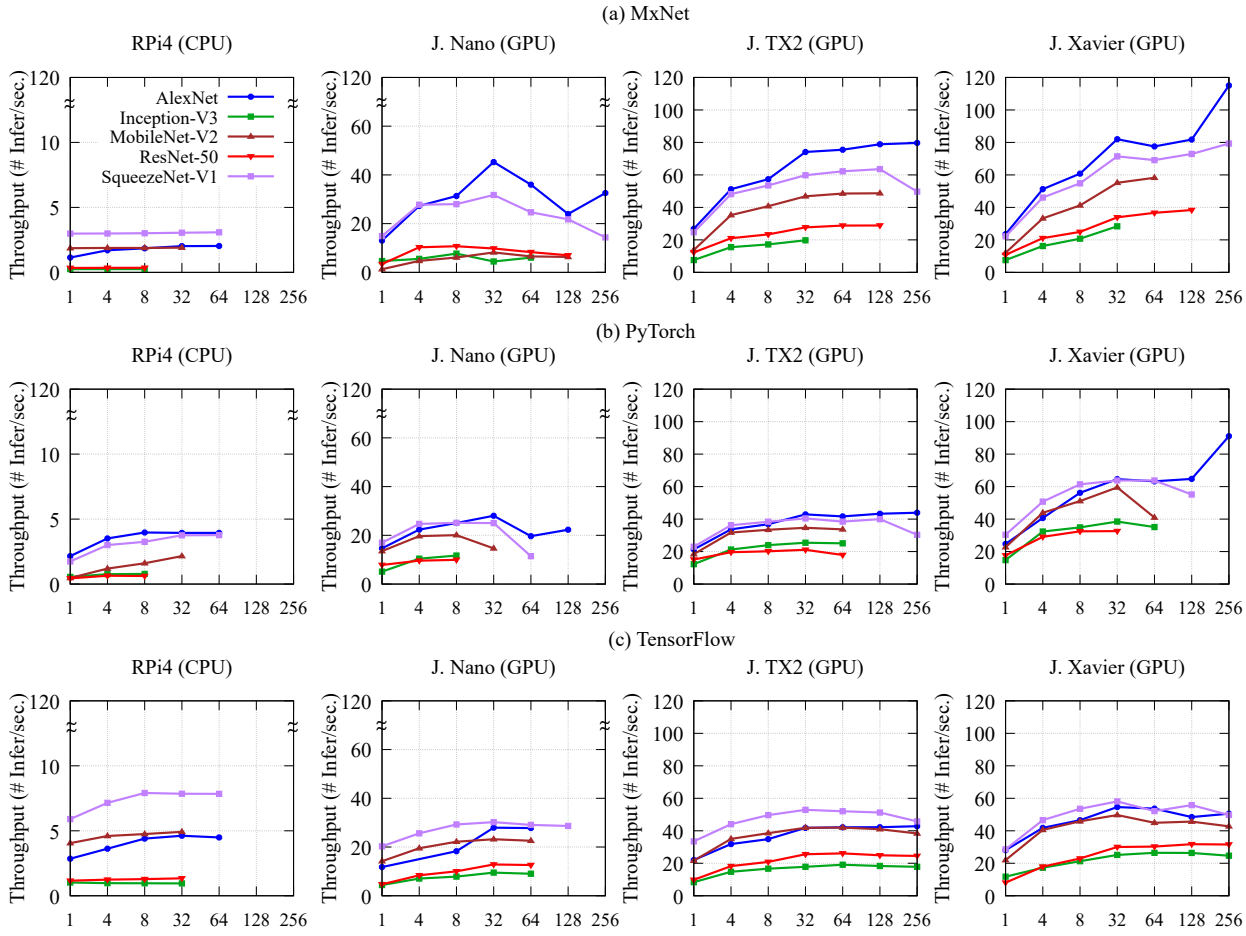


Figure 4.2: Throughput Variation across DL Models, Edge Devices and DL Framework with Different Batch Sizes

Figure 4.2 reports the results of running DL models for different batches of input images with increasing batch size, using the three ML frameworks. Although the experiment was run on all the models, just a representative set of models is reported as the ones that have been left out showcased similar results. In addition, `ODN2`'s results aren't reported, as, like `RPi4`, the impact of batching on CPUs is insignificant. Finally, as of yet, possibly due to limited RAM size, EdgeTPUs do not support batched inferencing and are prohibited by the *edgetpu* API itself.

As shown in Figure 4.2, there is a significant rise in throughput with increasing batch size for GPU embedded devices. On average, across `J.Nano`, `J.TX2`, and `J.Xavier`, and across all the frameworks

and models, for a batch size of 32, there is a 240% gain in throughput. AlexNet reports the highest overall throughput reaching as high 110 inferences per second on `J.Xavier`. Unsurprisingly, the impact of batching on `J.Xavier`, with more GPU cores, is higher than in `J.TX2` and `J.Nano`. Specifically, `J.Xavier`'s results are 25% and 42% greater than that of `J.TX2` and `J.Nano`, respectively. In general, while a larger batch size appeared to have a positive impact on throughput, very large batch sizes (> 128), *may* not always result in improved throughput and. This suggests that employing the right (or optimal) size of the input batch will be critical for improving the DL inference throughput on edge devices

Interestingly, contrary to the results from Section 4.1.1, MxNet is the best performing framework where the impact of batching, on an average, is 43% higher than PyTorch and 54% higher than TensorFlow. There is no apparent reason as to why one framework is better at batched inferencing than the other. A much more detailed inspection of the architecture and design of the frameworks may provide insight into such differences in their behavior. However, this is out of the scope of this study and has been left as a future work.

4.1.3 DL Inference Throughput on EdgeTPU with Single Tenancy

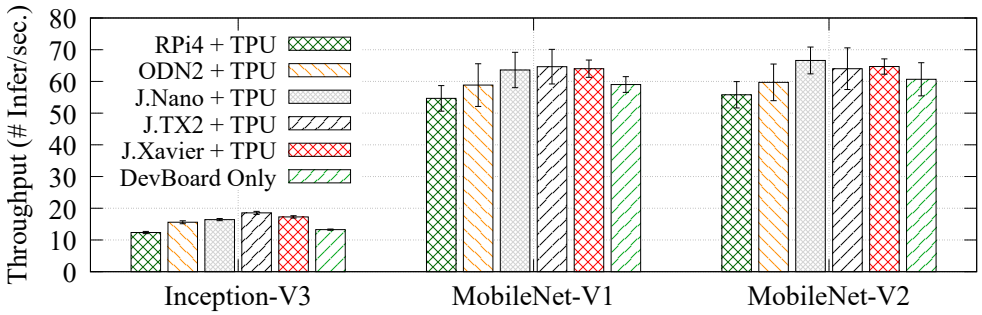


Figure 4.3: DL inference throughput of the three quantized DL models on EdgeTPU. The error bars indicate standard deviation.

EdgeTPUs can provide boost to DL inference throughput since they are specifically designed to process tensors, one of the primary components/objects of CNNs, rapidly. Figure 4.3 reports the DL inference throughput of `USB-Accelerator` and `DevBoard`. Please note that the `USB-Accelerator`

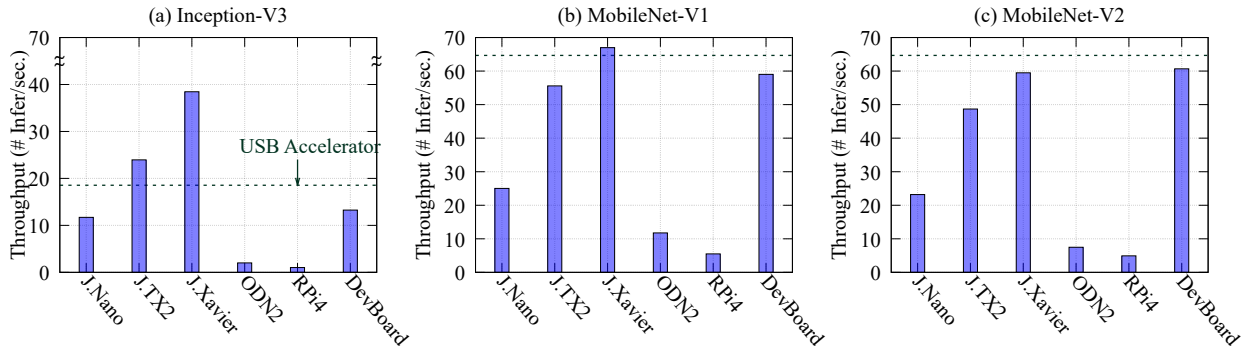


Figure 4.4: Comparison of inference throughput in CPU, GPU, and EdgeTPU. The throughput results of CPU- and GPU-based inferences are the maximum throughput results of those devices amongst all three frameworks. Please note that `USB-Accelerator`'s throughput in this graph is the maximum throughput from the results reported in Figure 4.3.

requires a host device to run because it is USB-type EdgeTPU. All five edge devices have been used with an `USB-Accelerator` to measure the throughput from the `USB-Accelerator`. Also, as described in Section 1.1.2, three DL models (Inception-V3, MobileNet-V1, and MobileNet-V2) are used in this evaluation because quantized version of only these models are available for EdgeTPU. Moreover, TensorFlow-Lite is used the DL framework in this evaluation.

As Figure 4.3 shows, the inference throughput on are fairly consistent and can reach as high as 65 inferences per second for MobileNet-V1/V2 and 16 inferences per second for Inception-V3. Furthermore, the benefits of using EdgeTPUs are confirmed by comparing the inference throughput against CPU- and GPU-based throughput results (shown in Figure 4.4). MobileNet-V1/V2 on `USB-Accelerator` show significantly higher throughput (6× higher than `ODN2`, 12× higher than `RPi4`) over CPU-based inferences and outperform GPU-based inferences on `J.Nano` and `J.TX2` by 3× and 1.5× higher throughput, respectively. And, `J.Xavier` and `USB-Accelerator` show comparable performance with minimal difference in throughput. With the same co-processor, `DevBoard` also show slightly lower throughput over `USB-Accelerator`. The lower throughput in `DevBoard` can be related 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 perform such management tasks for `USB-Accelerator`).

When performing DL inference tasks with Inception-V3, a larger model than MobileNet-V1/V2, `USB-Accelerator` also outperforms CPU-based inferences (8×, 17× higher throughput `ODN2` and `RPi4`) as shown in Figure 4.4). On the other hand, `USB-Accelerator`'s throughput with Inception-V3 is lower than the maximum throughput results from `J.TX2` and `J.Xavier`. However, `J.TX2`'s and `J.Xavier`'s throughout results in Figure 4.4 are obtained with *batched inferencing* on these devices' GPUs. For the single-batch inferencing (i.e one inference per request), `USB-Accelerator` outperforms all the other devices, including `J.TX2` and `J.Xavier` with all three models.

Finally, when comparing the three models, MobileNet-V1/V2, with smaller model size and fewer parameters show higher throughput than Inception-V3. This is because the parameters DL models have to be constantly swapped between the host memory and `USB-Accelerator`. So, DL inferencing with a larger model can considerably slows down the performance of `USB-Accelerator` due to the parameter swap overhead.

4.1.4 Analysis of Factors for Influencing DL Inference Throughput with Single-Tenancy

This subsection 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

First, a correlation analysis performed to investigate the factors to change the DL inference throughput on edge devices and EdgeTPUs. The correlation analysis is performed by calculating the Pearson Correlation Coefficient (in equation-(4.1)) [4] of measured throughput results and resource usage statistics. 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,

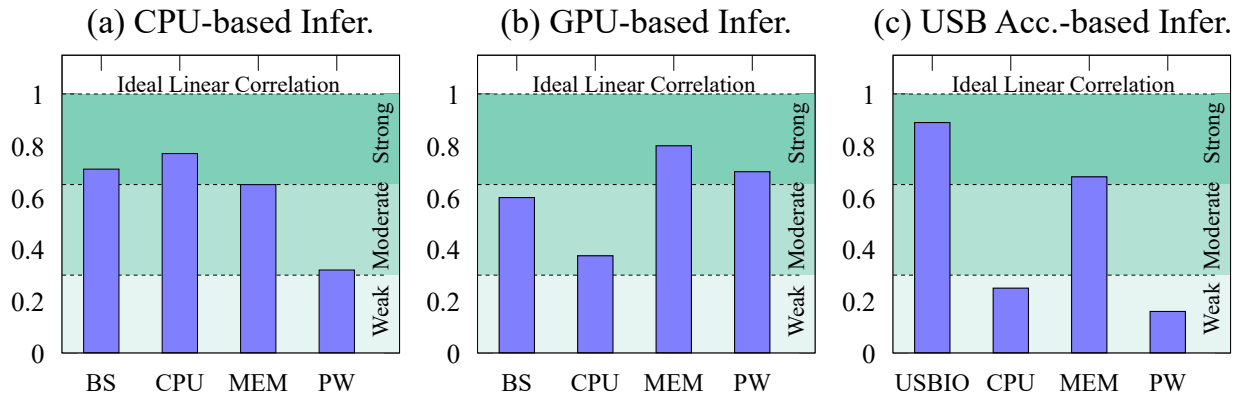


Figure 4.5: Correlated factors that change the inference throughput. (BS: Batch Size, CPU: CPU usage, MEM: memory usage, PW: Power consumption, USBIO: USB IO bandwidth usage)

and 0 means there is no correlation between two variables.

$$\rho = \frac{cov(x,y)}{\sigma_x \sigma_y} = \frac{\sum_i^n (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i^n (x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{4.1}$$

Fig. 4.5 shows the correlated factors for the DL inference throughput when using CPU, GPU, and EdgeTPU. For the CPU-based inferences on `RPi4`, `ODN2` (shown in Figure 4.5(a)), the batch size, CPU, and memory had a strong correlation with the inference throughput results. This is because CPU is the main computing resources for performing the DL tasks, and memory resources are used for loading and storing 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 4.1 summarizes the impact on throughput with increasing batch size and the corresponding increment in CPU usage on `RPi4`. As explainted earlier in Section 4.1.2, heavier models (DenseNet-161, Inception-V3) do not show much change in throughput with increasing batch size on CPUs because of the high processing demands as highlighted by the 100% CPU usage. However, for lighter models (AlexNet, MobileNet-V1/V2, SqueezeNet-V1), on an average, there is a gain of 40% in throughput with nearly 70% increase in CPU usage with increasing batch size.

Table 4.1: Variation in CPU usage and DL inference throughput using TensorFlow with different batch sizes in RPi4

Model	Batch Size	Avg. Throughput	Avg. CPU Usage (%)
AlexNet	1	2.854	53.851
	32	4.615	100
DenseNet-161	1	0.534	76.514
	32	0.564	100
Inception-V3	1	1.015	81.033
	32	0.95	100
MobileNet-V1	1	4.141	59.242
	32	5.499	93.02
MobileNet-V2	1	4.049	60.921
	32	4.918	87.388
ResNet-18	1	2.605	73.071
	32	2.898	98.324
ResNet-50	1	1.161	72.804
	32	1.341	98.102
SqueezeNet-V1	1	5.895	53.337
	32	7.851	85.821

For the GPU-based inference tasks on J.Nano, J.TX2 and J.Xavier (shown in Figure 4.5(b)), memory, power, and batch sizes had a relatively stronger correlation with the DL inference throughput. Specifically, the power consumption had a strong correlation with the throughput as the GPU module in edge devices consumed more power than typical CPUs in edge devices. As shown in Table 4.2, every increase in throughput results is associated with higher power consumption. On an average, 15-20% increase in power usage corresponds to 90-100% gain in throughput. Regarding the batch sizes, as we saw in Section 4.1.2, increasing batch size could significantly affect the DL inference throughput. In general, while a larger batch size appear to be positive for increasing the throughput, an interesting observation is that using larger batch sizes does not always result in increasing the throughput. For example, the batch sizes of 4, 8, or 32 often showed higher throughput than the batch sizes of 128 and 256 of AlexNet on J.Nano in Figure 4.2 in Section 4.1.2. This suggests that employing the right (or optimal) size of the input batch will be critical for improving the DL inference throughput on edge devices. On the other hand, CPU resources, as expected, show a relatively weaker correlation with the DL inference throughput in the GPU-based inference as CPU is only used for managing the device and processes co-running (non-DL) applications rather than performing the DL tasks.

For the inference tasks with USB-Accelerator (shown in Figure 4.5(c)), the USB bandwidth between a host edge device and the USB-Accelerator and memory usage on host edge devices have 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 in order to perform DL inference tasks. Therefore, low USB IO bandwidth between the host device and USB-Accelerator limits data rates for switching models/parameters so that the throughput can be decreased. Further analysis about the impact of USB bandwidth on the DL inference throughput on USB-Accelerator is also performed, which is described in the next subsection.

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

Table 4.2: Variation in Power Consumption with changing inference throughput in `J.Nano`, `J.TX2` and `J.Xavier` when using TensorFlow.

Devices	Models	Batch Size	Avg. Throughput	Avg. Power Usage (mW)
J. Nano	AlexNet	1	11.78	4299.55
		32	27.94	4433.60
	DenseNet-161	1	2.45	5553.82
		32	5.38	6517.80
	Inception-V3	1	4.41	4109.27
		32	9.50	4751.90
	MobileNet-V2	1	14.17	4307.73
		32	23.19	4625.20
	ResNet-50	1	4.69	4113.09
		32	12.85	4590.30
	SqueezeNet-V1	1	20.28	3876.18
		32	30.21	4385.10
J. TX2	AlexNet	1	22.01	5536.90
		32	41.79	5555.10
	DenseNet-161	1	5.13	4586.10
		32	12.19	6221.70
	Inception-V3	1	8.35	5008.40
		32	17.81	6074.50
	MobileNet-V2	1	21.60	4637.70
		32	41.86	5209.60
	ResNet-50	1	9.81	4934.50
		32	25.55	5897.90
	SqueezeNet-V1	1	33.48	4991.40
		32	52.91	5429.40
J. Xavier	AlexNet	1	28.09	4967.90
		32	54.61	5754.60
	DenseNet-161	1	4.98	4761.70
		32	18.13	7686.80
	Inception-V3	1	11.75	4877.40
		32	25.12	6691.20
	MobileNet-V2	1	21.96	4364.30
		32	49.71	5369.90
	ResNet-50	1	8.09	4648.60
		32	30.01	6209.40
	SqueezeNet-V1	1	28.60	4574.60
		32	58.09	5677.80

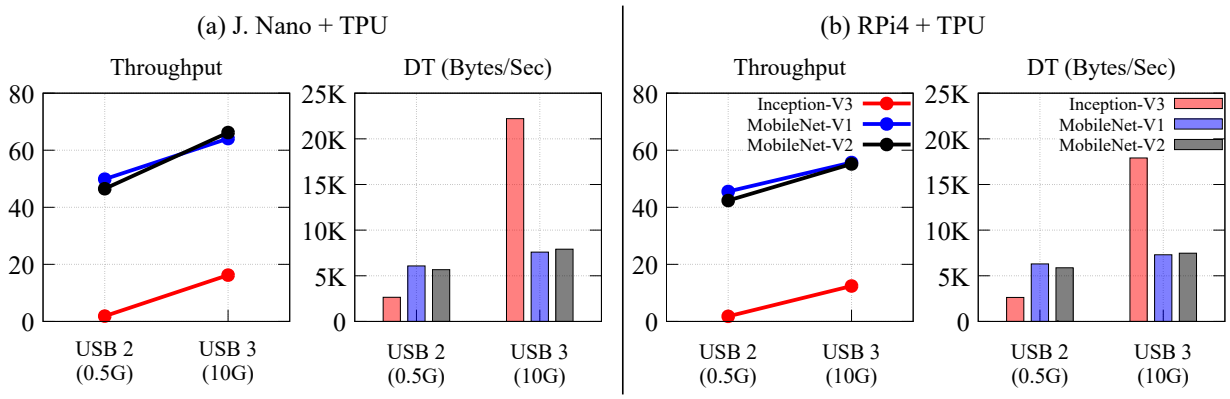


Figure 4.6: Difference in DL inference throughput and data transfer with USB 2.0 and 3.0 interfaces. (DT: Data Transfer Rate)

Impact of USB Bandwidth on `USB-Accelerator`'s DL Inference Throughput

To investigate the impact from the USB IO bandwidth, the DL inference throughput changes from `USB-Accelerator` is measured by connecting it with two edge devices (`RPi4` and `J.Nano`) with different USB interfaces; (a) USB 2.0 with up to 0.5GB of bandwidth, (b) USB 3.0 with up to 10GB of bandwidth. As shown in Fig. 4.6, the results confirm that USB's IO bandwidth can considerably change the DL 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 USB 3.0. `J.Nano` also showed 1.4× (MobileNet-V2) and 8.7× (Inception-V3) higher throughput than `USB-Accelerator` with USB 2.0. Clearly, larger USB IO bandwidth facilitates faster switching of model parameters and input data between the host device and `USB-Accelerator` which enables faster inference.

4.1.5 Summary

To summarize this section, the performance and behaviors of various commercial edge devices and EdgeTPU-based AI accelerators with single-tenancy was characterized. In addition, some of the correlating factors when it comes to improving the overall throughput were discussed. The results highlight the benefits of GPU-based inferencing, particularly when processing batches of inputs. In addition, the results of

running models on EdgeTPUs show the considerably high throughput that could be obtained by either connecting `USB-Accelerators` to other edge devices or using a `DevBoard`.

4.2 Evaluation with DL Multi-Tenancy

As discussed in Section 3.1, CME and DMP are to improve the throughput of DL inference tasks with DL multi-tenancy. In this section, benefits and limitations of both CME (Section 4.2.1) and DMP (Section 4.2.2) approaches are evaluated.

4.2.1 Evaluation Results with Concurrent Model Executions (CME)

This subsection reports our evaluation results of CME for DL multi-tenancy. In particular, by evaluating CME, answers to the following questions are being sought;

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?

For the rest of this study, only three DL models - Inception-V3, MobileNet-V1 and MobileNet-V2 are being used because pre-trained version of these models are *officially* available for all the edge devices including EdgeTPUs. Furthermore, TensorFlow has been excluded from this CME evaluation due to issues with `kerascv` [35] and `tf.Graph` [70] APIs that did not fully support concurrent executions (i.e operations were not thread-safe). Finally, regarding the throughput calculation with CME, the equation-(3.1) is changed such that number of inferences in the equation is now calculated as "concurrency level" $\times$ "batch size" $\times$ "the number of batches.", where concurrency level refers to the number of concurrent models executing at a time. The total inference time is the total time taken for *all* the concurrent models to finish execution.

Evaluation Procedure

With results from running one DL model available (Section 4.1), the number of co-running DL models ("*concurrency level*") is gradually increased on the devices. This process is continued until the benchmarker fails to run either because we have fully saturated the memory or because the system cannot spin-off more threads. 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, only the results with leveraging CME on GPUs (`J.Nano`, `J.TX2` and `J.Xavier`) and EdgeTPUs (`DevBoard` and `USB-Accelerator`) have been reported. The measurement results from CPU resources have been omitted because very marginal benefits were found.

The benchmarking process described in Fig 3.3 is tweaked such that instead of running a model in the main thread (step 6), 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.

CME Evaluation Results on GPU in Edge Devices

Fig. 4.7 and 4.8 show DL inference throughput changes with increasing concurrency levels using PyTorch and MxNet respectively on `J.Nano`, `J.TX2` and `J.Xavier`. The effect of leveraging both batched inferencing and CME can also be seen in the figures.

It is clear from the results that running multiple DL inferencing tasks can improve the overall throughput of the system. Similar to the results of batched inferencing, concurrent execution of lighter models like MobileNet-V1/V2 yield higher gain in throughput while heavier models like Inception-V3 show minor improvement. In particular, compared to single-tenancy cases, CME resulted in 1.4×-2×, 1.8×-2.7×, 1.7×-2.9× increase in overall throughput on `J.Nano`, `J.TX2`, `J.Xavier` respectively, across all the three models with PyTorch. The results with MxNet (Figure 4.8) are less impressive in that we see relatively lower throughput improvement - 1.3×-1.5× on `J.TX2` and 1.5×-1.8× on `J.Xavier`. `J.Nano`'s results with MxNet are particularly poor (even 13% lower throughput than single-tenancy cases) which can be at-

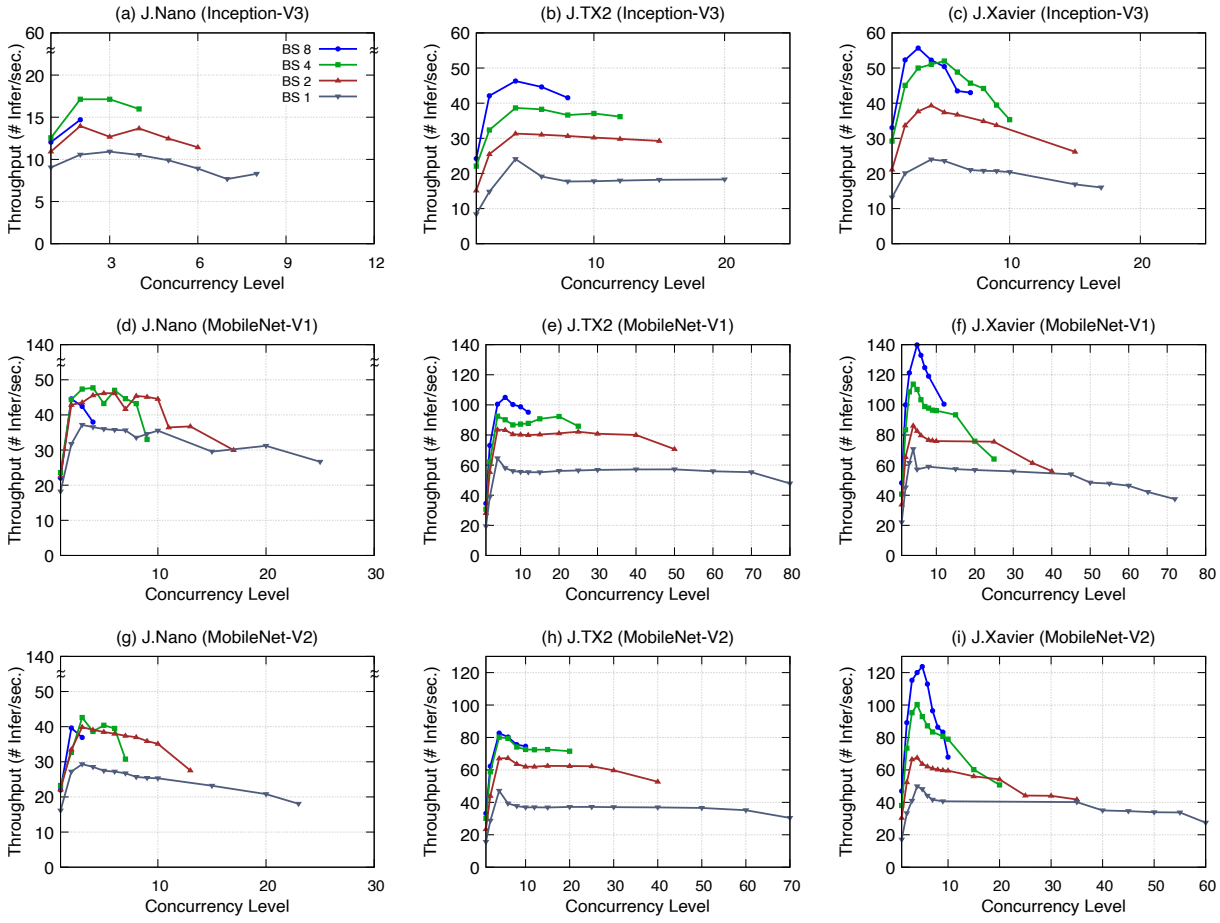


Figure 4.7: Concurreny measurement results on `J.Nano`, `J.TX2` and `J.Xavier` GPUs with PyTorch (BS: Batch Size)

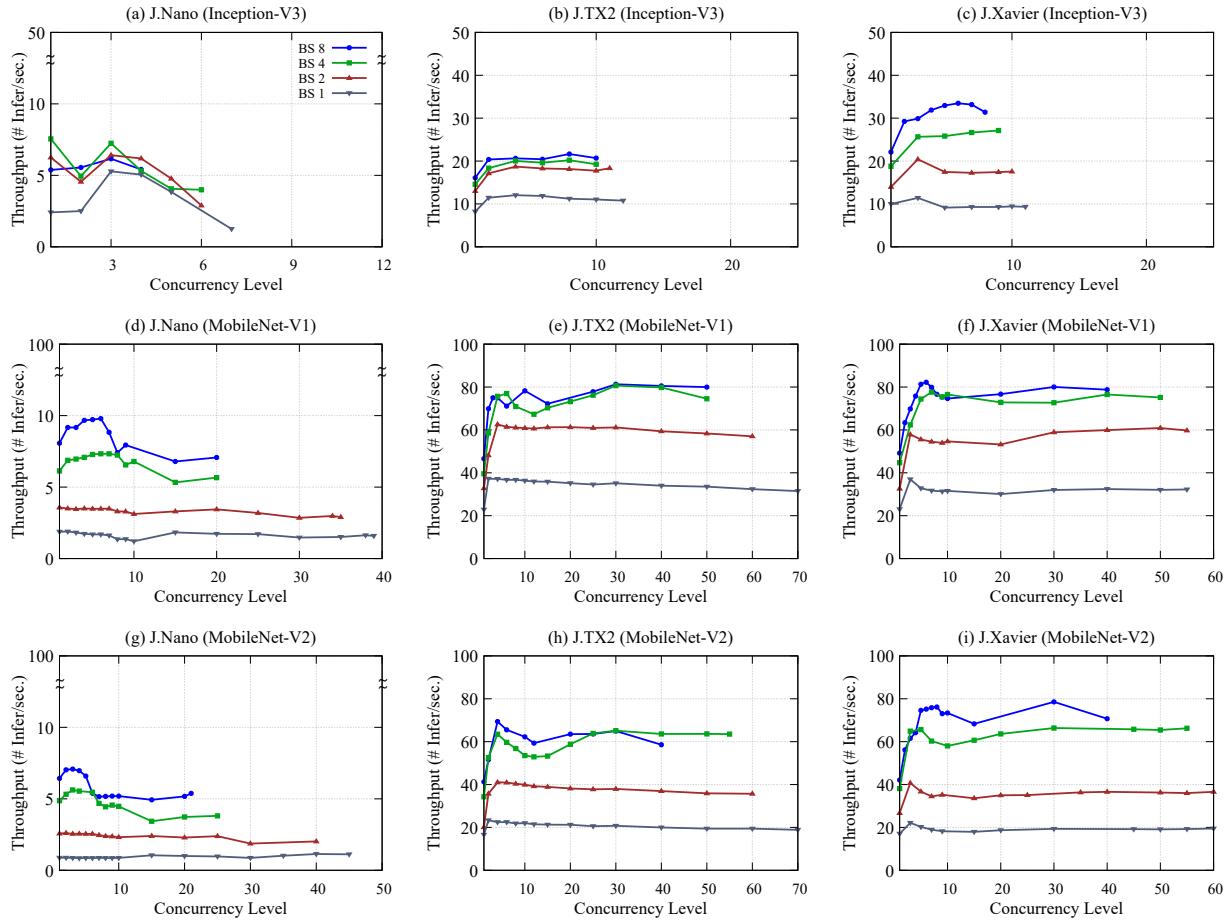


Figure 4.8: Concurreny measurement results on `J.Nano`, `J.TX2` and `J.Xavier` GPUs with MxNet (BS: Batch Size)

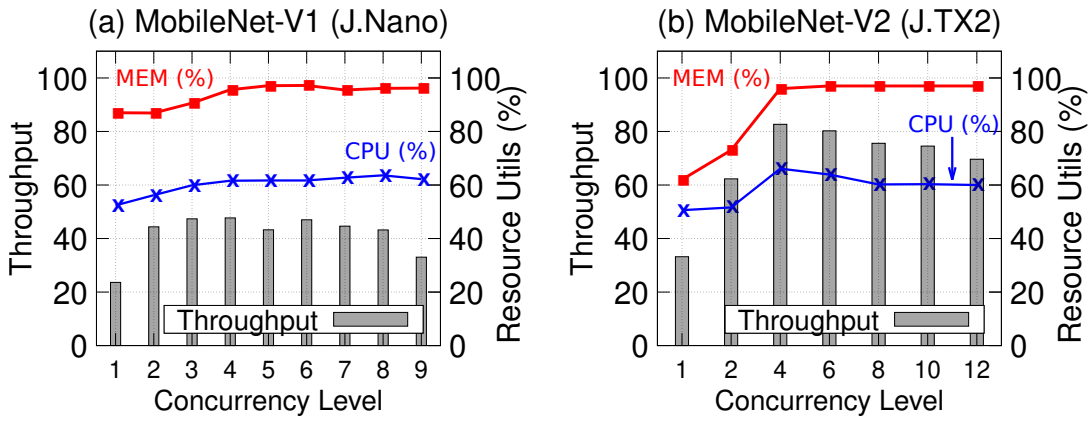


Figure 4.9: Resource utilization and throughput changes with CME (PyTorch). `J.Nano` uses a batch size of 4, and `J.TX2` employs a batch size of 8.

tributed to the fact `J.Nano`'s experiments were performed by disabling MXNet's `cuDNN auto-tune` [15] process as `J.Nano`'s memory size (4 GB) is insufficient to perform such optimization. Enabling or disabling `auto-tune` option can significantly impact DL inference throughput because, if this option is enabled, MXNet first runs a performance test to seek the best convolutional algorithm, and the selected algorithm is used for further inference tasks.

Input batch size and level of concurrency complement the performance gain as both the approaches rely on running multiple inferences at the same time. However, due to memory and CPU usage constraints, we cannot indefinitely increase both to maximize performance. In this study, 5 to 6 concurrent models with a batch size of 8 resulted in the highest throughput, after which increasing either of the two parameters results in lower performance.

The level of concurrency is directly related to the size of the model and the available memory in the system. `J.Nano` could run 8 (Inception-V3) to 25 (MobileNet-V1/V2) models concurrently on GPU while `J.TX2` and `J.Xavier` could run approximately 25 (Inception-V3) to 80 (MobileNet-V1/V2) models on their GPUs simultaneously, when working with a input batch size of 1. As we increase the batch size, the level of concurrency decreases as lesser memory becomes available. Fig. 4.9 shows the maximum throughput was highly correlated with memory utilization. After reaching the maximum throughput, the throughput was either decreased or stabilized with high memory utilization. It is worth noting that the

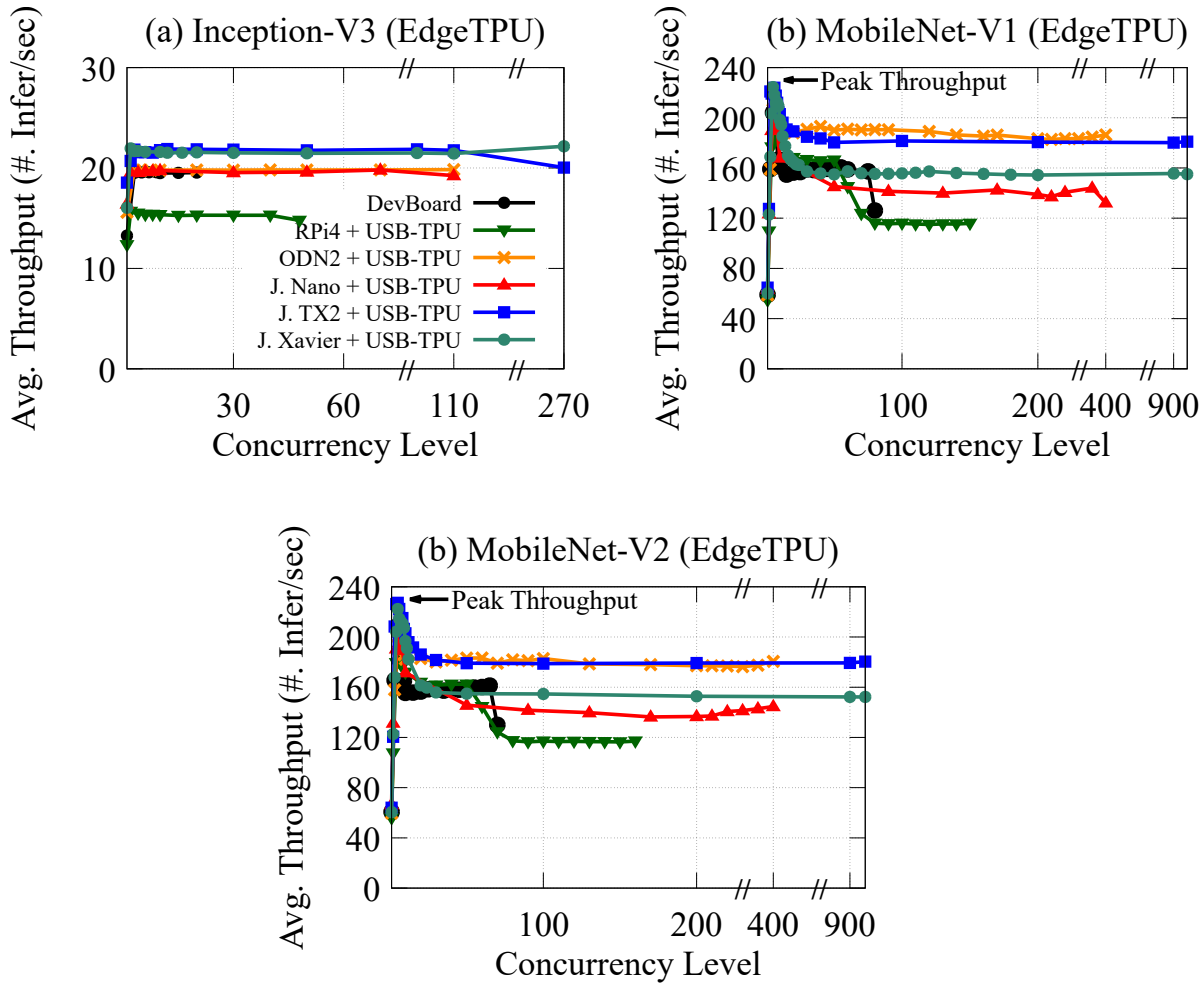


Figure 4.10: Results of CME measurement on EdgeTPU

high correlation between memory utilization and throughput increase is consistent with our observation reported in Fig. 4.5.

CME Evaluation Results on EdgeTPUs

The second CME evaluation is based on running concurrent models on EdgeTPUs. Fig. 4.10 reports DL inference throughput variations with increasing concurrency levels on EdgeTPUs (both `DevBoard` and `USB-Accelerator`). The result includes all the combinations of edge devices and `USB-Accelerator`.

Similar to the previous results on GPUs, CME on EdgeTPUs shows promising throughput improvement over single-tenancy cases. However, Inception-V3 showed significantly less improvement (1.3×) in performance, particularly when compared to MobileNet-V1/V2 (3.3×). This can be explained by considering the difference in the size of the model. The quantized Inception-V3 model is approximately 25MB in size (6× higher than MobileNet-V2), which means that it cannot be completely loaded into the `USB-Accelerator`'s 8MB of SRAM. Therefore, the model parameters have to be continuously swapped between the EdgeTPU and the host edge device. With the constant USB bandwidth, increasing concurrency levels show no improvement because most of the time is spent on this swapping operation. However, if the model size is small, e.g., 4MB of MobileNet-V2, the model can be fully loaded in EdgeTPUs' cache and does not require frequent operations of model parameter swapping, resulting in low USB IO overhead and significant throughput increases.

Since the `USB-Accelerators` rely on the host device's memory for storing model parameters, the number of concurrent models executed on the EdgeTPUs is very high given the smaller size of the quantized models. However, similar to the concurrency on GPUs, a higher concurrency level does not necessarily yield higher throughput. For MobileNet-V2, the concurrency level at which we achieve maximum throughput is 6. By increasing the concurrency level, we introduce more swapping operations, thereby resulting in degraded performance. This analysis suggests that, when using CME on EdgeTPU, model size and concurrency level should be carefully determined to increase the throughput.

Although minimal, there exists a difference in the throughput gain with different combinations of edge devices and `USB-Accelerator` despite using the same USB 3.0 interface. This difference could arise from the difference in CPU frequency. The CPU's frequency largely affects the efficiency (in terms of processing speed) of the system to spin off new threads and perform faster switching. Since devices like `J.TX2` and `ODN2` have higher CPU frequency (2 GHz for `J.TX2`, 1.8-1.9 GHz for `ODN2`) compared to other devices, CME on EdgeTPUs using these devices as host devices could result in better performance.

Regarding the varying concurrency levels, Fig 4.11 shows resource utilization changes with different concurrency levels measured from `USB-Accelerator` with `J.TX2` and `DevBoard`. The results show that

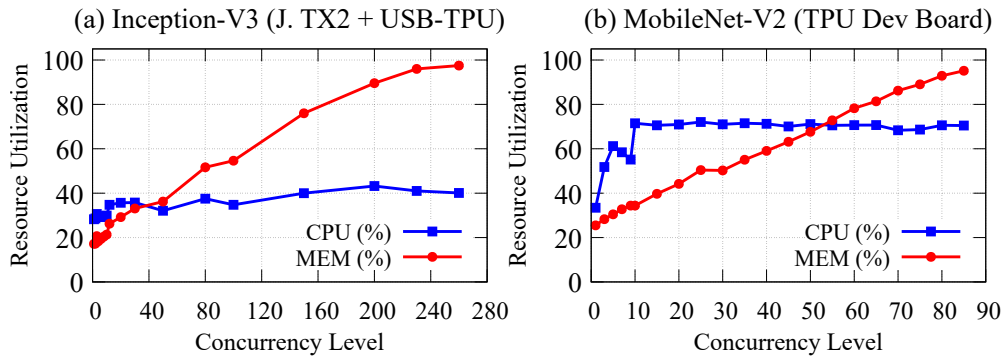


Figure 4.11: Resource utilization changes with increased concurrency level (EdgeTPUs)

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

The maximum throughput of nearly 230 inferences per second when running concurrent MobileNet-V1/V2 on EdgeTPUs is almost twice the maximum throughput achieved with CME on GPUs. However, this is not the case for larger models (i.e., Inception-V3). Thus, the maximum achievable throughput with Inception-V3 using EdgeTPUs is nearly half the value when using GPUs. This result highlights that careful consideration of model and device (GPU or EdgeTPU) is necessary to maximize the overall throughput.

CME on EdgeTPU Cluster

Based on the results discussed above, it is clear that connecting a `USB-Accelerator` to a stand-alone *edge* device and running concurrent models can significantly improve the overall throughput. This section further examines the impact in performance or throughput when running models concurrently on *more than one* `USB-Accelerators` simultaneously connected to the same device. The *edgetpu python API* was leveraged to load models in specific devices to ensure the equal number of models were running on all the `USB-Accelerators` connected.

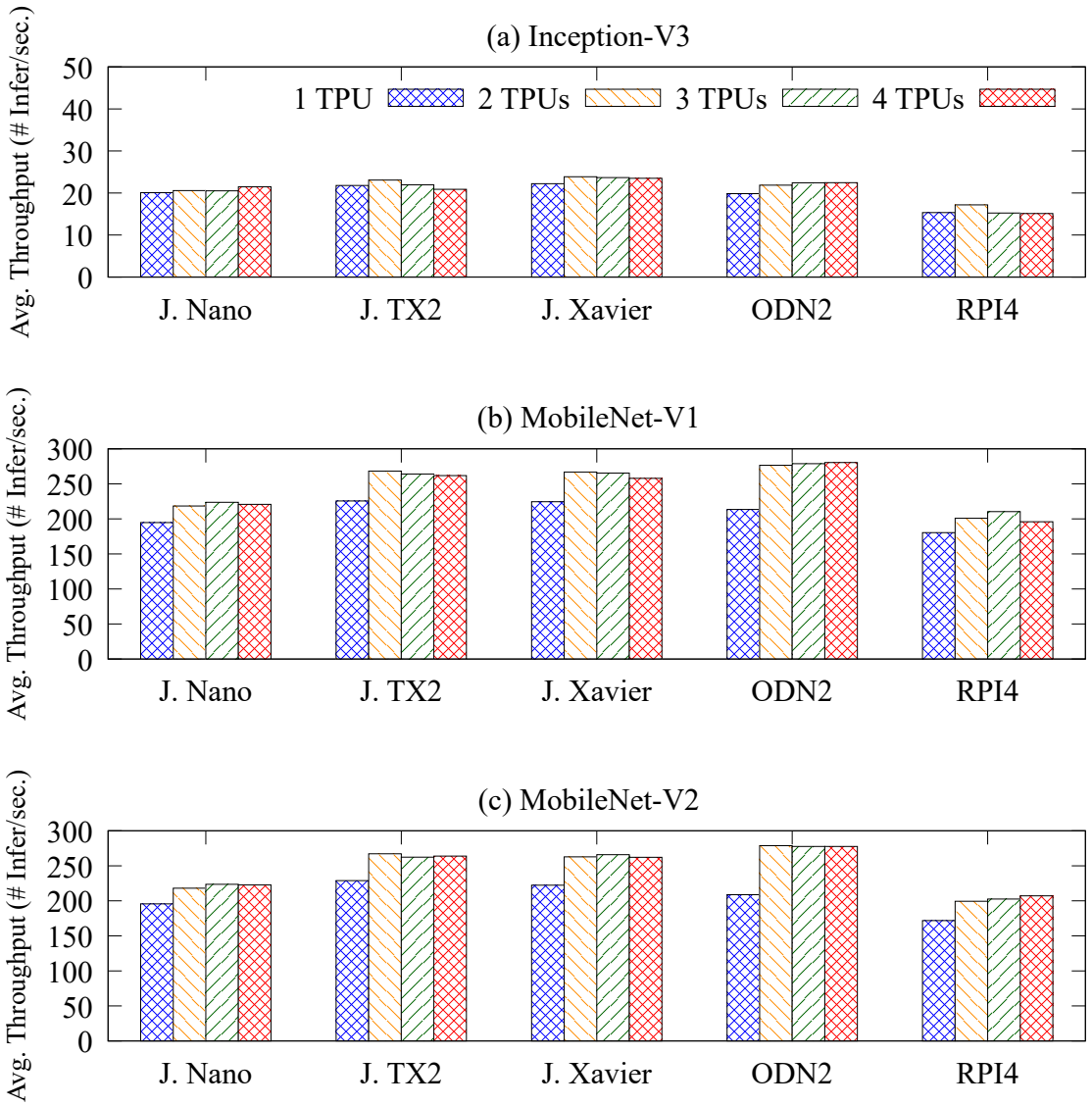


Figure 4.12: DL inference throughput variation with multiple `USB-Accelerators`

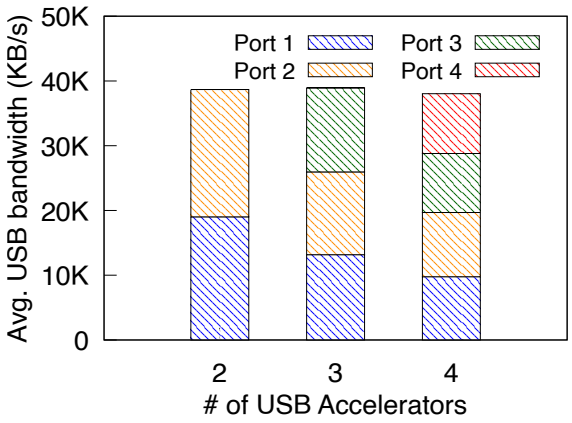


Figure 4.13: Variation in bandwidth allocation of USB ports when using using multiple `USB-Accelerators` in `J.TX2` while processing MobileNet-V2.

The first part of this experiment involved running each of the three models (Inception-V3, MobileNet-V1, MobileNet-V2) on 2 `USB-Accelerators` simultaneously on each device. Similar to 4.2.1, the experiment was repeated for multiple levels of concurrency. The throughput reported in fig. 4.12 is the maximum throughput that could be achieved with the respective number of `USB-Accelerators`. On average, there is a 15-30% increase in throughput going from 1 `USB-Accelerator` to 2 `USB-Accelerators` for models MobileNet-V1 and MobileNet-V2 across all devices. This increase is expected as two models could simultaneously be processed in the 2 `USB-Accelerators` while for 1 `USB-Accelerator` the jobs had to wait for the single TPU to finish the current task before moving on to the next task. The improvement isn't 2× as desired because the USB ports in all the devices on which the experiments were performed have an internal shared hub. This shared hub introduces some delay (due to its serial property) in data transfer, thereby increasing the overall latency.

As shown in Figure 4.12, there is hardly any improvement in throughput when using Inception-V3. As mentioned earlier, Inception-V3 is 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, limits the parallelism that could have been achieved from multiple `USB-Accelerators`. The `USB-Accelerators` are always waiting for the USB data transfer at all times.

The bottleneck imposed by the shared hub can be realized more prominently when dealing with more than two USB-Accelerators. The results from repeating the same experiment with 3 and 4 USB-Accelerators reveal that there is no advantage to using more than 2 USB-Accelerators, and it would be cost-ineffective to do so. Fig. 4.13 shows the variation in overall USB bandwidth available to each of the USB-Accelerator in a cluster, when running MobileNet-V2, connected to J.TX2. We can observe that, since the four USB ports share a common internal hub, the bandwidth available to each USB-Accelerator decreases with every addition of an USB-Accelerator. This reduced data transfer rate directly hampers the overall performance of the USB-Accelerators and seems to completely negate the benefits of having extra processing power (EdgeTPU).

Lastly, it is worth noting that that J.TX2 has just one USB (3.0) port, and only 2 out of 4 USB ports in RPi4 are USB 3.0. Several power-related issues were encountered when using a simple USB hub that drew power from the board. Therefore, an *externally powered* USB hub had to be used to be able to connect and extensively use multiple USB EdgeTPUs with J.TX2 and RPi4.

4.2.2 Evaluation Results with Dynamic Model Placements (DMP)

This section provides an evaluation of the dynamic model placement (DMP) technique for AI multi-tenancy on edge devices and EdgeTPUs. DMP allows running multiple DL models simultaneously by placing one model on an edge device's resource (CPU or GPU) and the other model on EdgeTPUs. Because USB-Accelerator can be attached 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 as well as 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, in this evaluation, the focus is 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 explicitly to CME for AI multi-tenancy?

Three DL models (Inception-V3, MobileNet-V1, and MobileNet-V2) are used in this section because they can execute inferencing on all computing resources in edge devices and `USB-Accelerators`. The equation-(3.1) is changed to correctly calculate the throughput with DMP. Specifically, the number of inferences for DMP is calculated as the number of inferences performed on the GPU combined with the number of inferencecs on TPU. Like CME, results from `RPi4` and `ODN2` have been excluded in this section because no obvious benefits could be observed. Specifically, CPUs 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` could be even lower (about 10%) than EdgeTPU-only inference throughput. Finally, MxNet and PyTorch are the frameworks selected to run on GPUs, while TensorFlow-Lite is the obvious framework for EdgeTPU.

Evaluation Procedure

Again, multi-threading techniques were utilized in order to run models on separate processors (GPU and TPU). Two separate threads are initiated with one thread running the workload for GPU while the other thread running the workload for TPU. In order to achieve maximum throughput, we need to ensure that both the models run concurrently on the system for most of the time. Since each model has a different execution time, which also varies with the type of processor, we cannot not use a predefined time interval. Therefore, a better mechanism was devised where both the threads would first perform an inference and notify each other of the their execution time. Once, each thread receives the execution time of the other thread, an *LCM* (Lowest Common Multiple) of the two execution times is calculated which becomes the *fixed* time interval over which the models will execute variable number of inferences on different processors concurrently. Choosing an LCM ensures that the inferencing tasks start and end at the same time on the two processors, thereby providing near optimal throughput. Due to some round off errors when performing the LCM, the two models would sometimes finish at a different time. Such

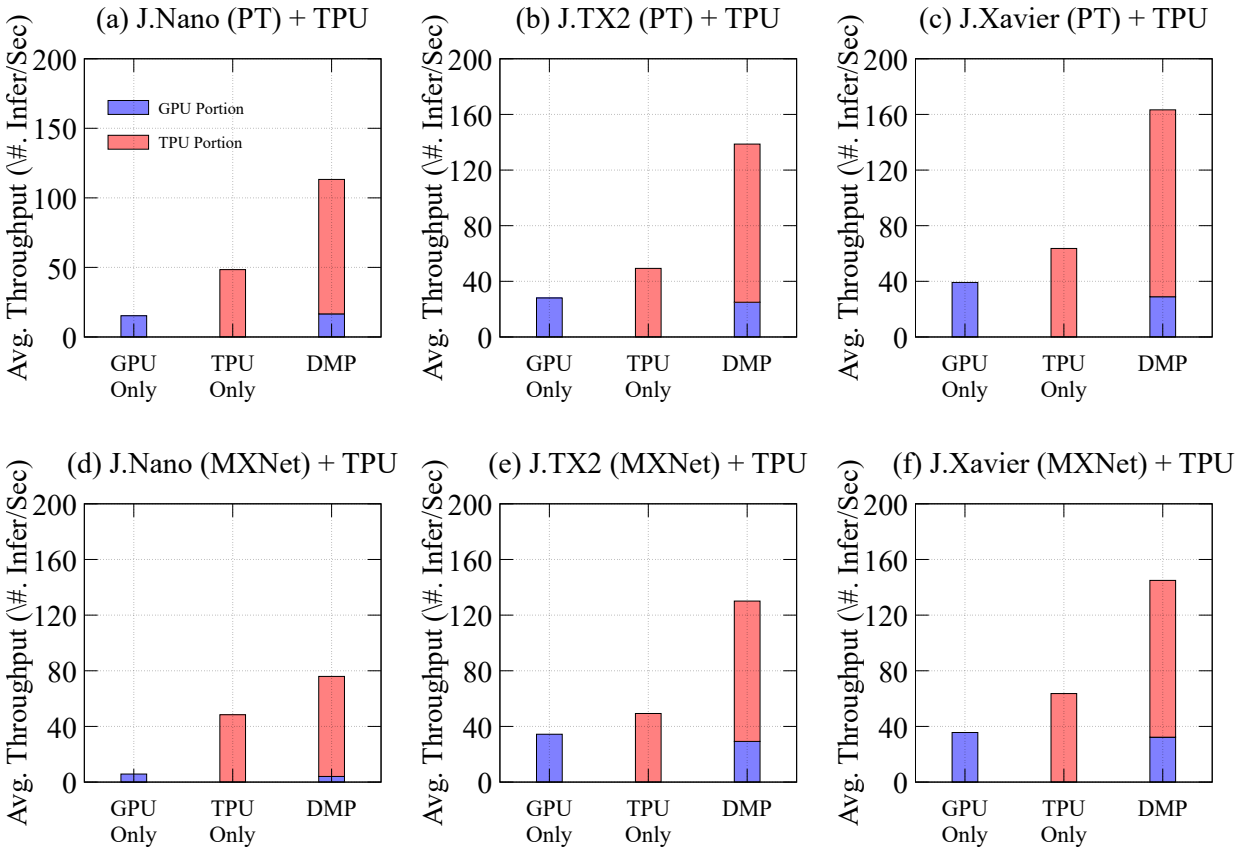


Figure 4.14: Throughput improvement compared to GPU only and TPU only.

results where the difference between the execution time of the two models exceeded by one second were discarded.

The ideal configuration, specifically concurrency level and batch size, is derived from the previous experiments(Section 4.2.1, Section 4.1.2). Therefore, instead of running a single model on each of the processor, we choose the best concurrency level and best batch size for the models executing on the two processors.

DMP evaluation results on GPU and single EdgeTPU

Fig. 4.14 shows DMP's DL inference throughput improvement against the single-tenancy cases. The "GPU only" and "TPU only" throughput is the average throughput of the three models (Inception-V3,

MobileNet-V1, MobileNet-V2) on GPU and `USB-Accelerator` respectively under single-tenancy. All three GPU-devices `J.Nano`, `J.TX2`, and `J.Xavier` show substantial increase in throughput with DMP. In particular, `J.Nano` shows a 13.1× improved throughput over the single-tenancy on GPU and 2.3× increased throughput over the single-tenancy on EdgeTPU. `J.TX2`'s throughput improvement is 4.9× (for GPU) and 2.8× (for EdgeTPU). Similarly, for `J.Xavier`, the improvement is 4.2× over GPU-only and 2.6× over TPU-only throughput. However, this throughput improvement can be due to leveraging both CME and DMP.

Fig. 4.15 reports the throughput comparison between (ideal) CME results 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 (from Section 4.2.1). The results are from `J.Nano` and `J.TX2` when using PyTorch (for GPU) and TFLite (for EdgeTPU). 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 and `J.Nano` with DMP showed 25.3% and 34.6% lower throughput than the ideal upper bounds with CME. These differences can be attributed to the resource contention and resource limits in the edge devices.

To understand the gap between the DMP's throughput and ideal throughput, further analysis on the resource consumption is performed. Fig. 4.16 shows the resource utilization (CPU, memory, USB IO) between the ideal sum of CME on GPU/EdgeTPU and DMP. As shown in the figure, the ideal throughput often cannot be achievable with current HW specifications. Specifically, to reach such high throughput, CPU and memory utilization should exceed the HW limitations (more than 100%). Moreover, similar to the CME analysis, memory is identified as a critical resource when enabling DMP. Specifically, memory utilization reaches 100% with DMP, but CPU utilization does not. Based on this observation, the DL inference throughput, when the memory resource is saturated, can be the empirical performance upper bound when enabling DMP. In addition, the throughput could be impacted by resource contention because the shared resources (e.g., memory and CPU) are needed to manage multiple models running on different resources. The decreased USB IO utilization (about 8% to 15%) with DMP (Fig 4.16(c)) is because

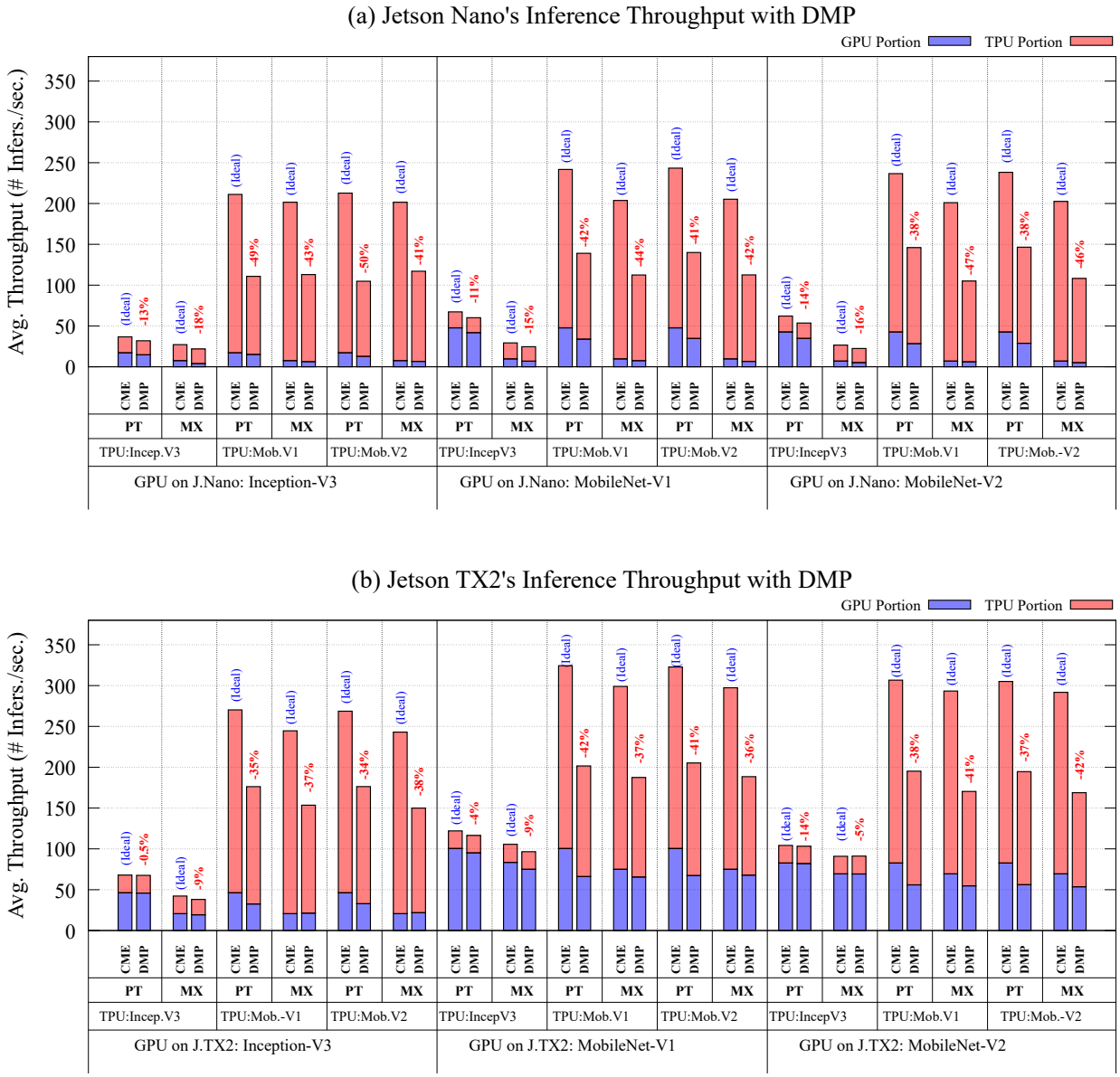


Figure 4.15: DL inference throughput comparison between (ideal) results from CME and DMP. The (ideal) results from CME are calculated by the sum of CME throughput on GPU and CME throughput on EdgeTPU, which were measured separately.

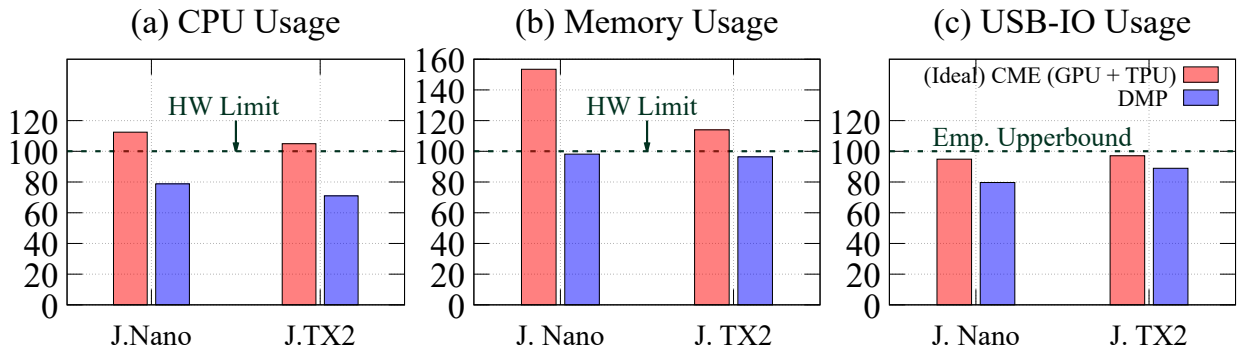


Figure 4.16: Resource usage comparison between (ideal) sum of CME on GPU/EdgeTPU and DMP.

of such resource contentions, and the reduced USB IO utilization can decrease DL inference throughput from EdgeTPU.

DMP evaluation results with reduced workload on GPU

Based on the above results, an apparent finding is that allocating more resources to the EdgeTPU might lead to higher overall throughput. Figure 4.17 reports the results from leveraging DMP with reduced workload on GPU. Only 1 model (i.e concurrency level is 1) is run on the GPU to mitigate resource-contention in the device. It has to be noted that both the GPU and the EdgeTPU are running the same model in Figure 4.17

The result shows that, for lighter models (MobileNet-V1/V2), the EdgeTPU's throughput increase quickly before reaching the peak while GPU throughput keeps decreasing due to resource-contention. But the decreasing speed is much slower compared to EdgeTPU. Please note that the reason of `J.TX2` and `J.Xavier` can't reach higher concurrency level than `J.Nano` for Inception-V3 is because a batch size (which is the optimal size) of 32 has been employed for `J.TX2` and `J.Xavier` while the permissible (in terms of available memory) batch size for `J.Nano` is just 4.

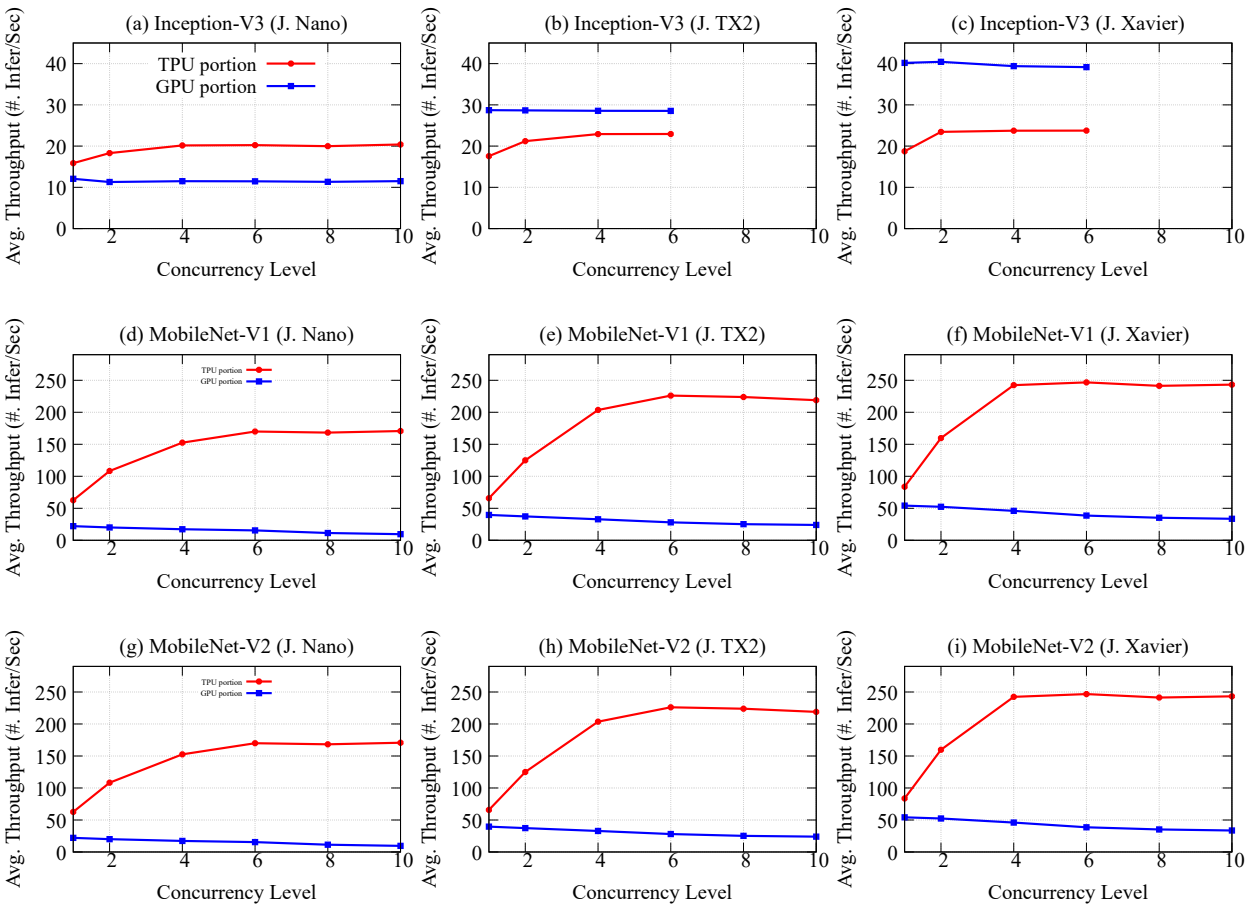


Figure 4.17: Overall impact on throughput with reduced GPU workload.

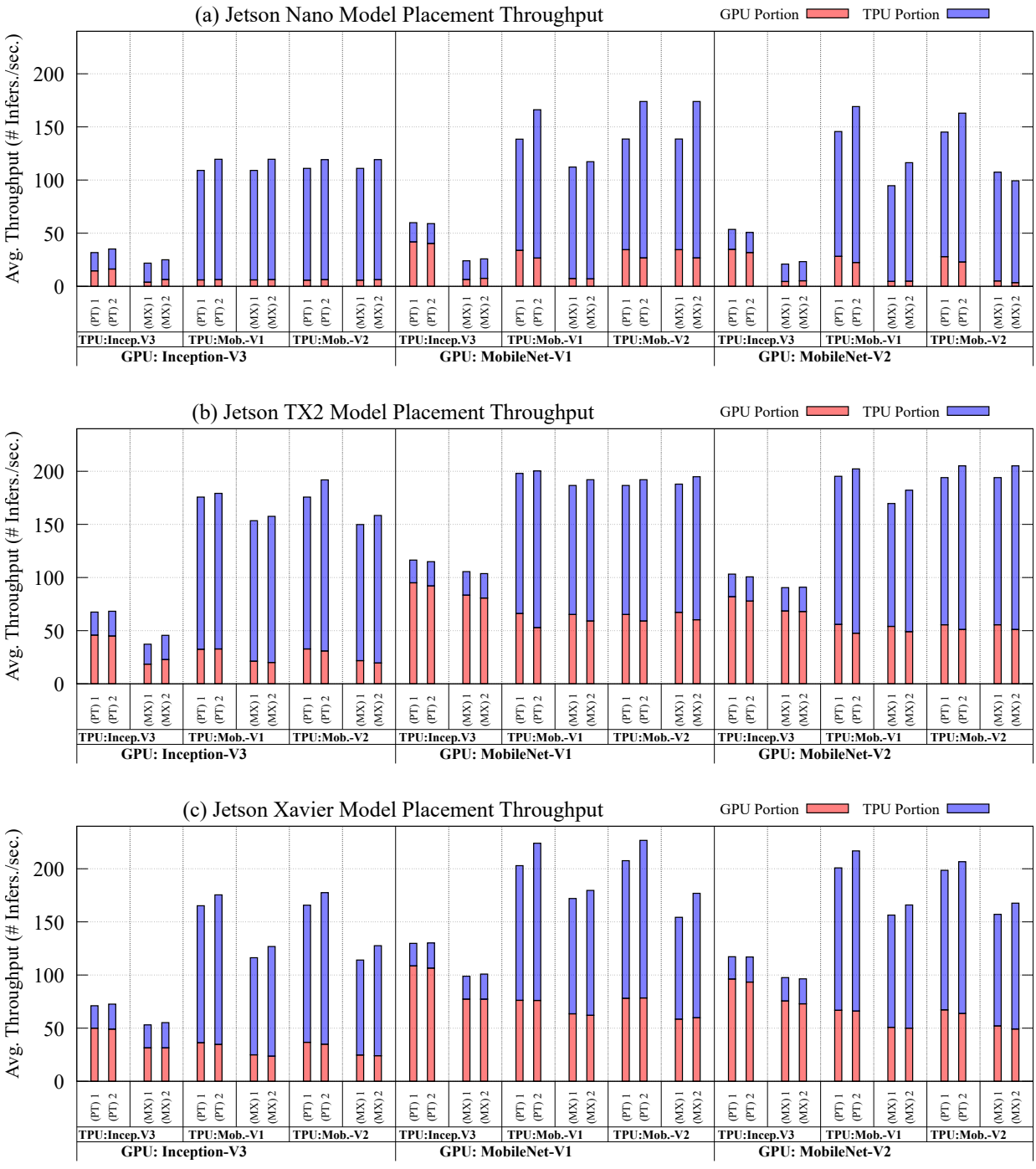


Figure 4.18: Performance variation with DMP on EdgeTPU cluster.

DMP evaluation results on GPU and EdgeTPU cluster

Fig. 4.18 provides a comparison of performance when using DMP on 1 EdgeTPU against 2 EdgeTPUs. As seen in Section 4.2.1, there are no advantages to using more than 2 EdgeTPUs.

Very similar to results from CME, the improvement in throughput when using 2 EdgeTPUs over 1 EdgeTPU is marginal (less than 10% on average). In addition to the three processors contesting for memory and other system resources, the shared hub architecture of USB ports limits the performance of `USB-Accelerators`.

4.2.3 Summary

Regarding CME on GPUs, edge devices with GPU resources showed significant throughput improvements. However, high concurrency levels do not necessarily result in maximizing DL inference throughput. Both concurrency level and batch sizes can change the DL inference throughput so that both configurations should be carefully determined when enabling CME on GPUs. The analysis suggests that the maximum throughput with CME on GPU can be realized when the memory utilization reached 100%. Furthermore, the throughput benefits of CME on GPUs can vary across different DL frameworks and edge devices. Specifically, the benefit can be limited, especially when enabled with MXNet on devices with small memory sizes.

On EdgeTPUs, considerable performance benefits were observed when enabling CME on EdgeTPUs along with high concurrency levels. Similar to CME on GPUs, high concurrency levels do not necessarily result in the maximum DL inference throughput. Specifically, if the DL model size cannot fit in the small cache (8MB) of EdgeTPUs, the benefits of using CME can be limited. Moreover, when using small DL models (e.g., MobileNet-V1/V2), employing the lower concurrency levels often results in the maximum performance gain. This observation strongly suggests that the techniques for minimizing model size (e.g., quantization and model compression) will be critical for the throughput improvement. In addition, using 2 TPUs can result in some throughput improvement but the improvement becomes negligible with more than 2 TPUs. The shared bus architecture of USB ports severely limits the performance gain.

This evaluation also confirms that DMP considerably improves the overall DL inference throughput over single-tenancy cases by leveraging GPU and EdgeTPU resources simultaneously. While the DMP's maximum throughput showed 25% to 35% lower throughput than the ideal cases, such differences are mainly due to the HW limitations. Furthermore, memory can be a critical resource factor when enabling DMP and can be saturated when reaching the maximum throughput. Similar to the results from CME with multiple EdgeTPUs, DMP with one GPU and multiple EdgeTPUs shows very marginal improvement over one GPU and one EdgeTPU case. Higher resource contention from using more than one EdgeTPU inhibits any kind of performance gain.

Chapter 5

Conclusion

This study focused on investigating system approaches that maximize the DL inference throughput on *resource-constrained* edge devices. The study began with the evaluation and characterization of various DL models' performance and behavior for image classification tasks on edge devices and AI accelerators with CPU, GPU, and EdgeTPU. With the characterization results from the first step, three system approaches were evaluated for maximizing DL inference throughput on various edge device settings. *Batched inferencing* is the approach for maximizing the throughput with DL single tenancy use cases. GPU-equipped devices showed significant throughput improvement with batched inferencing as multiple images could be processed in parallel on the GPU resources. Then, the feasibility and effectiveness of multi-tenancy at the edge was explored. Specifically, two approaches were applied - CME (Concurrent Model Executions) and DMP (Dynamic Model Placements). CME exploits the available system resources (CPU, memory, GPU) to load more models into the system and process multiple batches of inputs for each model in parallel. DMP, on the other hand, relies on making use of available computing capabilities by placing models on different processors (GPU or EdgeTPU) and processing inputs at both the processors simultaneously. Both the techniques were viable and successful at improving the system's overall throughput, including GPU and EdgeTPU, by a significant factor.

However, the limitations of the three approaches were also observed. In the case of batched inferencing, the performance improvements start decreasing once the batches' size exceeds the number of

GPU cores. Besides, due to limited memory available at the edge, there is a limit to the number of input images that can be simultaneously loaded in the memory. For CME and DMP with multi-tenancy, we start getting diminishing returns once the number of concurrently processed models exceeds the number of concurrent threads (or cores) supported by the CPU. System memory also becomes a bottleneck as we increase the number of concurrent models. Finally, since USB bandwidth drives the rate at which `USB-Accelerators` can process models, multi-tenancy on EdgeTPUs can show performance gain only when fewer data transfers (because of model parameters swapping) are involved. Inherently sequential hardware design, such as shared USB hubs, is also a restricting factor when using multiple `USB-Accelerators` simultaneously.

The results provided in this study show that multi-tenancy at the edge is a promising paradigm to improve the performance of DL tasks at the edge. 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.

Bibliography

[1] Martín Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, GA, USA, 2016.

[2] Mário Almeida et al. "EmBench: Quantifying Performance Variations of Deep Neural Networks across Modern Commodity Devices". In: *CoRR* abs/1905.07346 (2019). arXiv: 1905.07346. URL: http://arxiv.org/abs/1905.07346.

[3] *Azure AI*. https://azure.microsoft.com/en-us/overview/ai-platform/. [ONLINE]. 2021.

[4] Jacob Benesty, Jingdong Chen, and Yiteng Huang. "On the Importance of the Pearson Correlation Coefficient in Noise Reduction". In: *IEEE Transactions on Speech and Audio Processing* 16.4 (2008), pp. 757–765.

[5] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*. 2007, pp. 119–128. DOI: 10.1109/INM.2007.374776.

[6] Jiasi Chen and Xukan Ran. "Deep Learning With Edge Computing: A Review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[7] Tianqi Chen et al. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *CoRR* abs/1512.01274 (2015). arXiv: 1512.01274. URL: http://arxiv.org/abs/1512.01274.

[8] Tianqi Chen et al. "{TVM}: An automated end-to-end optimizing compiler for deep learning". In: *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 2018, pp. 578–594.

[9] Yu Cheng et al. "A Survey of Model Compression and Acceleration for Deep Neural Networks". In: *CoRR* abs/1710.09282 (2017). arXiv: 1710.09282. URL: http://arxiv.org/abs/1710.09282.

[10] Stuart Clayman et al. "The dynamic placement of virtual network functions". In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–9. DOI: 10.1109/NOMS.2014.6838412.

[11] *Cloud AI – Google Cloud*. https://cloud.google.com/products/ai/. [ONLINE]. 2021.

[12] *Coral Dev Board datasheet*. https://coral.ai/docs/dev-board/datasheet/. [ONLINE]. 2021.

[13] *Coral USB Accelerator datasheet*. https://coral.ai/docs/accelerator/datasheet/. [ONLINE]. 2021.

[14] Koustabh Dolui and Soumya Kanti Datta. "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing". In: *2017 Global Internet of Things Summit (GIoTS)*. 2017, pp. 1–6. DOI: 10.1109/GIOTS.2017.8016213.

[15] *Environment Variables – MXNet v1.7.0*. https://mxnet.apache.org/versions/1.7.0/api/faq/env_var. [ONLINE]. 2021.

[16] Jian Guo et al. "GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing". In: *Journal of Machine Learning Research* 21.23 (2020), pp. 1–7. URL: http://jmlr.org/papers/v21/19-429.html.

[17] Ramyad Hadidi et al. "Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices". In: *IEEE International Symposium on Workload Characterization (IISWC)*. 2019.

[18] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV].

[19] Kim M. Hazelwood et al. "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective". In: *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. IEEE Computer Society, 2018, pp. 620–629.

[20] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385v1*. 2015.

[21] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA, 2016.

[22] Yihui He et al. "AMC: AutoML for Model Compression and Acceleration on Mobile Devices". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*. Vol. 11211. Lecture Notes in Computer Science. Springer, 2018, pp. 815–832.

[23] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[24] Chuang Hu et al. "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge". In: *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 2019, pp. 1423–1431.

[25] Gao Huang et al. "Densely Connected Convolutional Networks". In: *arXiv:1608.06993*. 2017.

[26] Gao Huang et al. "Densely Connected Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA, 2017.

[27] Forrest N. Iandola et al. "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design". In: *arXiv:1602.07360v4*. 2016.

[28] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size". In: *CoRR* abs/1602.07360 (2016). arXiv: 1602.07360.

[29] *IBM Watson Machine Learning*. https://www.ibm.com/cloud/machine-learning. [ONLINE]. 2021.

[30] *INA219 – 26V, 12-bit, i2c output current/voltage/power monitor*. https://www.ti.com/product/INA219. [ONLINE]. 2021.

[31] *Intel Neural Compute Stick*. https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. [ONLINE]. 2021.

[32] *Jetson Nano | Nvidia Developer*. https://developer.nvidia.com/embedded/jetson-nano. [ONLINE]. 2021.

[33] Xiaotang Jiang et al. "MNN: A Universal and Efficient Inference Engine". In: *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. 2020.

[34] Yiping Kang et al. "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge". In: *SIGARCH Comput. Archit. News* 45.1 (2017), pp. 615–629. ISSN: 0163-5964. DOI: 10.1145/3093337.3037698. URL: https://doi.org/10.1145/3093337.3037698.

[35] *kerascv 0.0.40*. https://pypi.org/project/kerascv/. [ONLINE]. 2021.

[36] Wazir Zada Khan et al. "Edge computing: A survey". In: *Future Generation Computer Systems* 97 (2019), pp. 219–235. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.02.050. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18319903.

[37] In Kee Kim et al. "Orchestra: Guaranteeing Performance SLAs for Cloud Applications by Avoiding Resource Storms". In: *17th International Symposium on Parallel and Distributed Computing, ISPDC 2018, Geneva, Switzerland, June 25-28, 2018*. IEEE, 2018, pp. 53–60.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *NIPS Proceedings of the 25th International Conference on Neural Information Processing Systems*. 2012.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2012.

[40] En Li et al. "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing". In: *IEEE Transactions Wireless Communications* 19.1 (2020), pp. 447–457.

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

[42] Leandro Ariel Libutti et al. "Benchmarking Performance and Power of USB Accelerators for Inference with MLPerf". In: *International Workshop on Accelerated Machine Learning (AccML)*. 2020.

[43] Shaoshan Liu et al. "Edge Computing for Autonomous Driving: Opportunities and Challenges". In: *Proceedings of IEEE* 107.8 (2019), pp. 1697–1716.

[44] Marcia Sahaya Louis et al. "Towards deep learning using tensorflow lite on risc-v". In: *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*. Vol. 1. 2019, p. 6.

[45] *Machine Learning on AWS*. `https://aws.amazon.com/machine-learning/`. [ONLINE]. 2021.

[46] Thaha Mohammed et al. "Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading". In: *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 854–863.

[47] *NVIDIA Jetson Linux Developer Guide : Clock Frequency and Power Management*. https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/clock_power_setup.html. [ONLINE]. 2021.

[48] *Nvidia Jetson TX2*. https://developer.nvidia.com/embedded/jetson-tx2. [ONLINE]. 2021.

[49] *NVIDIA Jetson Xavier NX*. https://developer.nvidia.com/embedded/jetson-xavier-nx. [ONLINE]. 2021.

[50] *NVPModel – Nvidia Jetson TX2 Development Kit*. https://www.jetsonhacks.com/2017/03/25/nvpmodel-nvidia-jetson-tx2-development-kit/. [ONLINE]. 2021.

[51] *ODROID-N2*. https://wiki.odroid.com/odroid-n2/odroid-n2. [ONLINE]. 2021.

[52] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Annual Conference on Neural Information Processing Systems (NeurIPS 2019)*. Vancouver, BC, Canada, 2019.

[53] *pi-ina219 1.4.0*. https://pypi.org/project/pi-ina219/. [ONLINE]. 2021.

[54] *Raspberry Pi 4*. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [ONLINE]. 2021.

[55] Ju Ren et al. "Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing". In: *IEEE Network* 31.5 (2017), pp. 96–105.

[56] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[57] Colin Samplawski et al. "Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators". In: *International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AUChallengeIoT)*. ACM, 2019.

[58] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA, 2018.

[59] Omid Setayeshfar et al. "ChatterHub: Privacy Invasion via Smart Home Hub". In: *IEEE International Conference on Smart Computing, SMARTCOMP 2021, Virtual, August 23-27, 2021*. IEEE, 2021, pp. 1–8.

[60] Weisong Shi and Schahram Dustdar. "The Promise of Edge Computing". In: *IEEE Computer* 49.5 (2016), pp. 78–81.

[61] Weisong Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[62] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556v6*. 2014.

[63] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*. San Diego, CA, USA, 2015.

[64] Ion Stoica et al. "A Berkeley View of Systems Challenges for AI". In: *CoRR* abs/1712.05855 (2017).

[65] Xian-He Sun and Yong Chen. "Reevaluating Amdahl's law in the multicore era". In: *Journal of Parallel and Distributed Computing* 70.2 (2010), pp. 183–188. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2009.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0743731509000884.

[66] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *arXiv:1512.00567v3*. 2015.

[67] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA, 2016.

[68] Martin Takác et al. "Mini-Batch Primal and Dual Methods for SVMs". In: *CoRR* abs/1303.2314 (2013). arXiv: 1303.2314. URL: http://arxiv.org/abs/1303.2314.

[69] *TensorFlow Lite – ML for Mobile and Edge Devices*. https://www.tensorflow.org/lite. [ONLINE]. 2021.

[70] *tf.Graph – TensorFlow v2.4.1*. https://www.tensorflow.org/api_docs/python/tf/Graph. [ONLINE]. 2021.

[71] *tf.hub – TensorFlow Hub*. https://www.tensorflow.org/hub. [ONLINE]. 2021.

[72] *torchvision 0.5.0*. https://pytorch.org/vision/. [ONLINE]. 2020.

[73] Kuan Wang et al. "HAQ: Hardware-Aware Automated Quantization With Mixed Precision". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 8612–8620.

[74] Yu Wang, Gu-Yeon Wei, and David Brooks. "A Systematic Methodology for Analysis of Deep Learning Hardware and Software Platforms". In: *Conference on Machine Learning and Systems (MLSys)*. 2020.

[75] Carole-Jean Wu et al. "Machine Learning at Facebook: Understanding Inference at the Edge". In: *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 2019, pp. 331–344.

[76] Ben Zhang et al. "The Cloud is Not Enough: Saving IoT from the Cloud". In: *7th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud '15, Santa Clara, CA, USA, July 6-7, 2015*. USENIX Association, 2015.

[77] Haotian Zhang et al. "Eye in the Sky: Drone-Based Object Tracking and 3D Localization". In: *ACM International Conf. on Multimedia (MM)*. Ed. by Laurent Amsaleg et al. Nice, France, 2019.

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

[79] Serena Zheng et al. “User Perceptions of Smart Home IoT Privacy”. In: *Proceedings of the ACM on Human-Computer Interaction* 2.CSCW (2018), 200:1–200:20.

[80] Zhi Zhou et al. “Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing”. In: *Proceedings of IEEE* 107.8 (2019), pp. 1738–1762.

[81] Zhiting Zhu et al. “Understanding the security of discrete GPUs”. In: *Proceedings of the General Purpose GPUs*. 2017, pp. 1–11.