Toward Building Intelligent In Situ Sensing Systems: From Edge AI to IoT

by

Jianwei Hao

(Under the Direction of In Kee Kim)

Abstract

The advent of edge computing and IoT technologies has led to the ubiquitous presence of IoT-edge sensing systems, crucial for revolutionized data collection and analysis across numerous domains. However, the effective utilization and deployment of IoT-edge systems face significant challenges, particularly in terms of energy management and computational resource constraints.

This dissertation addresses these challenges through a comprehensive approach that integrates three critical research areas to enhance the sustainability and computing capabilities of IoT-edge systems. Firstly, I introduce an innovative energy scheduling method for environmental sensors that harnesses solar power, enabling sustained operation in areas lacking conventional power infrastructures. It leverages an AI energy prediction model and a scheduler to optimize sensing activities in harsh, remote environments. Secondly, the study enhances edge computing capability by incorporating deep learning (DL) models and advanced system techniques leveraging AI multi-tenancy and heterogeneous AI accelerators. These techniques significantly increase the processing speed, inference throughput, and efficiency of data processing, reducing both network and computational delays. Finally, the research extends to the environmental monitoring of coastal wetlands with a tailored IoT-edge soil monitoring system that employs energy optimization and low-cost sensor calibration to effectively monitor soil properties under challenging field conditions.

This work integrated approaches improving the intelligent aspects of IoT-edge sensing systems and contributes to the effective monitoring and conservation of critical ecosystems, demonstrating a balanced focus on innovation and environmental sensing.

Index words: IoT, edge, Energy Management, DL, AI

Toward Building Intelligent In Situ Sensing Systems: From Edge AI to IoT

by

Jianwei Hao

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

Doctor of Philosophy

Athens, Georgia

2024

Toward Building Intelligent In Situ Sensing Systems: From Edge AI to IoT

by

Jianwei Hao

Major Professor: In Kee Kim

Committee: Lakshmish Ramaswamy
Deepak Mishra

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
August 2024

Acknowledgments

I extend my deepest gratitude to my dissertation advisor, Dr. In Kee Kim, whose unwavering support, guidance, and mentorship were instrumental throughout this research journey. His expertise, patience, and encouragement were invaluable, and I am profoundly grateful for his dedication and commitment.

I am also indebted to the members of my dissertation committee, Drs. Lakshmish Ramaswamy and Deepak Mishra, for their insightful feedback, constructive criticism, and scholarly contributions, which enriched the quality of this work and broadened my perspectives.

I am profoundly grateful to my family. Especially my wife, Ruiling Liu, and my daughter, Tian Hao, for their unwavering support, patience, and encouragement throughout this challenging journey of completing my dissertation. I am also deeply grateful to my friends and colleagues for their encouragement, understanding, and support throughout this rewarding journey.

Finally, I acknowledge the countless individuals whose work and contributions have shaped the field of IoT-edge sensing systems. Their dedication to advancing knowledge and addressing critical issues has paved the way for my research endeavors.

This dissertation would not have been possible without the collective support, guidance, and encouragement of all those mentioned above. Thank you for being part of this journey.

Contents

Chapter 1

Introduction

The advancements in edge computing and Internet of Things (IoT) technologies have led to the widespread adoption of IoT-edge sensing systems, which play a crucial role in various applications such as data monitoring, collection, storage, and analysis. These systems are important in emerging distributed applications like autonomous vehicles, precision agriculture, and drone-based surveillance.

Despite their utility, IoT-edge sensing systems face several challenges that need to be addressed. One significant challenge is energy management, especially in harsh environments where stable power infrastructure is lacking. While renewable energy sources like solar power are commonly used, ensuring a stable energy supply remains a challenge. Additionally, facilitating the execution of complex deep learning (DL) models on resource-constrained edge devices poses another challenge. DL models are highly accurate but are resource-intensive, making their deployment on edge devices challenging. Moreover, IoT-edge systems often operate in harsh environments like coastal wetlands, where continuous monitoring of soil health is essential. Traditional soil sampling methods are costly and labor-intensive, necessitating the development of IoT sensors capable of accurately monitoring soil parameters.

To address these challenges, researchers are focusing on developing novel resource and energy management algorithms tailored to IoT-edge sensing systems. This research aims to build sustainable IoT-edge systems capable of operating in challenging environments with unreliable power sources and resource-constrained devices.

1.1 Background

With the advancements in edge computing and IoT technologies, IoT-edge sensing systems have become ubiquitous [1], playing a significant role in data monitoring, collection, storage, and analysis [2]. IoT-edge sensing systems are widely used in emerging distributed applications, including autonomous vehicles, precision agriculture, and drone-based surveillance.

Environmental sensors need to be powered with energy storage units (ESUs). However, harsh environments commonly lack stable power infrastructure (or power grid), elevating energy management to be a primary research challenge [3]. While leveraging renewable energy sources is a widely adopted ap-

proach, these energy sources do not always ensure a stable energy supply, demanding a high requirement for sustainable operation capability in the sensing system [4].

Furthermore, as artificial intelligence (AI) and DL technologies continue to expand across diverse domains, the need for facilitating the execution of DL models on IoT-edge devices is growing rapidly [5]. These DL models are known to be highly accurate but are typically resource and energy-intensive due to the large number of operational parameters in their architecture [6]. This makes their operation on resource-constrained edge devices quite challenging. Despite attempts to overcome this limitation, existing research has not fully addressed the challenge of efficiently deploying these DL models on such devices [7, 8, 9]. We further discuss these challenges in IoT-edge sensing systems.

These IoT-edge systems are often deployed and operated in harsh environments e.g. coastal wetlands, necessitating them to be sustainable and resilient [10]. Coastal wetlands are an important natural carbon storage system, containing approximately half of the world's blue carbon [11]. However, coastal wetland ecosystems are facing increasing *vulnerabilities* from threats, such as coastal development, invasive species, rising global temperatures, and sea-level rise [12, 13]. Therefore, it is imperative to preserve such wetland areas by continuously monitoring and improving soil health in the coastal wetlands [14]. The traditional soil sampling methods are not only costly but also labor-intensive and time-consuming [15]. Due to the advancements in IoT (Internet-of-Things) technologies, various soil sensors have been developed. These sensors are now widely used in various domains, such as agricultural systems [16, 17, 18]. Given their capability to monitor crop yields and diverse soil parameters, they present a promising approach for soil monitoring. Moreover, soil sensors are cost-effective, offering a more economical way than traditional soil sampling methods and proving more accurate than conventional remote sensing.

1.2 Research Challenges

To build a sustainable and effective IoT-edge system, there are three research challenges. These three research challenges involve energy management for energy harvesting (EH) systems, facilitating AI executions on resource-constrained edge devices, and *in situ* environmental sensing. the following subsections describe the details of three challenges.

1.2.1 Challenge 1: Energy Management for IoT-edge Systems

IoT-edge systems are often deployed in challenging environments without stable power supply infrastructure. To address this, EH techniques, which leverage renewable sources such as solar power, wind, and heat, are widely used approaches to supply energy for sensing and data processing operations [19]. Solar energy, in particular, is a popular choice, and many solar panels are compatible with edge computing boards like Raspberry Pi and various IoT sensors. Yet, the reliability of solar energy is contingent upon weather conditions. Sensing systems might encounter a series of rainy or cloudy days, leading to energy shortages and disrupted sensing operations. Consequently, EH systems often incorporate energy storage units (ESUs), like portable battery packs, linked directly to the sensors or computing boards. These units

recharge during optimal conditions (e.g., sunny days), ensuring the sensors have a stable energy supply. Moreover, these solar-powered EH IoT-edge systems often find themselves in extreme locations, such as wetlands and reservoirs, where they monitor environmental conditions [20]. These areas typically lack stable network connectivity [21], making it challenging to access timely weather forecasts from well-known services. This lack of real-time information complicates energy management and efficient power distributions, potentially causing frequent interruptions in data collection. Thus, intelligently managing energy while ensuring uninterrupted sensor operations within a constrained power budget remains a significant challenge for IoT systems.

1.2.2 Challenge 2: Facilitating AI Executions on Resource-constrained Devices

edge computing devices and AI accelerators have been increasingly developed in recent years [22]. There are various hardware, device, and accelerator configurations available on the market, such as Raspberry Pi, Jetson boards, and edge TPU accelerators. For the deployment of complex DL/AI models on these edge devices, it is essential for researchers and practitioners to understand the performance and execution characteristics of their models and devices [23]. Subsequently, they can optimize scheduling and resource allocation for DL/AI models to meet various performance goals, such as latency and service level objectives (SLOs) [24, 25, 26]. Unfortunately, there is a lack of comprehensive performance studies for facilitating AI/DL executions on various edge devices and accelerators. The comprehensive performance study should cover a broad set of factors related to DL/AI models, such as different frameworks, software libraries, and device-specific performance metrics. This analysis should involve various aspects, including but not limited to throughput, latency, power efficiency, thermal metrics, resource utilization, the implications of parallel and concurrent executions, as well as the performance variations introduced by heterogeneous accelerators. Through these comprehensive characterizations, researchers and practitioners can enhance system designs, advance hardware/accelerator development, and optimize model development. Therefore, the absence of a comprehensive performance study represents a significant barrier to advancing DL and AL models on edge devices.

1.2.3 Challenge 3: In Situ Environmental Sensing

When building an IoT system with soil sensors to monitor soil health in coastal wetlands, there are significant research and engineering challenges. Many of these challenges arise due to the unique environmental dynamics of the coastal wetlands [27]. For example, coastal wetlands often lack a stable power supply to support the IoT system's long-term sustainability and operability. To address this challenge, energy harvesting (EH) emerges as a promising approach as it generates electricity from natural resources [28, 29]. Specifically, solar power is a widely used energy source for these EH systems. Various solar panels are compatible with IoT computing boards, such as Raspberry Pi and Arduino, and with sensors. However, due to the weather dynamics, solar power can exhibit non-uniform energy generation patterns [30]. Thus, EH-based IoT systems need intelligent energy management for long-term operations at varying energy harvesting rates. Additionally, while current sensors can measure multiple soil parameters, there is, to

our best knowledge, no IoT sensor capable of directly measuring essential information for evaluating soil health. For example, soil health can be deduced from the variance and quantity of soil organic carbon (SOC) [31, 32], bulk density [33], and microbial biomass [34]. Therefore, these properties need to be estimated from other environmental parameters (*e.g.*, temperature, salinity, and pH) that can be collected by soil sensors. To this end, domain scientists should determine the most appropriate sensors (with respect to cost, accuracy, and data quality) that can measure environmental/soil parameters, and then they should create a (machine learning or empirical) model to infer SOC, bulk density, and/or microbial biomass for determining soil health.

1.3 Dissertation Research Design

This research aims to address the above research challenges by developing novel resource and energy management algorithms specifically tailored to IoT-edge sensing systems. My research focuses on building sustainable edge-IoT systems deployed in challenging environments and conditions with unreliable power sources, frequent network disconnectivity, and resource-constrained devices. Specifically, I aim to answer the following research questions (RQs).

RQ #1: In what ways can we formulate and implement intelligent and adaptive energy management algorithms and policies specifically tailored for EH IoT-edge sensing systems, with the overarching goal of guaranteeing prolonged and consistent operations in the long term?

RQ #2: When integrating multiple resource-constrained devices into IoT-edge sensing systems, what performance outcomes (throughput, latency, energy consumption) can we expect? How can we build novel management algorithms for DL models and edge devices/accelerators that can maximize the DL inference performance by leveraging the resource heterogeneity of edge devices and accelerators?

RQ #3: How to evaluate and select low-cost sensors? How to design a low-cost and sustainable IoT system for environmental sensing? How reliable is the low-cost sensors' data for soil properties?

To answer the RQs above, this dissertation outlines the approaches as shown in Fig. 1.1.

1. **Energy Scheduling.** I will describe the desig of a power management algorithm for an *in situ* IoT-edge device to dynamically adjust the energy allocation and sensors operation activities. A simulator will be developed to accelerate this work. To determine the energy allocation, we need to estimate the future energy gain. A more accurate energy predictor would facilitate the system better on energy allocation. Therefore, an accurate energy predictor will be designed and developed before the energy allocation. The energy allocation function will be designed and implemented on the minimizing-interval principle.

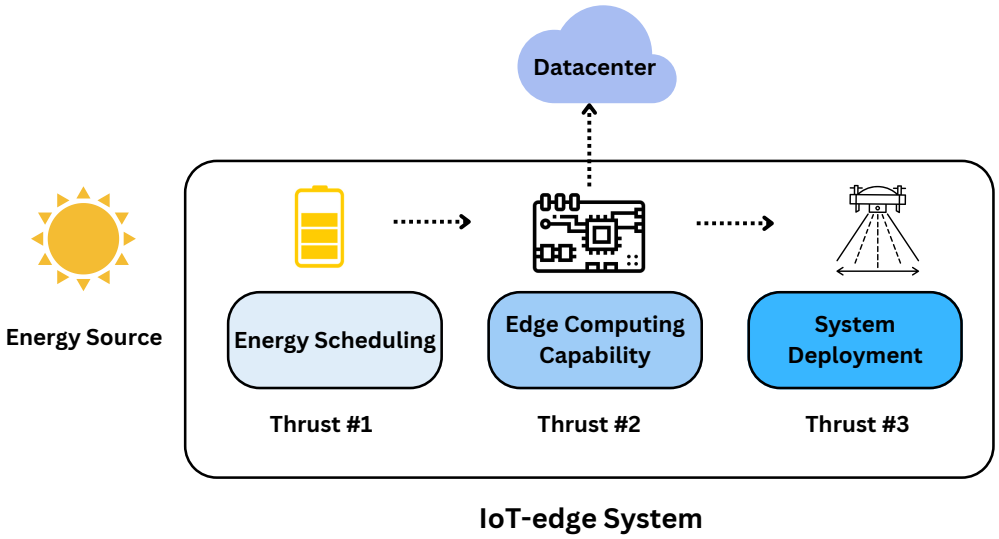


Figure 1.1: Research structure

2. **Edge Computing Capability.** I will thoroughly characterize the performance and behaviors of different edge devices and accelerators during DL inference tasks. Furthermore, I'll craft techniques and resource management strategies to optimize DL inference by harnessing different resource models across edge devices and DL frameworks. To this end, I will begin by employing well-established principles like parallel and concurrent executions to enhance DL inference performance. Ultimately, my goal in this thrust is to devise algorithms that can dynamically exploit the resource heterogeneity of edge devices and accelerators.

3. **IoT-Edge System Deployment.** Build a cost-effective, sustainable, field-ready IoT platform to monitor soil health in coastal wetlands. Following this, we delve into selecting and evaluating various soil sensors integrated into the platform. Moreover, a machine-learning model will be used to predict one of the soil health indicators e.g., soil bulk density, using data measured by the soil sensors.

1.4 Research Contributions

The most important contribution of this research is that I introduce an IoT-edge system for environmental sensing to ensure sustainability and performance. To this end, I design and develop three essential components in the energy management system, DL acceleration on edge devices, and *in situ* system de-

ployment, which address the implement of IoT-edge system to coastal marshland for facilitating domain science and data collection. More specifically, this research makes the following contributions.

The contributions of this dissertation are as follows:

1. A sustainable energy management system with accurate DC power gain prediction and dynamic energy scheduling algorithm. The DC power gain estimator in `DynaES` can accurately predict ESU's DC power gain by combining lightweight predictors, which are carefully determined by their accuracy, overhead, and power consumption. This prediction mechanism enables solar-powered EH sensors to perform stable operations without external support from online weather forecasting services. `DynaES`' scheduling algorithm shows effective utilization of harvested energy for sensors. This algorithm dynamically adjusts the intervals and frequency of sensing operations, ensuring high energy efficiency and preserving sensing data quality.

2. A thorough characterization and quantitative analysis of the performance with three systematic techniques for maximizing DL inference throughput. We propose and develop three techniques – *batch inferencing, CME (Concurrent Model Executions), and DMP (Dynamic Model Placements)* – to maximize DL inference throughput on various edge devices. In particular, batched inferencing is a throughput maximization approach for single DL tasks. Furthermore, CME and DMP maximize the inference throughput with AI multi-tenancy. Based on these techniques, we provide a thorough characterization and quantitative analysis of the performance (i.e., latency and throughput) of various edge devices and AI accelerators when running DL tasks for image classifications. We also provide a thorough analysis of the factors that affect the DL inference throughput.

3. Design of IoT system for low-cost, sustainable soil monitoring. We design an *in situ*, low-cost IoT system for soil health monitoring. This system offers long-term, cost-effective, and non-intrusive methods for data collection and *in situ* analysis of soil and environmental parameters. By incorporating carefully selected yet accurate low-cost soil sensors, an energy harvesting mechanism and scheduling, and a durable and robust mechanical architecture, this IoT sensing system can effectively and sustainably monitor wetlands.

4. Data collection for domain science. In the absence of soil sensors that can directly monitor health parameters such as SOC or bulk density, we used IoT soil sensor field measurements to estimate bulk density, which is one of the critical metrics for determining soil health. Leveraging our domain knowledge and various machine learning (ML) models, we developed an accurate and time-efficient method for estimating bulk density. This approach addresses challenges posed by traditional sampling (which is time-consuming and labor-intensive) and remote sensing methods (often limited by weak signals and low image resolutions). Additionally, we conducted extensive evaluations of multiple ML models to optimize results, cross-referencing our findings with baseline sensors and ground-truth data from the field.

1.5 Dissertation Organization

The rest of this dissertation is organized as follows: Chapter 2 discusses the related work of this research. Chapter 3 presents the energy scheduling approach to manage unstable energy. Chapter 4 presents the

approach to improving DL performance on resource-constrained edge devices. Chapter 5 presents the *in situ* IoT-edge system with low-cost sensors and data analysis from this system. Chapter 6 provides directions for future exploration and conclusions of this dissertation.

Chapter 2

Related Work

This chapter discusses related research work. First, weather forecasting and sensor scheduling techniques for EH sensors are described in section 2.1.1 and section 2.1.2 respectively. Then, we discuss research on DL performance and acceleration on edge devices (Section 2.2) for cloud infrastructure and cloud application management. Lastly, this chapter describes the *in situ* environmental sensing systems (Section 2.3).

2.1 Energy Management for IoT-edge Systems

2.1.1 Weather forecasting for EH sensors.

Exponentially weighted moving average (EWMA) is a commonly used approach for weather prediction in EH sensors due to its lightweight nature [35, 36]. However, in section 3.2.1, HWES (a type of EWMA) was shown to have lower accuracy compared to more advanced LSTM models. While we used HWES, it is only for less significant parameters (*e.g.*, temperature). Guermoui *et al.* [37] utilized SVR to predict solar irradiance and demonstrated higher accuracy than a prediction model based on Multi-layer Perceptron. However, this approach did not consider other important parameters for predicting solar irradiance. Jalali *et al.* [38] used a variation of LSTM to predict GHI with historical data from weather stations, while Capizzi *et al.* [39] employed a different variation of RNN to predict GHI by incorporating other weather parameters. However, both approaches only provided short-term predictions of GHI (*e.g.*, 2 days). In contrast, `DynaES` can produce GHI predictions for a much longer period (at least weeks). Moreover, our prediction goal is not only to predict GHI but also eventually estimate DC power gain by incorporating other important parameters.

Sharma *et al.* [40] developed an approach to predict the energy harvested from natural sources (*e.g.*, solar and wind) by combining forecast data (cloud coverage) from online weather forecasts [41] with data collected from weather stations. While their approach shares a similar goal with the DC power gain estimator in `DynaES`, we did not consider cloud coverage from weather forecasts for power gain prediction. Furthermore, the DC power gain estimator in `DynaES` focuses on capturing the seasonality in various

environmental parameters. Importantly, DynaES can be utilized for EH sensors without network access to obtain online weather forecasting information.

2.1.2 Sensor scheduling in EH sensors.

Limited energy is a major challenge for EH sensors, and several schedulers have been proposed to tackle this problem. Kansal *et al.* [36] designed a greedy algorithm that adjusts sensing intervals according to sensing frequency requirements, but their approach did not consider the power consumption of each sensor. Caruso *et al.* [42] and Loreti *et al.* [43] addressed scheduling using optimization techniques. Yang *et al.* [44] also developed the AsTAR task scheduler that adjusted sensing intervals based on available energy. However, all these approaches did not consider each sensor's priority. In contrast, DynaES schedules sensors with different priorities by giving high-priority sensors more energy and scheduling priority to achieve their sensing goals, unlike most existing approaches that use a fixed sensor priority. Moreover, in our evaluation, we compared DynaES against AsTAR, and DynaES significantly outperformed AsTAR by performing more frequent sensing operations and having longer operation hours.

2.2 Facilitating AI Executions on Resource-constrained Devices

Several studies have conducted quantifying the performance of various edge devices for DL and ML inference tasks [45, 46, 47, 48, 49, 50]. 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 [45] 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. [49] have characterized various edge devices and AI accelerators (e.g., EdgeTPUs) with DL inference tasks. The authors analyzed the impact of DL frameworks and SW stacks and measured energy consumption and temperature when performing DL inference tasks. Moreover, several studies [47, 46, 51] have focused on characterizing the performance of DL inference tasks on different HW architectures and resource models (CPU, GPU, portable AI accelerator). EmBench [46] performed a per-layer analysis of DL inference tasks to identify performance bottlenecks. Libutti et al. [48] conducted performance evaluations of DL inference tasks with portable, USB-based edge accelerators, including USB-Accelerator and Intel Neural Compute Stick [52].

More recently, Liang et al. [50] have conducted an experimental study to evaluate model splitting and compression techniques on edge devices and accelerators when performing co-inference tasks with clouds. Network latency, bandwidth usage, and resource utilization with various configurations were also reported when applying model splitting and compression to cloud-edge co-inference use-cases. Additionally, the authors have evaluated the concurrency model executions for multi-tenancy use cases. However, the concurrency evaluation is narrowly performed with only one model having a single batch size. Moreover,

in addition to evaluating the CME strategy, our work also evaluates and characterizes the DMP strategy for AI multi-tenancy that leverages heterogeneous resources in edge and EdgeTPU.

2.3 In Situ Environmental Sensing

Coastal wetland soil monitoring typically employs remote sensing, utilizing satellites and UAVs. In contrast, smart agriculture leverages soil sensors extensively to assess nutrients, water availability, and microclimatic conditions, such as soil temperature, aiming to enhance crop yield predictions and disease infestation monitoring [16, 53, 54, 17, 18]. On-the-go, field-scale sensing predominantly adopts visible and near-infrared spectroscopy, utilizing either handheld spectrometers or vehicle-mounted configurations for immediate SOC predictions. Nonetheless, handheld sensor deployment in coastal wetlands incurs significant costs, while vehicle-mounted systems introduce undue invasiveness.

Although some literature proposes the utilization of expansive soil sensor networks, like the wireless soil water content system, SoilNet [16, 55, 56], these propositions often fall short in coastal wetlands due to commercial or logistical constraints. Yin *et al.*. [16] advocate for multi-layered sensor networks, spanning measurements of nutrients, pests, pH, moisture, temperature, and pollutants. Their ambit also extended to plant-wearable sensors, integrating into a comprehensive smart agriculture system. However, such intricate systems present substantial energy demands, rendering them logistically challenging in wetland terrains. Furthermore, the financial implications associated with such expansive soil monitoring setups may render them unsuitable for ecological monitoring in coastal wetlands.

Our IoT soil monitoring system is well-balanced, choosing necessary sensors while considering energy efficiency and ease of maintenance. Core soil properties, commonly monitored, encompass soil moisture, temperature, pH, and electrical conductivity. In agricultural frameworks, soil moisture monitors predominantly facilitate irrigation automation [57, 58, 59]. In contrast, within coastal wetlands, they serve as invaluable tidal phase indicators. An intriguing approach by Knadel *et al.*. [60] utilized electrical conductivity and temperature to deduce SOC within a Danish agricultural setting. However, their mobile tractor-mounted sensing strategy, although innovative, is excessively intrusive for coastal wetland terrains.

Moreover, when deploying IoT sensors, energy limitations pose a significant challenge for EH sensors, prompting the development of various scheduling strategies to address this constraint. Kansal *et al.* [36] devised a greedy algorithm that adjusts sensing intervals based on sensing frequency requirements; however, their approach does not account for the power consumption of individual sensors. In contrast, Caruso *et al.* [42] and Loreti *et al.* [43] tackled scheduling through optimization techniques. Our scheduler outperforms baseline methods by enabling more frequent sensing operations and longer operational hours, significantly enhancing its efficiency and effectiveness in EH sensor applications.

Chapter 3

DynaES: Dynamic Energy Scheduling for Energy Harvesting Environmental Sensors

This chapter presents DynaES, a novel energy scheduling method for EH sensors without relying on online weather forecasts. DynaES comprises two components: a DC power gain estimator that predicts future power gain by individually estimating changes in environmental parameters and ensembling them, and a dynamic energy scheduler that distributes energy to sensors based on priority and adjusts sensing intervals and frequency. We evaluate DynaES via simulation-based studies on real-world datasets and compare its performance against state-of-the-art baselines. Evaluation results show that DynaES accurately predicts future energy gain with low estimation errors and enables $1.8\times - 4\times$ more frequent sensing operations with shorter sensing intervals while achieving longer operation hours without complete battery drains.

3.1 Chapter Introduction

With the emergence of low-cost sensors and IoT technologies, various environmental sensors are increasingly developed and deployed to monitor, collect, store, and analyze various environmental factors[61, 62, 63], such as soil [64, 65] and ecological properties [66]. One of the most critical design considerations when deploying such systems is ensuring that the sensors have a stable and sufficient power supply to remain operational. This becomes particularly challenging for deployment places where electric power networks are not available [66].

Energy harvesting (EH) is a promising approach for powering sensors by converting and storing energy from natural sources [28, 29]. In particular, solar energy is a widely used and convenient power source, and various solar panels are well-operational with computing boards like Raspberry Pi and Arduino, as well as IoT sensors. However, the stability of solar energy supply is subject to weather conditions. For example, deployed sensing systems may face multiple rainy and cloudy days, resulting in an energy shortage and discontinuation of sensor operations. Therefore, as shown in Fig. 3.1, EH sensing systems are typically

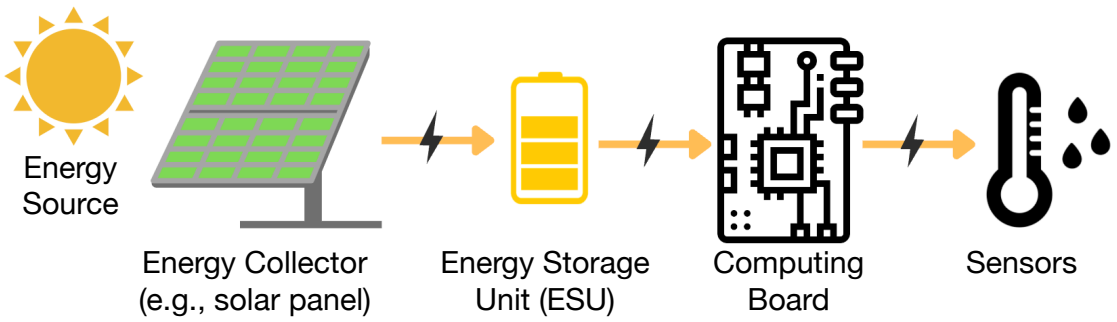


Figure 3.1: Energy harvesting sensing pipeline

equipped with energy storage units (ESUs), such as portable battery packs, that are directly connected to the sensors/computing boards. These units regularly recharge during sunny days to ensure a stable energy supply and continuous operation of the sensors.

The instability of the power supply poses another challenge for solar-powered environmental sensors, as these sensors are often deployed in harsh environments (*e.g.*, wetlands, reservoirs) to collect environmental factors. Unfortunately, these areas often lack reliable network access, making it difficult to obtain timely weather forecasts from well-known weather services [41, 67], thereby hindering the effective management of charged energy levels and power distribution to the sensors. This can result in frequent interruptions and delays in sensor operations to collect target sensing data. Therefore, it is crucial for EH environmental sensors to manage their energy capacity intelligently [68] while enabling seamless sensor operations within a limited power budget [69, 70].

This chapter directly tackles the energy management problem of EH environmental sensors, especially when deployed in harsh environments without network access. We present `DynaES` that accurately estimates the ESU's battery level (DC power gain) and dynamic energy scheduling based on sensor priority. We first develop a DC power gain estimator for `DynaES` to predict the available energy amount during the next operation period. With a thorough analysis of the National Solar Radiation Database (NSRDB) [71] and Photovoltaic Data Acquisition (PVDAQ) Public Datasets [72], we reveal that three parameters – global horizontal irradiance (GHI) [73], temperature, and solar zenith angle (SZA) [74] – are most critical to determine the DC power gain. The power gain estimator forecasts changes in these three parameters for the upcoming sensors' operation period (*e.g.*, one or two weeks) and utilizes a random forest model with the three estimated parameters to determine the final power gain. As this power gain estimator requires *in situ* deployment with the various sensors, we carefully determine the best predictor for each parameter based on the accuracy, overhead, and power consumption in a computing board (Raspberry Pi). Specifically, we use a combination of lightweight deep learning and statistical time-series models for estimating these parameters.

Moreover, the scheduling component in `DynaES` performs dynamic power distribution based on each sensor's priority (significance). The sensing priority is determined based on domain-specific characteristics

(*e.g.*, ideal sensing frequency for high data quality) and real-world metrics (*e.g.*, power drain). The scheduler then dynamically adjusts the intervals and frequency of sensing operations in order to ensure high energy efficiency and preserve sensing data quality.

I created a trace-based simulator to evaluate the effectiveness of DynaES, which utilizes input data created from two publicly available solar radiation datasets to estimate the DC power gain and simulate dynamic power scheduling for different EH sensors. To ensure a realistic simulation study, we collected data on the real power consumption of the DC power gain estimator and environmental sensors, including pH, electrical conductivity (EC), oxidation-reduction potential (ORP), and temperature sensors. I used this data as the simulation parameters for evaluating DynaES.

My evaluation of DynaES focused on measuring the accuracy of the estimated DC power gain, monitoring changes in the charged battery level, and analyzing the frequency and intervals of sensing operations to determine whether DynaES can ensure sustainable sensor operations and data quality. According to our evaluation results, DynaES was able to accurately estimate the DC power gain with a RMSE of less than 100 within a week. I further compared the scheduling performance of DynaES with state-of-the-art baselines. The results revealed that DynaES enabled 1.8× to 4× more sensing operations, with shorter sensing intervals, without experiencing any outage caused by a complete battery drain.

3.1.1 Chapter Organization

The rest of this chapter is organized as follows: Section 3.2 describes the detailed design of DynaES. Section 3.3 discusses our evaluations of DynaES, focused on its power gain prediction accuracy and energy scheduling efficiency. Section 3.4 provides a discussion regarding this study. Section 3.5 summarizes this chapter.

3.2 The Design of DynaES

This section describes the detailed design of DynaES. Fig. 3.2 shows the overall design of DynaES that consists of two main components: ESU's DC power gain estimator (section 3.2.1) and dynamic energy scheduler (section 3.2.2) for various sensors. As stated in the introduction section, EH environmental sensors often have constraints, such as unreliable network access, which prevents them from obtaining timely weather information for future operational intervals. Consequently, DynaES should be capable of knowing the future power gain from solar energy. To achieve this, the **DC power gain estimator** of DynaES predicts future power gain by utilizing various parameters obtained by solar panels and equipped sensors in the deployed system.

After obtaining the predicted power gain, the **dynamic energy scheduler** in DynaES performs energy scheduling and distribution for various sensors based on their priority. The priority of each sensor is determined by its sensing objectives, with input from domain experts. The goal of the scheduler is to enable frequent operations of the more important sensors, thereby ensuring that the EH environmental

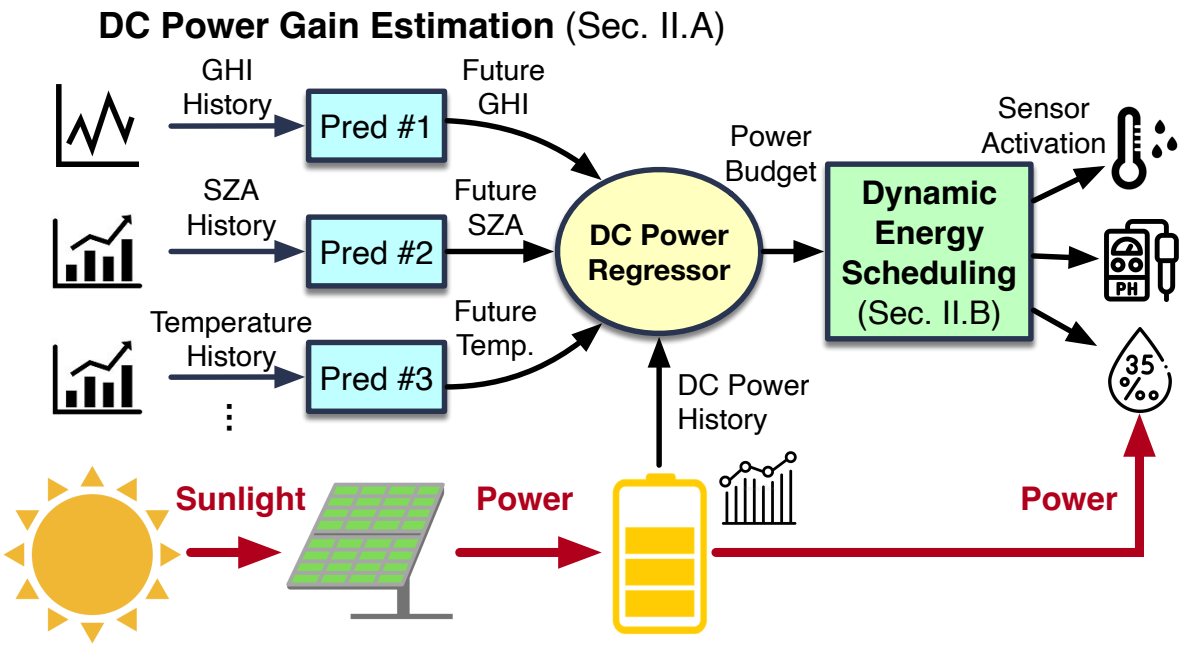


Figure 3.2: The overall design of DynaES

sensors achieve their sensing goals efficiently while maintaining stable operations by avoiding a complete battery drain.

3.2.1 DC Power Gain Estimation

To design the DC power estimator, we start by investigating the relationship between solar/environmental parameters and DC power gain through solar panels. Unfortunately, there is no single dataset/DB containing solar energy-related factors and DC power gain. However, we found two datasets separately containing solar and environmental factors (NSRDB [71]) and DC power gain via photovoltaic solar energy (PVDAQ [72]).

Data preparation

NSRDB, maintained by the National Renewable Energy Laboratory (NREL), contains time-series data of meteorological, environmental, and solar irradiance from various locations across the U.S. The DB provides access to 19 critical environmental and solar-energy-related parameters, including temperature, wind speed/direction, SZA, GHI, and other essential information. While NSRDB has various time-series data regarding environmental parameters and solar irradiance, it lacks information connecting DC power gain leveraging solar irradiance and other factors. Therefore, we decided to merge NSRDB with PVDAQ DB.

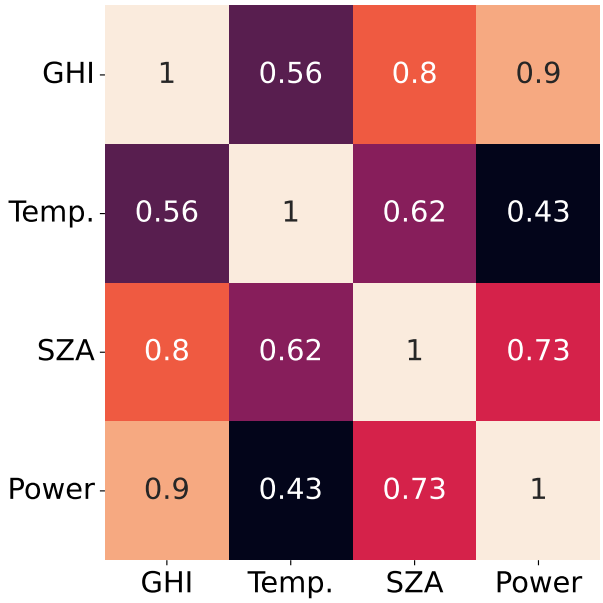


Figure 3.3: Correlation coefficient of environmental/solar-irradiation parameters with DC power gain

PVDAQ is also collected and maintained by NREL. The datasets contain large-scale time-series data of PV (Photovoltaic) system metadata (*e.g.*, PV system models and dimensions of solar panels) and performance data (*e.g.*, DC power gain) from various experimental and public PV sites across the U.S. To merge these two separate datasets, we first found data chunks collected in the same locations and same measurement periods. One challenge in the data integration was that PVDAQ often had missing data points (data in specific months or days), so we had to focus on generating the longest continuous data from two DBs. Specifically, we could create integrated datasets from 2015 to 2018 containing all parameters in both DBs.

Estimating future changes in GHI and environmental parameters

With the integrated datasets, we performed correlation analysis to find the most significant environmental parameters to determine the DC power gain. Fig. 3.3 shows the correlation coefficient of various parameters (in NSRDB) with determining DC power gain (in PVDAQ), and we observed that GHI, temperature, and SZA are strongly or moderately correlated with DC power gain.

GHI refers to the amount of solar irradiance received by a solar panel on a horizontal surface [73]. GHI is directly linked to the amount of energy the solar panel can generate. Additionally, the efficiency of a solar panel in energy generation is directly impacted by ambient temperature. SZA is the angle between the sun and the zenith and can be an important factor in determining the amount of sunlight that a solar panel receives at different times of day and year [74]. When SZA is low, the sun is closer to the zenith,

and the panel receives more direct sunlight, resulting in increased power generation. Conversely, when the SZA is high, the sun is farther from the zenith, and the panel receives less direct sunlight, resulting in reduced power generation. Therefore, these three parameters can be useful in predicting future DC gain. Please note that while we analyzed the coefficient of all 19 parameters, Fig. 3.3 only reports the three most important parameters.

To estimate the future DC power gain, it is critical to know the future changes of GHI, SZA, and temperature. Moreover, the estimation of future parameter changes needs to be performed on a computing board deployed together with EH sensors. Therefore, we should carefully decide to use proper predictors for these parameters based on their performance and power consumption (as the predictors should not consume a significant amount of power in the sensing system). For the predictor selection, we tested three time-series predictors from deep-learning and statistical learning approaches: long short-term memory (LSTM) [75], NeuralProphet (NP) [76], and Holt-Winters exponential smoothing (HWES) [77] methods.

LSTM is a type of recurrent neural network (RNN) designed for sequence or time-series predictions. In each inference task, LSTM generates both a prediction and hidden states. Specifically, the hidden states are passed back into the model for the next prediction during each inference. These hidden states store valuable information about the sequence of data, such as long-term and short-term trends in time series, which can assist in making accurate inferences [75]. LSTM can be a suitable choice for predicting future parameter changes since the seasonal period of each parameter could extend up to 3 months (the average length of a meteorological season). In such cases, the long-term memory of LSTM can be useful. Additionally, even within a specific season, environmental parameters can have fluctuations. Therefore, the short-term memory of LSTM can be beneficial in such cases.

NP is also designed for time-series predictions with supporting automatic detection of trends and seasonality in the given time-series dataset. NP is known to be an efficient approach in various time-series prediction problems [76]. And finally, HWES is a lightweight and widely used time series model focusing on capturing trends and seasonality in its forecasting. We chose HWES as it is lightweight (potentially consuming less power), and the parameters (*e.g.*, GHI and temperature) we want to predict can have strong seasonality.

	Accuracy (RMSE)	Overhead (Time in Sec.)		Power Consumption (mAh)	
		Inference	Training	Inference	Training
HWES	150.8	**0.04**	–	**0.01**	–
NP	179.0	1.56	91.68	0.25	18.62
LSTM	**117.9**	6.45	138.41	0.99	25.52

Table 3.1: Accuracy, overhead, and power consumption measurement results of three prediction models on Rasberry Pi. The bold values are the best results we obtain in the measurement.

To determine the proper predictor, we examined three candidates with GHI prediction, as GHI is the most important contributor to determining DC power gain. This test was performed on Rasberry Pi 4B, a

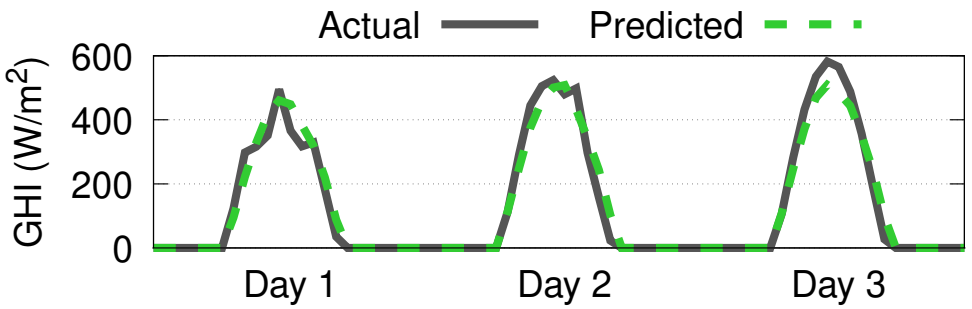
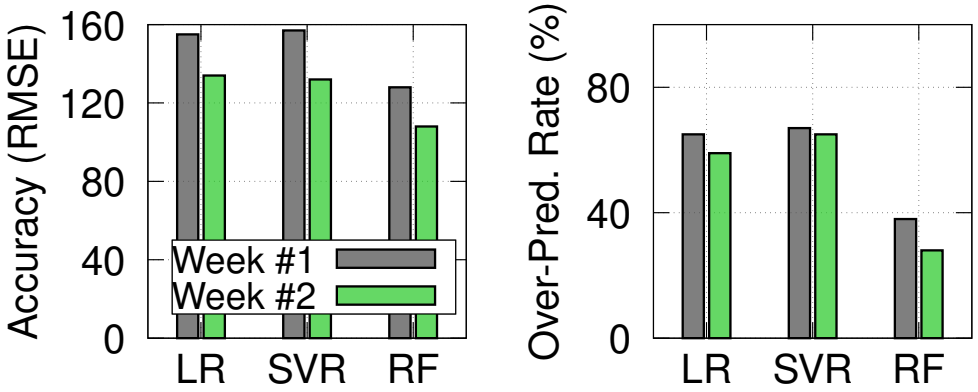


Figure 3.4: 3-day GHI prediction results by LSTM (solid line: actual GHI values, dotted line: predicted GHI values)



(a) Overall Accuracy (RMSE)

(b) Over-prediction Rate (%)

Figure 3.5: Accuracy evaluation results of three DC regressors

widely used general-purpose computing board. We measured the prediction accuracy (RMSE), overhead in the training model and performing inference tasks, and actual power consumption. Specifically, we used `INA-219`[1], a voltage, current, and power measurement chip, for measuring the power consumption of the models on the computing board.

The measurement results are reported in Table 3.2.1. LSTM produced the most accurate prediction results but required higher computation time in training and inference than the other two models. LSTM's power consumption measurements on Raspberry PI were 0.99mAh (inference) and 25.52mAh (training). On the other hand, HWES showed a significantly lower overhead than the two other models, and its power consumption was also negligible. But its prediction accuracy was worse than LSTM (but better than NP).

[1] `https://www.ti.com/product/INA219`

As both LSTM and HWES models have strengths and weaknesses, we decided to use different predictors for estimating different parameters based on their significance. Specifically, LSTM is used for GHI predictions due to its strong correlation coefficient (0.9) with DC power gain. `DynaES` can leverage the accurate prediction results powered by LSTM. For example, Fig. 3.4 shows 3-day GHI prediction results by LSTM, and the figure shows that LSTM could accurately predict the GHI changes. Moreover, LSTM's power consumptions were only 0.99mAh (inference) and 25.52mAh (training), which can be easily accommodated by the typical ESU's capacity of 20,000 – 30,000mAh, given that training and inference will be performed periodically (*e.g.*, once a week). For the temperature and SZA, we decided to use HWES as it has reasonable prediction accuracy and negligible power consumption.

DC power gain regression. The next step is to design a DC power gain regressor that takes three predicted parameters from the previous step as inputs and estimates the future gains of DC power in ESU. To achieve more accurate prediction, this DC power regressor also leverages the history of DC power gains. To incorporate the predicted parameters and the DC power gain history, we considered various regressors, including linear regression (LR) [77], support vector regression (SVR) [77], and random forest (RF) [78] models.

LR is a commonly used approach for various regression problems due to its intuitive nature. We also considered the SVR because it can model both linear and non-linear relationships between the parameters and DC power via the kernel. Finally, we tested RF, which not only captures linear and non-linear relationships (like SVR) but is also known for its robustness to outliers and over-fitting. To compare the accuracy of these three regressors, we randomly selected two weeks of data from our dataset and assessed the performance of each regressor on this data.

Fig. 3.5 shows the accuracy evaluation results of three DC power regressors. Specifically, Fig. 3.5a reports the RMSE results of all the predictions. Overall, RF showed significantly higher accuracy than the other two regressors. For example, RF's RMSE results were 128 (week #1) and 108 (week #2), but the other two regressors had around a RMSE of 155 (week #1) and 133 (week #2).

Additionally, it is also important for the DC regressor in `DynaES` to have minimal over-prediction cases. Over-prediction occurs when the predicted DC power gain is higher than the actual DC power gain, which can lead to several scheduling issues. *i.e.*, overly allocating energy to sensors, resulting in more frequent sensing operations than future energy gain. Fig. 3.5b shows the over-prediction rates of the three regressors. Consistent with previous results, the RF-based regressor showed the lowest over-prediction rates, with 38% (week #1) and 28% (week #2), respectively. In contrast, LR had over-prediction rates of 65% and 59%, and SVR showed over-prediction rates of 67% and 65% for weeks #1 and #2, respectively.

Our further analysis revealed that the better accuracy of RF is mainly due to its robustness to outliers. For example, DC power gain prediction relies on various parameters with seasonality but can also be impacted by several intra-seasonal variations (*e.g.*, temperature changes within a specific season). RF is an ensemble model that trains each constituent tree on a random subset of the features, indicating that an outlier variable would not affect the prediction outcome of all the trees. Additionally, because the final prediction outcome is an average of the prediction outcomes of each tree, the outlier variable would have a reduced impact on the final prediction outcome in RF compared to other models.

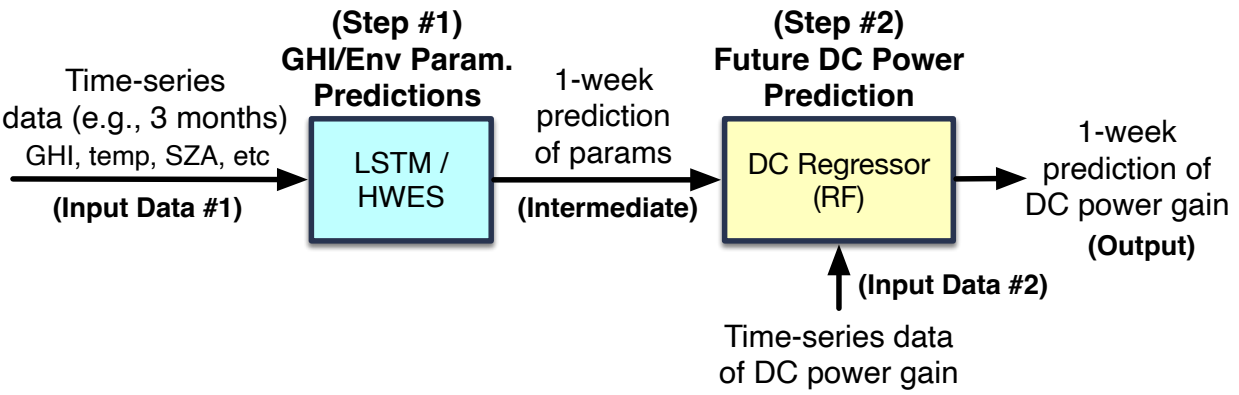


Figure 3.6: Overall prediction procedure to estimate DC power gain

With accuracy and over-prediction rate evaluations, we decided to use RF for our DC power regressor. The overall prediction procedure is illustrated in Fig. 3.6. In summary, the power gain estimation process in DynaES consists of two steps. The first step (step #1 in Fig. 3.6) takes past time-series data of GHI, temperature, and SZA and predicts future changes in these parameters. In this step, a more accurate LSTM is used to predict future GHI, while power-efficient HWES models are used to predict other parameters. The DC regressor (step #2 in Fig. 3.6) then utilizes RF to predict future (*e.g.*, 1-week) power gain by taking predicted parameters (step #1) along with the history of DC power gains. The output from the DC regression will be predicted power gain for the next week (hourly data for seven days, in total 24×7 predictions), which is the energy budget for the dynamic energy scheduler in the following subsection to facilitate further operations.

3.2.2 Dynamic Energy Scheduling

Given the prediction for future DC power gain (section 3.2.1), the dynamic energy scheduler in DynaES creates scheduling plans for each sensor. Our scheduling algorithm is described in Algorithm 1. The scheduler takes the following inputs: DC power gain prediction (E_{gain}), sensor profile (SP), battery (ESU) profile (B), and scheduling time window (T). Specifically, T refers to the duration for which we intend to schedule the power distribution to sensors in advance. And the scheduler's output is the scheduling plan for sensors during T.

The scheduler initiates the power distribution process by computing the total (E_{tot}) and accumulated energy (E_{accum}) for T (lines 1–5). T is a user-defined parameter (*e.g.*. one or two weeks). Following this, the scheduler evaluates whether E_{tot} is sufficient to meet the minimum energy requirement ($B[min\_energy\_required]$) for non-sensing activities, such as power consumed by computing boards or other equipment (line 6). If the energy is inadequate, then the EH sensor cannot carry out any sensing operations during T (line 7).

Algorithm 1: DynaES Energy Scheduling Algorithm

Input: DC power (energy) gain prediction: E_{gain}, sensor profile: SP, battery profile: B, scheduling time window: T

Output: sensor schedule plan: $sched$

1 initialize hourly accumulated energy $E_{accum}[0] = E_{gain}[0] + B[current\_level]$
2 calculate total energy amount over T $E_{tot} = B[currrent\_level] + sum(E)$
3 **for** $t \in [1, T]$ **do**
4 $\quad E_{accum}[t] = E_{accum}[t-1] + E_{gain}[t]$
5 **end**
6 **if** $E_{tot} \leq B[min\_energy\_required]$ **then**
7 $\quad$ sleep(T)
8 **end**
9 calculate energy priority ratio $E_{pr} = E_{tot} \div \sum_{s \in SP} s[priority]$
10 calculate energy allocation to each sensor $E^s_{alloc} = E_{pr} \times s[priority]$
11 calculate number of sensing operations for each sensor $S^s_{ops} = E^s_{alloc} \div s[consumption]$
12 calculate the sensing period for each sensor $S^s_{period} = T \div S^s_{ops}$
13 choose the shortest sensing period $T\_min = min(S^s_{period})$
14 choose sensors with the shortest sensing period
$S^s_{min\_period} = \{s : S^s_{period} == T\_min \ \ \forall \ s \in SP\}$
15 calculate the energy required for the schedule
$E_{req} = B[base\_consumption] + \sum_{s \in SP} S^s_{min\_period} \times s[consumption]$
16 **for** $t \in [T\_min, T]$ **do**
17 $\quad$ search for the sufficient energy time
18 $\quad$ **if** $E_{accum}[t] > E_{req}$ **then**
19 $\quad\quad T_{end} = t$
20 $\quad$ **end**
21 **end**
22 return $[S^s_{min\_period}, T_{end}]$

The scheduler then assigns energy distribution priority (E_{pr}) to each sensor ($s, \forall s \in SP$) based on each sensor's priority level ($s[priority]$) in line 9 and allocates energy for each sensor accordingly (E^s_{alloc}) as described in line 10. Subsequently, the scheduler calculates the frequency (line 11) and interval (line 12) for each sensor during T. Also, the shortest sensing period is calculated as the sensing time (line 12). For the sensors that have the shortest sensing period are selected to be scheduled (line 14). Next, the scheduler calculates the required energy (E_{req}) to execute all sensing operations (line 15) and searches the time that has enough energy in E_{accum} to determine the termination time (T_{end}) of sensing operations (lines 16-21). Finally, the scheduler generates the schedule plan, which includes the calculated sensing interval T_{end} and all sensors ($S^s_{min\_period}$) for the sensing operations.

3.3 Evaluation of DynaES

Our evaluation aims to assess the performance of the DC power estimator and energy scheduling in DynaES. The accuracy of the DC power estimator is critical for DynaES as it predicts future energy gains for scheduling operations. We measured the accuracy (RMSE) and over-prediction rate of the DC power estimator, as well as the impact of various parameters on producing accurate prediction results.

Additionally, the performance evaluations of the dynamic energy scheduler were based on simulations with realistic scenarios of EH environmental sensors. Specifically, we measured changes in energy charging level in ESU, sensing operation frequency, and sensing intervals in DynaES with various sensors. Furthermore, we compared the performance of our dynamic energy scheduling against three baselines: static, adaptive energy scheduler, and AsTAR.

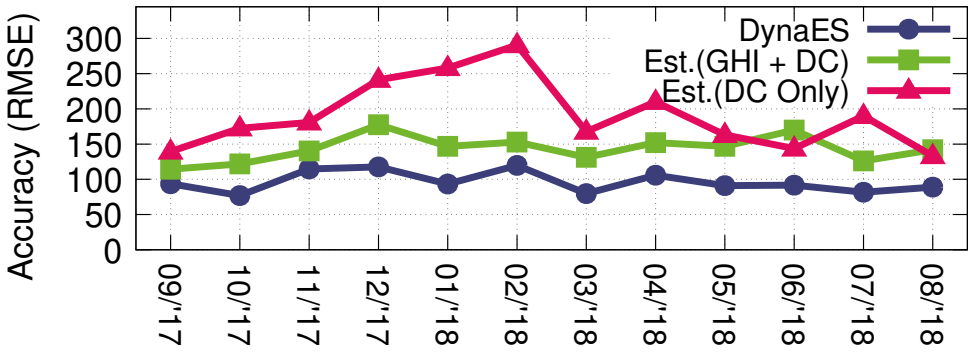
Evaluation dataset

As discussed in section 3.2.1, we merged two publicly available datasets collected by NREL to create new datasets. For the performance evaluation, we prepared datasets for a two-year period (2017-2018). The datasets include GHI, temperature, SZA parameters, and DC power gain in the Denver area, CO. It is worth noting that the dataset used to design DynaES was not used for its performance evaluation to ensure a fair evaluation.

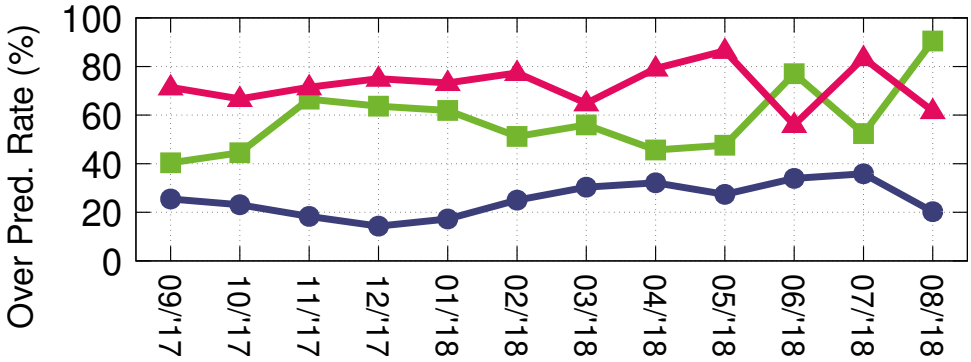
Simulator design and implementation.

We developed a trace-based simulator modeled on the behavior of environmental sensors in EH systems. The simulator has four main components: a timer, an energy generator, a battery, and a scheduler. The timer component is responsible for controlling the simulation time. The energy generator takes a trace file containing time-series data of GHI and other environmental parameters as input and generates corresponding values for GHI, temperature, and SZA at predefined simulation times. The battery component mimics the behavior of ESUs in EH sensors. This component simulates power charging and discharging functions and generates a trace of the DC power history, which is used by the scheduler component. The battery leak function is also implemented to simulate the real-world behavior of batteries. The scheduler is the core unit of the simulator and performs critical operations. This component predicts DC power gain, schedules energy delivery to sensors, traces all sensor operations and coordinates other system functions. This component ensures that the simulator accurately mimics the behavior of EH systems in real-world scenarios. This simulator is written in Python. Pandas[2] [79] and Numpy[3] [80] libraries are used to process the time-series data. Scikit-learn[4] [81] and TensorFlow[5] [82] are used to implement RF and LSTM models.

[2] https://pandas.pydata.org
[3] https://numpy.org
[4] https://scikit-learn.org
[5] https://www.tensorflow.org



(a) Overall Accuracy (RMSE)



(b) Over-prediction Rate (%)

Figure 3.7: Accuracy evaluation results of three DC power gain estimators. (a) shows RMSE results for 12 weeks (the first week of every month) from September 2017 to August 2018. (b) shows the over-prediction ratio of three estimators. Please note that Est.(GHI + DC) in the figure indicates the DC power estimator using GHI and DC power history, and Est.(DC Only) is the estimator only using DC power history.

3.3.1 Accuracy of DC Power Gain Estimation

Dataset for power gain estimation

We used 12 weeks of data spanning from September 2017 to August 2018. Specifically, we tested the estimation accuracy of the first week of every month over a one-year period, resulting in a total of 12 prediction tests. This allowed us to determine if the DC power gain estimator consistently yields high prediction accuracy.

Power gain estimators

While we previously revealed that GHI is the most important parameter for predicting future DC power gain (in section 3.2.1), we also wanted to evaluate the contribution of other parameters, such as temperature, and SZA. To this end, we created three different versions of the DC power gain estimator, which are `DynaES`' DC power estimator, a DC power estimator utilizing GHI and DC power history, and a power estimator using only DC power history.

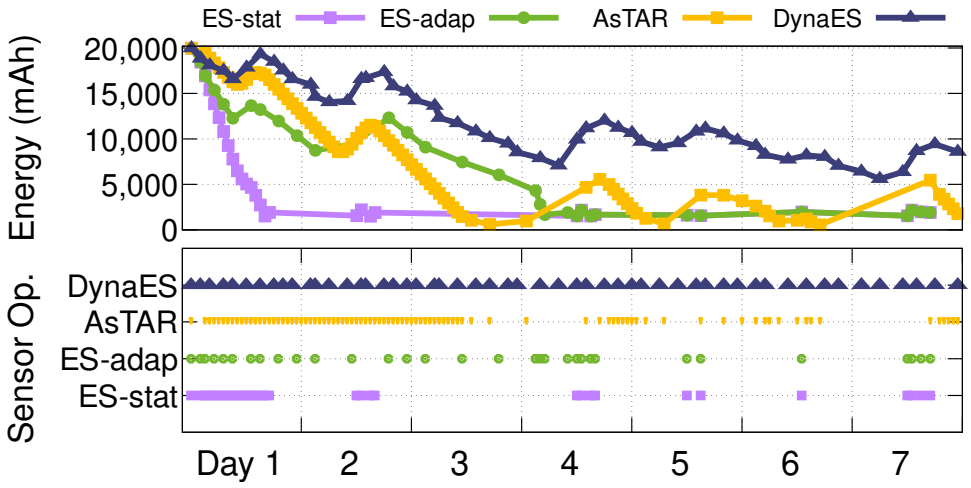
Evaluation results

Fig. 3.7 shows the accuracy results of the three power estimators. Specifically, as shown in Fig. 3.7a, `DynaES`' power estimator outperformed the other two models and consistently produced prediction results with low prediction errors. On average, `DynaES`' estimator had a RMSE of 96. The second estimator (`Est.(GHI + DC)` in the figure), which utilizes GHI and DC power history, had a RMSE of 143, which is 50% higher than `DynaES`. As this estimator does not take into account the other three environmental parameters, the results indicate that both temperature and SZA parameters significantly contribute to reducing prediction errors. The third estimator (`Est.(DC Only)` in the figure), which only utilizes DC power gain history, had the highest prediction errors with a RMSE of 190, significantly higher than `DynaES`. Moreover, the accuracy of the third estimator was much more variable than `DynaES` and the second estimator. This finding highlights the importance of GHI and other parameters in improving the prediction accuracy of future DC power gain.

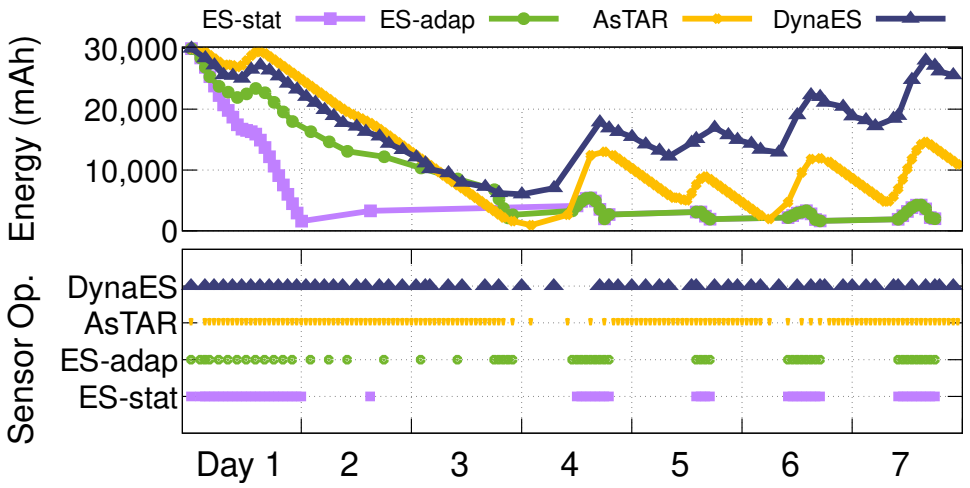
Fig. 3.7b reports the over-prediction rates of all three power estimators. Over the one-year evaluation period, `DynaES` had an over-prediction rate of 14% – 35%, with an average over-prediction rate of 25%. `DynaES` consistently generated lower over-prediction rates (32% – 47% lower) than the other two estimators, which can lead to more stable management in energy scheduling to target sensors.

3.3.2 Energy Scheduling Evaluation

This evaluation is to test the effectiveness of the energy scheduling algorithm in `DynaES`. Our evaluation involves monitoring changes in the charged energy levels of the ESU, as well as adjustments in the sensing frequency and intervals. Specifically, monitoring charged energy-level changes in the ESU is to assess



(a) 4th Week in December (2017)



(b) 2nd Week in November (2017)

Figure 3.8: Evaluation results of energy scheduling performed by `DynaES`, `ES-stat`, `ES-adap`, and `AsTAR`. (a) represents the evaluation results with a 20,000mAh of battery capacity in the 4th week of December 2017, and (b) shows the results with a 30,000mAh of battery capacity in the 2nd week of November 2017. The upper graphs show the battery level changes (charged by solar panel and the battery consumption by sensing operations) during evaluation weeks. The lower graphs represent the sensing operation frequencies.

whether the EH sensors can operate efficiently without a complete power drain. Furthermore, changes in sensing frequency and intervals are also essential metrics, as the EH sensors should not compromise their performance simply to conserve energy for longer operation hours. To ensure efficient functionality, the energy scheduling algorithm in `DynaES` should be capable of maintaining the frequency and interval for each sensor based on its priority while considering the dynamics of the charged energy level.

Sensor and ESU models in simulation

We consider the deployment of an EH system with four sensors, including temperature, pH, EC, and ORP sensors. We assume that temperature and pH sensors have higher sensing priority (more frequent sensing operation, *e.g.*, two operations per hour) and EC and ORP sensors have lower priority (*e.g.*, one operation per hour). For realistic simulation, we profiled the power consumption of all the sensors using the same approach used in section 3.2.1 and used the profiled data as simulation parameters. For the ESU models, we modeled ESU components in our simulator based on battery packs with 20,000 and 30,000mAh, commonly available on the market.

Dataset for scheduler evaluation

We randomly picked five individual weeks in our dataset. Three weeks are selected from 2017, and two weeks are chosen from the 2018 dataset.

Baselines

Static energy scheduler (`ES-stat`), adaptive energy scheduler (`ES-adap`), and `AsTAR` are used as baselines. `ES-stat` performs sensing operations based on a pre-determined sensing interval. As `ES-stat` does not consider the changes in the ESU's energy level or other environmental dynamics, `ES-stat` is to show the baseline performance of the energy scheduler in EH sensors.

`ES-adap` dynamically adjusts sensor operation frequencies based on the current energy level in the ESU. `ES-adap` starts scheduling by checking if the current energy level exceeds a threshold set to decrease the chance of complete battery drain. If the current energy level is higher than the threshold, `ES-adap` enables sensing operations. After each sensor operation, `ES-adap` updates the energy level by calculating the energy gain and drain from the sensors. It then determines the next sensing interval by calculating:

$$\alpha \times \frac{battery\_level - threshold}{threshold}, \tag{3.1}$$

where α is a scaling factor used to regulate the energy-capacity scaling. In this way, `ES-adap` adjusts the sensing interval based on the current energy level. Thus, when the battery level is higher, `ES-adap` schedules more frequent sensing operations, and when the battery level is lower, it schedules fewer sensing operations.

`AsTAR` [44] is an energy scheduler focused on performing energy-aware adaptation for sensor tasks. It aims to balance the trade-off between higher temporal resolution of sensed data and increased energy

consumption. AsTAR is designed to operate without benchmarking or environmental modeling, instead observing and reacting to the system state at runtime. This energy scheduler is also designed to address platform heterogeneity and unpredictable energy-harvesting environments.

Evaluation results

We performed energy scheduling in the five weeks in 2017 and 2018 and measured energy level changes in the ESU and sensing frequency (interval) in the five weeks. Fig. 3.8 reports the energy scheduling results in two weeks. Please note that we only show two weeks' results due to the page limit, and the other three weeks had similar results. As shown in the upper graphs of Fig. 3.8 (the changes in battery level), DynaES demonstrated its ability to effectively manage battery levels and enable more frequent sensing operations without fully depleting the battery. Although there were fluctuations in the charged battery level, these fluctuations were primarily due to weather dynamics. Conversely, the other three baselines could not support sustainable operations of EH sensors. For example, ES-stat experienced a complete battery drain on day 1 of both weeks and thereafter, ES-stat only conducted very limited sensing operations. ES-adap performed better than ES-stat. However, ES-adap and AsTAR also encountered a complete battery drain on days 3 and 4 of Fig. 3.8a and day 3 of Fig. 3.8b, leading to limited support for future operations due to the shortage of charged energy.

The lower graphs of Fig. 3.8 further illustrate the sensing operation frequencies performed by DynaES and three baselines. The results show that DynaES enabled $1.8\times - 4\times$ more frequent sensing operations, indicating that DynaES does not sacrifice the frequency of its sensing operations to conserve more energy. By conducting more sensing operations, the EH sensors equipped with DynaES can collect more data from its sensors, thereby facilitating the achievement of the sensors' objectives. Moreover, we observed that DynaES could dynamically adapt the sensing interval based on the charged battery level. *i.e.*, day 4 of Fig. 3.8b's lower graph. DynaES extended the sensing interval to prevent the complete battery runout because the less energy gain on day 4 was predicted from the DC power gain estimator.

Fig. 3.9 shows the CDF of the sensing interval for the four schedulers. The results show that the sensing intervals for DynaES were consistently smaller compared to the three baselines. Conversely, the three baselines exhibited long-tail distributions of sensing intervals, indicating their inability to support consistent sensing operations. By having shorter sensing intervals (without long-tail distribution) in Fig. 3.9 and higher sensing frequency in Fig. 3.8, DynaES can support more frequent and consistent sensing operations, leading to improving the sensing quality of EH sensors.

We also implemented and deployed DynaES in Athens, GA. RPi4 is used as the controller. The power management chip is PiJuice HAT [83]. It can wake up RPi4 with a built-in timer function. The baseline is ES-adap. AsTAR is not used due to that PiJuice can only provide the integer part of the battery level. The decimal part is truncated, which leads to an imprecise reading of the battery level. This affected AsTAR making a proper sensing schedule.

Fig. 3.10 illustrates the battery level change for the deployed schedulers. It is evident from the depicted data that DynaES exhibits a notably higher battery level compared to ES-adap. This observation

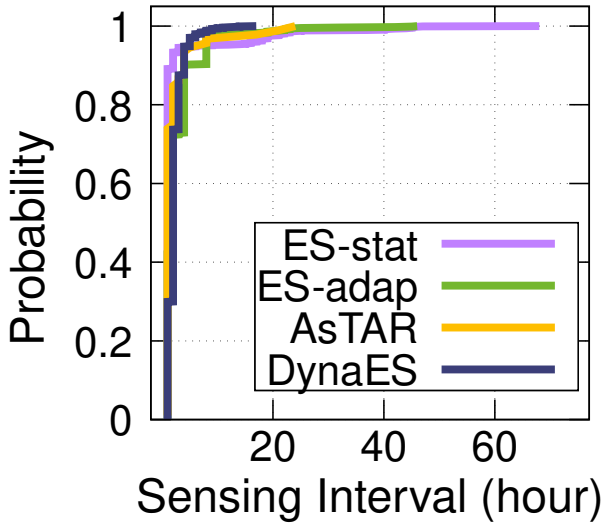


Figure 3.9: CDF of sensing intervals in four schedulers

underscores the practical efficacy of DynaES in energy management strategies. Notably, these findings corroborate the simulation outcomes, thus validating the efficacy and utility of the simulator.

In this evaluation, we showed that DynaES outperformed the three baselines. The better performance of DynaES is primarily due to its accurate prediction of future power gain using various parameters and its scheduling algorithm, which can dynamically adjust scheduling intervals to manage the battery level, thereby supporting sustainable overall runtime of EH sensors with increased frequency of sensing operations.

3.4 Discussion

This study presents DynaES, an energy scheduler for EH sensors without reliable network access to online weather forecasting APIs. DynaES uses a DC power gain estimator to predict future solar panel power gain for ensuring sustainable sensor operations. However, to utilize DynaES, EH sensors should be capable of collecting GHI, temperature, and SZA parameters. This section discusses two key questions for adopting DynaES in real-world deployments: 1) how to obtain GHI and other parameters for EH sensors, and 2) DynaES's performance when GHI and other parameters are not available.

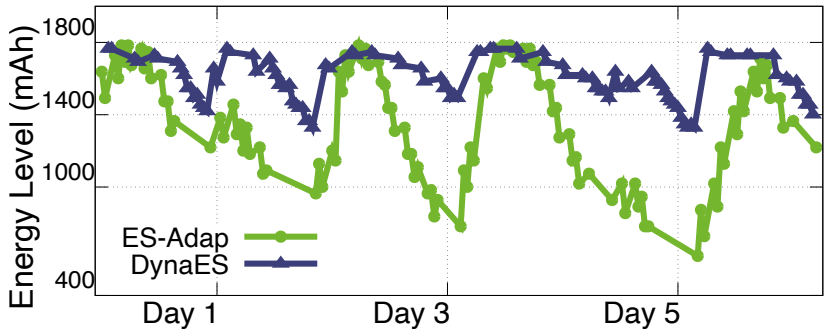


Figure 3.10: Changes in energy level for deployed schedulers.

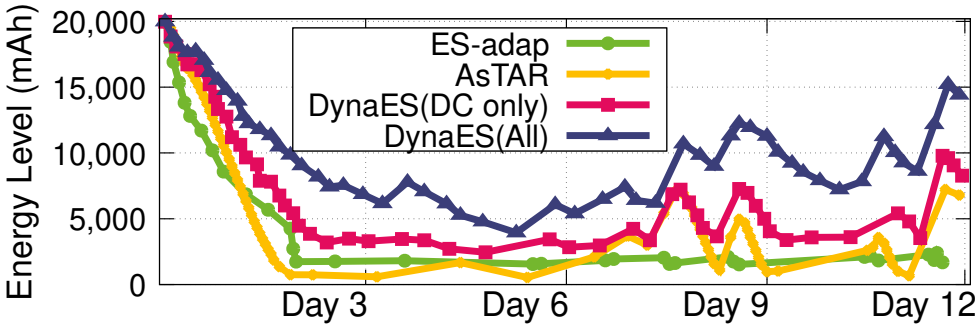


Figure 3.11: Changes in energy level by four schedulers. `DynaES` (all) refers to `DynaES` with DC power gain prediction using GHI, SZA, temperature, and DC power history. `DynaES` (DC only) only uses DC power gain estimation using DC power history. All the dots on the lines indicate sensing operations.

3.4.1 Question 1: How to collect GHI, SZA, and temperature parameters in in situ EH sensors.

A pyranometer can collect GHI parameters [73]. A typical pyranometer setup includes a light sensor that filters out non-sunlight waves. This light is then converted into thermal energy, which is subsequently transformed into electricity using a thermoelectric device, such as a thermopile.

In a practical deployment of EH sensors, a pyranometer can be installed near or alongside the EH sensors to collect GHI data. This information can then be utilized in our DC power gain estimator in `DynaES`. Furthermore, the pyranometer can help to enhance the prediction accuracy of our power gain estimator by comparing the predicted GHI parameters against the pyranometer's actual measurements.

In addition, SZA parameters [74] can be calculated using equations that utilize the current time and location coordinates (longitude and latitude). Temperature parameters can be obtained using low-cost sensors widely available on the market.

3.4.2 Question 2: The performance DynaES without GHI, SZA, and temperature parameters.

Given the critical role of GHI along with SZA and temperature parameters in estimating future DC power gain, we seek to determine whether DynaES can still effectively schedule energy to sensors even without GHI, SZA, and temperature parameters, as these sensors (*e.g.*, pyranometer) may not always be available in real-world deployment scenarios. In this discussion, we evaluate the performance of DynaES powered solely by DC power gain prediction with historical DC power information, which can be easily obtained by APIs supported by commercial battery packs.

Fig. 3.11 illustrates the changes in energy levels of four schedulers; DynaES(All), DynaES(DC only), AsTAR (baseline), and ES-adap (baseline). DynaES(All) employs all parameters for DC power gain prediction, while DynaES(DC only) only uses DC power history for energy-gain prediction. The results show that DynaES(DC only) was not able to maintain the same energy level as DynaES(All) due to lower accuracy of future energy gain prediction. However, it showed higher efficiency than the baseline by managing the battery level and performing more sensing operations (dots on lines). Notably, DynaES(DC only) was observed to perform a similar frequency of sensing operations with DynaES(All).

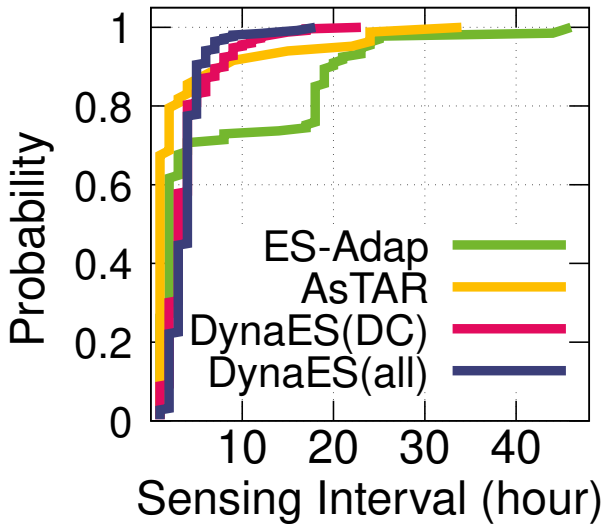


Figure 3.12: CDF of sensing intervals of DynaES(All), DynaES(DC only), and ES-adap (baseline)

Additionally, we measured the sensing intervals performed by the schedulers. Fig. 3.12 reports the CDF of the sensing interval. The results show that DynaES(DC only) had similar sensing intervals to DynaES(All), with only the lower 20% of sensing intervals being slightly longer than those of DynaES(All). Specifically, we did not observe a long-tail distribution of sensing intervals in DynaES(DC only), suggesting that it can support sufficiently frequent sensing operations despite relatively inaccurate DC power gain estimations.

3.5 Chapter Summary

This chapter presents DynaES, a new energy scheduler for solar-powered EH environmental sensors, which are deployed in environments without access to weather information networks. We designed DynaES with two components: a DC power gain estimator and a dynamic energy scheduler. For the DC power gain estimator, we first merged two datasets and then used deep learning and statistical time-series approaches to predict changes in GHI, SZA, and temperature parameters to estimate future DC power gain. DynaES schedules energy distributions to each sensor based on priority, dynamically adjusts sensing intervals/frequency and ensures energy efficiency and data quality. Evaluation results showed that DynaES accurately predicts future energy gain and effectively schedules energy to support more frequent operations and longer battery life. In the near future, we plan to deploy solar-powered EH sensors with DynaES and evaluate the efficacy and performance of DynaES in a real-world deployment.

Chapter 4

Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-tenancy

This chapter investigates system techniques, such as batched inferencing, AI multi-tenancy, and the cluster of AI accelerators, which can significantly enhance the overall inference throughput on edge devices with DL models for image classification tasks. In particular, AI multi-tenancy enables collective utilization of edge devices' system resources (CPU, GPU) and AI accelerators (*e.g.*, Edge Tensor Processing Units; EdgeTPUs). Furthermore, we present a detailed analysis of hardware and software factors that change the DL inference throughput on edge devices and EdgeTPUs, thereby shedding light on areas that could be further improved to achieve high-performance DL inference at the edge.

4.1 Introduction

High network latency and privacy concerns have hindered the adoption of cloud computing for Internet-of-things (IoT) and cyber-physical system (CPS) applications that require prompt responses and collect sensitive user information [84, 85]. Wide-area data transmission from IoT sensors to cloud data centers often causes intolerable network latency [86, 87]. 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 [88, 89]. 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) [90, 91, 92]. The proliferation of IoT sensors and single-board computers (SBCs), *e.g.*, Raspberry Pi [93], enables the adoption of edge computing paradigm across a wide range of latency-sensitive applications, including emergency alert services, traffic monitoring, wearable cognitive-assistance systems, and augmented reality [94, 90].

However, there are computationally-intensive edge applications and tasks that still need offloading to clouds. A typical example is *deep learning* (DL) applications. While DL technologies are increasingly

leveraged in real-world edge applications, the majority of DL tasks are still relying on resources in cloud data centers rather than being processed at the Internet edge. [87, 95, 96]. 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 [97, 98, 99, 100, 101]. Therefore, emerging edge applications, such as autonomous driving, disaster response systems, drone-based surveillance [102, 103, 104], *may* offer poor quality of services (QoS) and user experience due to high network latency and privacy issues.

Significant efforts have been conducted to bring DL to the edge of the Internet. In particular, optimizations in hardware and software functionalities are carried out to ensure that the DL models fit in the resource-constrained devices. For example, model compression [105, 106] and quantization [107, 108] techniques are developed to downsize DL models without significant degrade in their accuracy [109, 110, 111]. Similarly, studies involving efficient partitioning of heavier models and subsequent offloading to the cloud [87, 112, 113] show how the edge-cloud hybrid model could be leveraged for DL inference tasks. Moreover, light-weight machine learning (ML) frameworks, *e.g.*, Tensorflow Lite [114], mobile neural network [115], are developed for enabling on-device inference. Finally, the advent of edge devices with GPUs, *e.g.*, Nvidia's Jetson Series [116, 117, 118], and AI accelerators, *e.g.*, Google's Coral Accelerator [119] and Intel's Neural Compute Stick [52], 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 [45, 46, 47, 48, 49, 50] have quantified the efficiency of various edge devices for DL inference tasks. However, most existing studies have focused on characterizing performance (*e.g.*, latency and throughput) of edge devices and AI accelerators with single DL tasks, which is a significantly limiting assumption in the DL deployment scenario. In real-world edge applications, AI multi-tenancy-based deployments are widely required, in which multiple DL tasks are co-running on edge devices. For instance, drone-based surveillance requires simultaneous executions of inference tasks on video and audio streams [120]. Furthermore, system approaches to maximize DL inference throughputs (*e.g.*, the number of inferences per second) on edge devices have not been deeply investigated.

In this chapter, we start by characterizing the performance (*e.g.*, inference throughput) of various edge devices and AI accelerators with a set of pre-trained DL models and popular DL frameworks. Through the characterization step, we obtain the baseline performance of various edge devices/AI accelerators and investigate factors that change the throughput of DL inference tasks on such devices. We then employ three techniques to maximize the DL inference throughput on the devices with single and multi-tenancy based use cases: *batched inferencing*, *concurrent model executions*, and *dynamic model placements*. Batched inferencing is for single-tenancy use cases on edge devices and maximizes the DL inference throughput by exploiting the parallel computing capabilities of computing resources on edge devices. Both concurrent model executions (CME) and dynamic model placements (DMP) are techniques for maximizing DL inference throughput with AI multi-tenancy. CME allows the deployment of multiple DL models on either GPU or EdgeTPU resources and runs them parallel to improve the overall DL inference throughput by simultaneously enabling the execution of different DL models. DMP enables AI multi-tenancy by de-

ploying and executing DL models on different resources on edge devices at the same time. *e.g.*, DL models on both GPU and EdgeTPU. DMP is particularly useful when AI accelerators (*e.g.*, EdgeTPU) enhance edge devices, and it can significantly increase the resource utilization and the DL inference throughput by utilizing multiple resources on the devices and the accelerators. Furthermore, we evaluate DL deployment strategy on edge devices with multiple AI accelerators (*e.g.*, EdgeTPU cluster) and report the benefits and limitations of the deployment scenarios with EdgeTPU clusters.

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

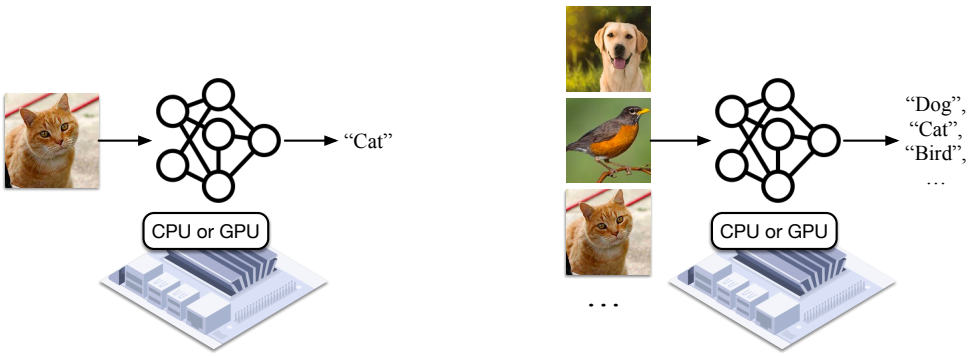
In particular, to the best of our knowledge, this work is the first study on evaluating DMP on heterogeneous edge resources/EdgeTPUs and characterizing the performance of EdgeTPU cluster for DL inference tasks.

4.1.1 Chapter Organization

The rest of the chapter is structured as follows. Section 4.2 provides the description of edge devices, EdgeTPUs, DL models, and DL frameworks used in this chapter. Section 4.3 describes the evaluation design and the benchmark tools used for accessing edge devices and AI accelerators. Section 4.4 reports evaluation results with single DL tasks (single-tenancy). Section 4.5 provides evaluation results when deploying multiple DL models (AI multi-tenancy). Section 4.6 provides the discussion and lesson learned from this research. Section 4.7 summarizes this chapter.

4.2 Approaches for Deep Learning Inference Throughput Maximization

The goal of this study is to investigate different system approaches for maximizing the DL inference throughput on various edge devices and AI accelerators. For the DL single-tenancy use cases, we investigate the batching approach. For the AI multi-tenancy use cases, two approaches are studied; *concurrent model executions* and *dynamic model placements*.



(a) Single-Tenancy with Single Batch

(b) Single-Tenancy with Batched Inferencing

Figure 4.1: DL inference for AI single-tenancy on edge devices. (a) shows single-batch inferencing. (b) shows batched inferencing.

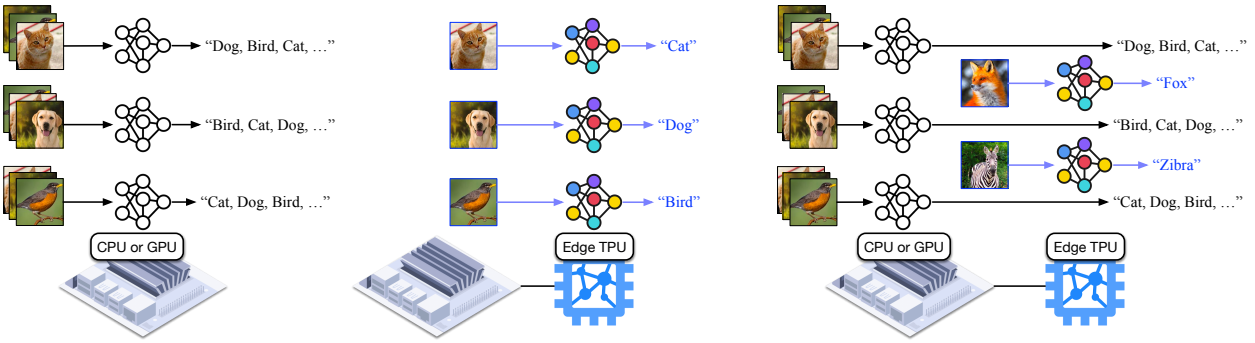
4.2.1 DL Throughput Maximization for AI Single-Tenancy on Edge Devices

For AI single-tenancy cases, where only one DL model is running on an edge device, we use *batched inferencing* (or multi-batch inferencing) to maximize the DL inference throughput. In the context of inference, *Batching* refers to enabling a single DL model to process multiple inputs simultaneously by implementing the concept of single instruction/multiple data operations. Fig. 4.1 illustrates single batch inferencing and batched inferencing in single-tenancy on edge devices.

In real-world use cases, batched inferencing is beneficial as edge devices are often required to handle batches of data either from multiple IoT sensors (*e.g.*, autonomous cars with multiple cameras) or from end devices that collect data over a period of time and send requests in batch (*e.g.*, traffic monitoring, wearable devices). Therefore, we investigate the impact of batched inferencing on the overall throughput, specifically on GPU-enabled devices (`J.Nano`, `J.TX2`, `J.Xavier`). The *batch size* (the number of input images to models) gradually increases in powers of two as GPUs can efficiently map virtual processing units onto physical processing units if the data to be parallelized is in the power of two [121]. Also, optimized libraries working with matrix operation can be effectively performed when processing batches of size in the order of powers of two [122].

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

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



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

Figure 4.2: DL throughput improvement techniques for AI multi-tenancy on edge devices. Fig. 4.2(a) and (b) illustrate CME on edge devices (using CPU or GPU) and on EdgeTPU respectively. Fig. 4.2(c) shows DMP on edge devices with AI accelerator (EdgeTPU).

Concurrent Model Executions (CMEs)

CME leverages the idea of parallel processing and enables AI multi-tenancy by simultaneously executing multiple DL inference tasks (models) on edge devices' resources (either GPU or EdgeTPUs). Fig. 4.2(a) and (b) illustrate the CME on edge devices and AI accelerators.

CME can provide two potential benefits to edge devices and EdgeTPUs; 1) improvement in the overall DL inference throughput and 2) ability to run multiple (often different) DL (*e.g.*, inference) tasks. However, due to the resource-constrained nature of edge devices (*e.g.*, limited memory sizes and the number of CPU cores), it is unclear how many DL models can be concurrently executed and which level of concurrency yields the maximum throughput. Therefore, it is important to empirically identify the upper bound of throughput improvement and the concurrency level (the number of co-running DL models) on the devices by CME. To this end, we will measure throughput changes with different levels of concurrency. The concurrency level obtained from the last successful execution will be considered as the maximum concurrency level supported by the edge devices and EdgeTPUs. Furthermore, because CMEs provide software-level parallelism, we will also evaluate the impact of introducing the idea of *EdgeTPU-cluster* (*e.g.*, running DL models on multiple `USB-Accelerators`) for maximizing the DL inferencing. Running concurrent models with more resources can lead to higher throughput, but other hardware bottlenecks like USB bandwidth can hinder the performance.

Dynamic Model Placements (DMPs)

DMP is another approach for maximizing DL inference throughput for AI multi-tenancy with the idea of leveraging heterogeneous computing resources. DMP leverages the collective powers of edge devices and `USB-Accelerator` (EdgeTPU). In other words, it allows running multiple DL models simultaneously

by placing DL models on an edge device's resource (CPU and/or GPU) and other DL models on EdgeTPUs. Fig. 4.2(c) shows the DMP approach for AI multi-temancy on edge devices and AI accelerators.

Furthermore, because `USB-Accelerator` relies on USB interfaces to connect edge devices, the potential benefits from DMP can be improved DL inference throughput using heterogeneous resources in both edge devices and `USB-Accelerator` and high resource utilization of both resources. However, DL inference tasks from both on-board edge resources and `USB-Accelerator` are managed by the host edge devices so that there can be a performance penalty from resource contention. Therefore, we will thoroughly investigate and analyze this penalty when employing DMP in section 4.5. Moreover, similar to CME, we also study the performance impact of using the EdgeTPU cluster for DMP.

4.3 Evaluation Process and Benchmarker Design

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

$$T_{single} = \frac{The\ number\ of\ inferences}{Total\ execution\ time} \tag{4.1}$$

However, the definition of *the number of inferences* varies with the type of experiment. For instance, when leveraging the single-tenancy with batched inferencing (feeding multiple input images into one DL model on a device), the number of inferences in equation-(4.1) is "batch size (bs)" $\times$ "the number of batches (bc)." The equation for batched inferencing throughput (T_{batch}) is formulated below.

$$T_{batch} = \frac{bs \times bc}{Total\ execution\ time} \tag{4.2}$$

On the other hand, when leveraging AI multi-tenancy with CME, the number of inferences will be calculated by "concurrency level (cc)" $\times$ "bs" $\times$ "bc". The DL inference throughput with CME (T_{cme}) is expressed in equation-(4.3).

$$T_{cme} = \frac{cc \times bs \times bc}{Total\ execution\ time} \tag{4.3}$$

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

$$T_{dmp} = \sum_{i=1}^{n} T_{cme_i}, \tag{4.4}$$

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

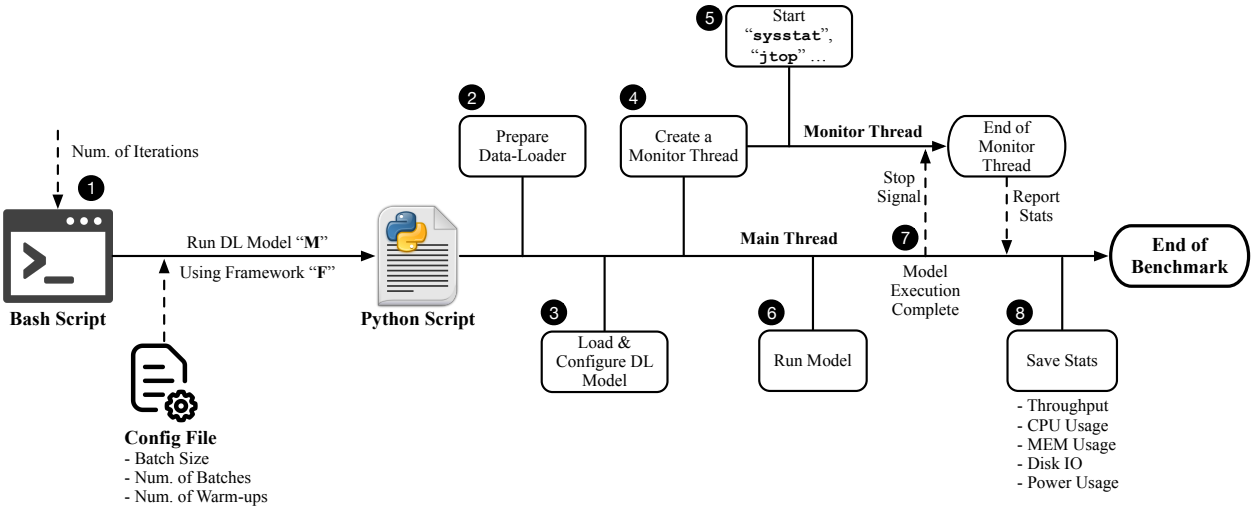


Figure 4.3: Benchmark Procedure

Benchmarker Design

We develop a benchmarker that measures the DL inference throughput and collects other necessary system statistics together. We deploy it along with an image classification application on the edge devices and EdgeTPUs accelerators. The measurement procedure of the benchmarker is illustrated in Fig. 4.3.

The benchmarker is invoked from a bash script (❶ in Fig. 4.3) that takes parameters in a config file specific to a measurement. The config file specifies the DL model, framework, and the number of iterations to run the experiment. The config file also contains other parameters that are common across experiments, such as the number of warmup executions to perform, the input batch size, the number of batches, and resources (CPU, GPU, or EdgeTPU) used for the inference task. The bash script then runs the benchmarker (written in Python) with all these configurations. Invoking the Python interpreter using the bash script ensures that the cache constructed and maintained by the Python runtime gets cleared with each new iteration.

The benchmarker then prepares a framework-specific data-loader (❷) that uses the validation dataset from `ImageNet ILSVRC-2012` to construct batches of inputs for the DL model. Next, the benchmarker initiates a DL framework as per the config file. It loads the DL model into the memory and configures the model to be executed on CPU, GPU, or EdgeTPU (❸). The next step (❹) is the warm-up phase, which ensures all the necessary components are loaded, and the DL framework configures suitable optimization strategies before performing the measurement. After the warm-up phase, the benchmarker creates a new background monitoring thread that captures system statistics while executing the model (❺). Simultaneously, the main thread of the benchmarker starts to run inference tasks using the DL model and the input data from the data-loader (❻). Once the pre-defined number of input batches is processed, the main thread instructs the monitoring thread to stop capturing system statistics (❼). Finally, a measurement

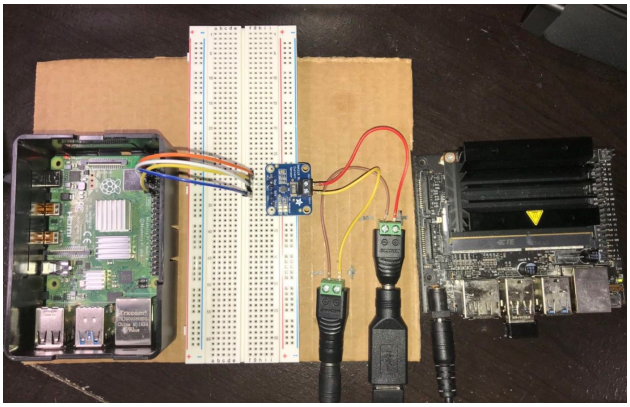


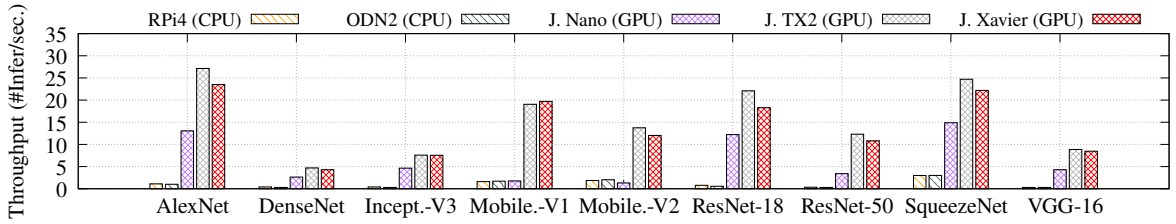
Figure 4.4: Experimental setup for power measurement. Power consumption by a target edge device is being measured and transmitted to a computing board using I2C cables by `INA-219` chip.

report of throughput and system statistics is saved (❽). This entire benchmarking process is performed at least 30 times for each set of configurations to guarantee the statistical confidence of the measurement.

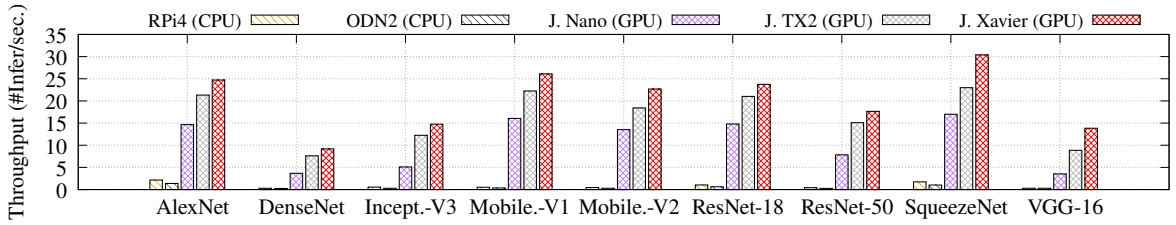
The monitoring thread is responsible for collecting diverse system statistics using `sysstat`, `usbtop`, and `jtop`. `sysstat` measures CPU and memory usage during the benchmark. `usbtop` measures USB IO bandwidth, and `jtop` collects power consumption statistics. However, `jtop` is only available to measure power-related statistics for Nvidia's Jetson devices. So, we also employ `INA-219` [123], a voltage, current and power measurement chip, for measuring the power usage of non-Nvidia devices (`RPi4` and `ODN2`). With a default resistance of $0.1\,\Omega$, the 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. We use `pi-ina219` [124], a Python library to communicate with the `INA-219` chip. The experimental setup with `INA-219` is shown in Fig. 4.4.

4.4 Evaluation with DL Single-Tenancy

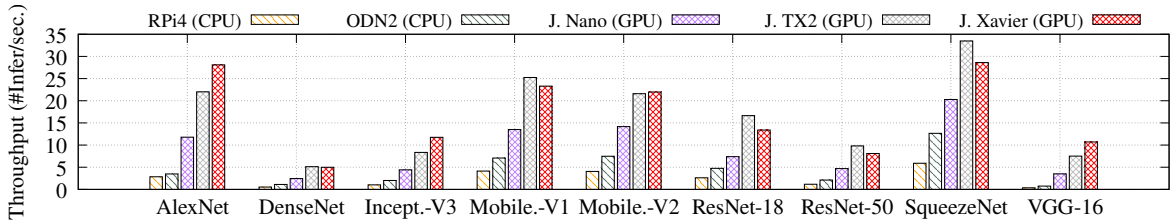
We first report the evaluation results with single-tenancy on edge devices and EdgeTPU accelerators. As single-tenancy is a common use case of running AI tasks at the edge, it can be used on edge devices (with CPUs or GPUs) or EdgeTPUs. Regarding the single-tenancy on edge devices, section 4.4.1 provides DL inference throughput with single-tenancy on edge devices. Moreover, as an approach for maximizing the inference throughput for single-tenancy, we evaluate the impact and performance of *batched inferencing*, where a DL model processes a batch of input images and outputs the classification results of all the images simultaneously. For single-tenancy on EdgeTPU, section 4.4.2 discusses the evaluation results on EdgeTPUs. Finally, section 4.4.3 thoroughly analyzes the experiment results and identifies the factors altering the DL inference throughput on edge devices and EdgeTPUs. The results reported in this chapter will serve as the baseline performance to evaluate throughput maximization approaches (CME and DMP) for AI multi-tenancy on edge devices.



(a) DL Inference Throughput on MxNet



(b) DL Inference Throughput on PyTorch



(c) DL Inference Throughput on TensorFlow

Figure 4.5: DL Inference Throughput Variations across Models, Edge Devices and DL Frameworks with a Batch Size of 1

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

The first set of experiments measured the inference throughput of all the DL models on edge devices with a batch size of 1. i.e., single input image per model per iteration. Fig. 4.5 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 GPUs while the results from `RPi4` and `ODN2` are measured on CPUs.

DL Model Size

Fig. 4.5 shows that the inference throughput varied significantly across different DL models. In particular, DL models with fewer parameters and floating-point operations (*e.g.*, `AlexNet`, `MobileNet-V1`,

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

GPU vs. CPU

Fig. 4.5 also confirms that, for single-batch inference, the GPU-based devices' DL inference throughput significantly outperformed the CPU-based devices' inference throughput. The edge devices with GPUs (J.Nano, J.TX2, and J.Xavier) processed 4× – 80× more inference requests than the devices without a GPU (RPi4 and ODN2) for all the models across all three frameworks. The advantage of using GPU is dominantly observed when DL inference using PyTorch. On average, J.Nano, J.TX2 and J.Xavier showed 17×, 30×, and 38× higher throughput than RPi4, respectively, and the DL inference throughput results on these devices were 38×, 62×, and 75× higher than ODN2. The results also showed the performance differences among the three GPU-based edge devices. Regarding the GPU performance, J.Nano has the least powerful GPU, which was clearly observed in the results. There was no clear winner between J.Xavier and J.TX2. But, J.TX2's GPU frequency (1.3 GHz) is slightly higher than that of J.Xavier's (1.1 GHz). This can be attributed as the reason why J.TX2's results were better, particularly when performing inferences on MxNet and TensorFlow. For PyTorch, however, we observed that J.Xavier outperformed J.TX2. This result was partly because of the difference in DL framework capability and partly due to the different GPU core numbers. PyTorch has better parallelization support than the other two frameworks. Also, J.Xavier has more GPU cores (384) than J.TX2 (256 GPU cores). As a result, PyTorch had better support in parallelizing its dynamic computation graph on more numbers of cores than the other two frameworks, hence producing higher DL inference throughputs.

DL Frameworks

Among the three DL frameworks, PyTorch showed the highest throughput on GPUs. On average, the throughput of DL models with PyTorch was 31% and 26% higher than MxNet and TensorFlow, respectively. PyTorch's superiority on GPUs was because of Torch library designed to make tensor operations on GPU faster and more efficient. On the other hand, TensorFlow significantly outperformed the other two on CPUs. The average throughput across all the models on CPUs using TensorFlow was about 5×

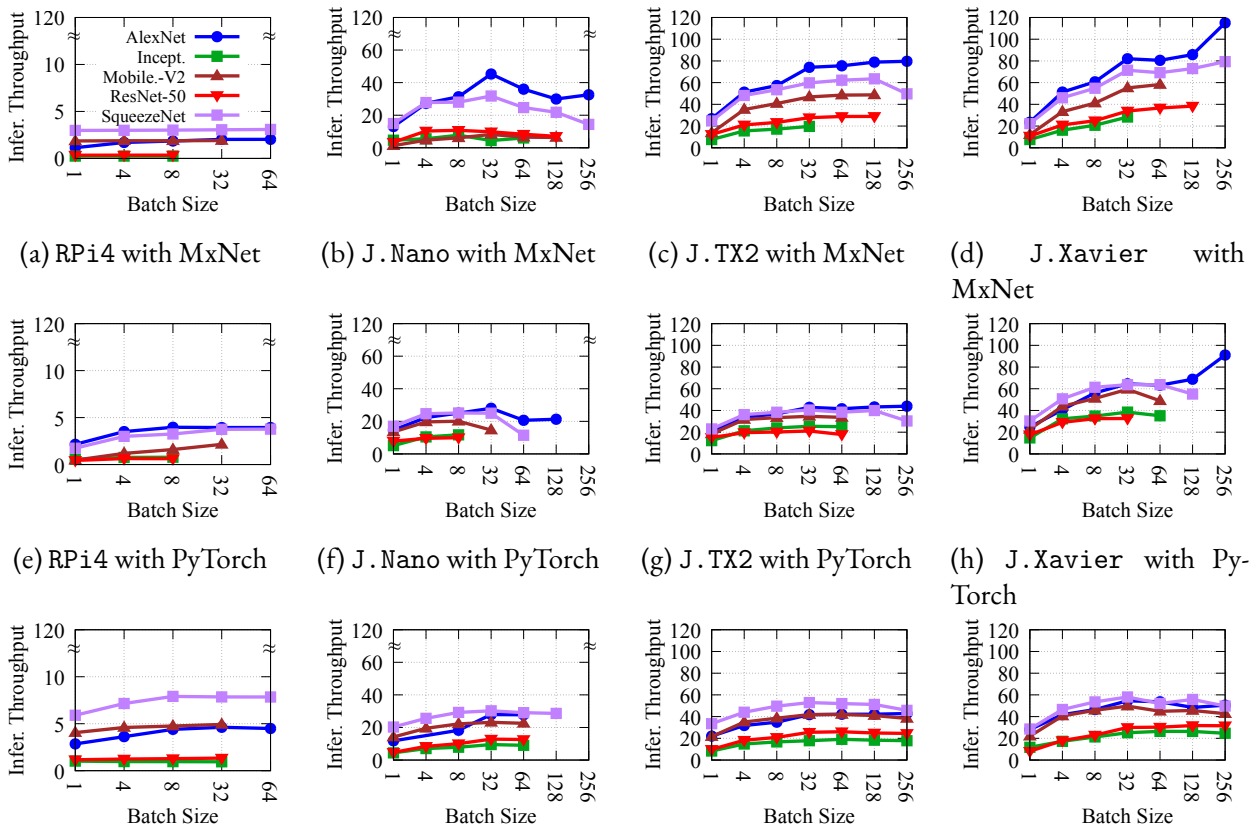
the results from MxNet and $10\times$ from PyTorch. This result was due to TensorFlow's design to support mapping nodes (from computational graph) across multicore CPUs [125]. Therefore, TensorFlow could enable faster computation, processing more DL inferences on CPUs than the other two frameworks.

MxNet, on all the devices, was the least performing framework. We observed that two `MobielNet` models' throughput with MxNet on `J.Nano` was exceptionally lower than the throughput with other frameworks. For example, both models processed less than 5 inferences per second, which was even $4\times$ less than the throughput of the same models with the other two frameworks. Such lower throughput is because of two reasons. First, when MxNet leveraged GPUs, the framework first performed an "auto-tunning" process that used Nvidia's cuDNN library to automatically tune convolution layers. In other words, MxNet tried to find the best performing convolution algorithm in its first run, enabling subsequent model executions to run faster. Unfortunately, this process is a highly memory-intensive operation, and `J.Nano`'s 4GB memory is not large enough to complete this process, frequently resulting in out-of-memory errors. Because of the above reason, the measurement had to be performed with disabling "auto-tune", meaning that MxNet relied on sub-optimal convolution algorithm. Second, the two `MobileNet` models performed frequent small-size memory-bound element-wise operations, such as ReLU [110]. Without optimization strategies like operator fusion [126] enabled, the processing time of such models increased steeply. Such performance tuning strategies worked only well on edge devices with larger memory size, but not on edge devices with smaller memory sizes like `J.Nano`.

Impact of Batched Inferencing

As discussed in section 4.2.1, batched inferencing is the approach to maximize the throughput for single-tenancy. Fig. 4.6 reports the DL inference throughput of *batched inferencing* with increasing batch sizes, on the three ML frameworks. Please note that Fig. 4.6 includes the results of five models on four devices due to the page limitation, and other omitted results have similar patterns. We observed that there was a significant throughput improvement with increasing batch size for GPU-enabled devices. On average, a batch size of 32 showed 240% higher DL inference throughput. The impact of batching on `J.Xavier`, with more GPU cores, was higher than in `J.TX2` and `J.Nano`. Specifically, `J.Xavier`'s results were 25% and 42% greater than `J.TX2` and `J.Nano`, respectively. Another interesting observation is that, while a larger batch size appeared to increase the throughput, large batch sizes (> 128) may not always improve the throughput. For example, when using `AlexNet`, the batch sizes of 4, 8, or 32 often showed higher throughput than the throughput with the batch sizes of 128/256 on `J.Nano` (Fig. 4.6b and Fig. 4.6f).

Another observation is that the inference throughput did not always increase as batch size increased. While inferencing without batching, there is an interval between computations in two activation layers in the model. With batched inferencing, the model can calculate other images' activation layers in the same batch and store the results in the memory for the next layer. The batch size is limited by an edge device's memory capacity [127]. If the memory is sufficient enough to store all the activations, that batch can be processed directly [128]. However, when the memory size is insufficient, batching would trigger the edge device's memory saving mechanism, such as swap space. The data in the memory will then be moved to the storage (SD card in edge devices). This mechanism can slow down the inference speed, hence



(a) RPi4 with MxNet (b) J.Nano with MxNet (c) J.TX2 with MxNet (d) J.Xavier with MxNet

(e) RPi4 with PyTorch (f) J.Nano with PyTorch (g) J.TX2 with PyTorch (h) J.Xavier with PyTorch

(i) RPi4 with TensorFlow (j) J.Nano with TensorFlow (k) J.TX2 with TensorFlow (l) J.Xavier with TensorFlow

Figure 4.6: Throughput Variation across DL Models, Edge Devices and DL Framework with Different Batch Sizes

decreasing inference throughput. Therefore, employing the right (or optimal) batch size is a critical factor for maximizing the DL inference throughput on edge devices.

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

Table 4.1: DL inference throughput of the three quantized DL models on EdgeTPUs

4.4.2 DL Inference Throughput on EdgeTPU with Single-Tenancy

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

Table 4.4.1 reports the DL throughput fluctuations on EdgeTPUs connected with different host devices, and the throughputs fluctuated across the different host devices. For example, when USB-Accelerator was connected with Jetson devices, its inference throughput could reach as high as 65 inferences per second for MobileNet-V1/V2 and 16 inferences per second for Inception-V3 However, the throughput decreases when using RPi4 and ODN2 as the host device. Several factors could result in such throughput fluctuations. Memory bandwidth on the (host) edge devices is one of the factors for such fluctuations. For example, the latency when swapping in/out of a DL model and its parameters between the host devices and USB-Accelerator rely on the memory bandwidth. Furthermore, USB IO can also change the DL inference throughput. Such factors will be analyzed in section 4.4.3.

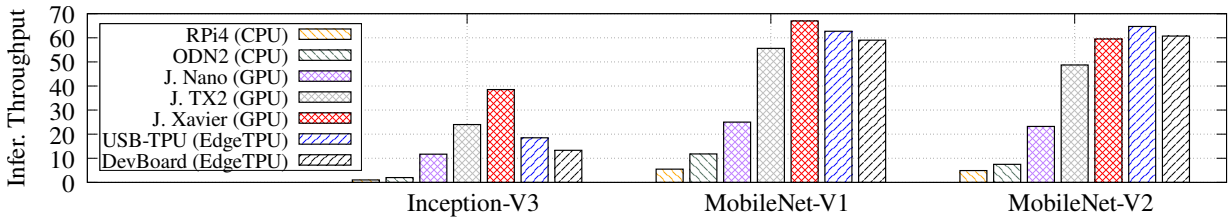
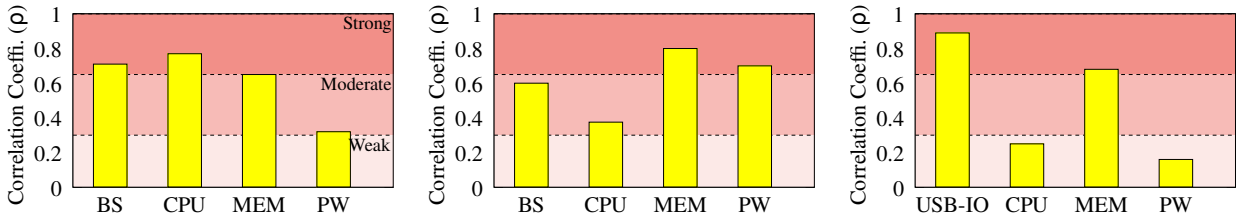


Figure 4.7: Comparison of inference throughput in CPU, GPU, and EdgeTPU. The throughput results of CPU- and GPU-based inferences are the maximum throughput results of those devices amongst all three frameworks. Please note that `USB-Accelerator`'s throughput in this graph is the maximum throughput from the results reported in Table 4.4.1.

The benefits of using EdgeTPUs are confirmed by comparing the inference throughput against other edge devices' (CPU- and GPU-based) throughput results. As shown in Fig. 4.7, `USB-Accelerator` and `DevBoard` showed significant improvement in DL inference throughput of all three models compared to the edge devices (`RPi4` and `ODN2`) relying on CPU resources. On average, EdgeTPU processed $8.5\times$ (`DevBoard`) and $9\times$ (`USB-Accelerator`) more image classification tasks than CPU-based inferencing.

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

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



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

Figure 4.8: Correlated factors that change the inference throughput. (BS: Batch Size, CPU: CPU usage, MEM: memory usage, PW: Power consumption, USB-IO: USB IO bandwidth usage)

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

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

Correlation Analysis Between System Factors and DL Inference Throughput. We first performed correlation analysis to investigate the factors that change the DL inference throughput on edge devices and EdgeTPUs. The correlation analysis was performed by calculating the Pearson Correlation Coefficient (equation-(4.5))between measured throughput results and resource usage statistics [129].

$$\frac{cov(x,y)}{\sigma_x \sigma_y} = \frac{\sum_i^n (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i^n (x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{4.5}$$

This coefficient represents the linear relationship between two variables, ranging from -1 to 1. Please note that the coefficient of 1 indicates an ideal positive correlation, negative values mean reverse correlation, and 0 means there is no correlation between two variables.

Model	Batch Size	Avg. Throughput	Avg. CPU Usage(%)
AlexNet	1	2.85	53.9
	32	4.63	100.0
DenseNet-161	1	0.53	76.5
	32	0.56	100.0
Inception-V3	1	1.02	81.0
	32	0.95	100.0
MobileNet-V1	1	4.14	59.2
	32	5.51	93.0

Model	Batch Size	Avg. Throughput	Avg. CPU Usage(%)
MobileNet-V2	1	4.05	60.9
	32	4.92	87.4
ResNet-18	1	2.61	73.1
	32	2.90	98.3
ResNet-50	1	1.16	72.8
	32	1.34	98.1
SqueezeNet-V1	1	5.89	53.3
	32	7.86	85.8

Table 4.2: Change in CPU usage and inference throughput (on TensorFlow) with varying batch sizes in RPi4

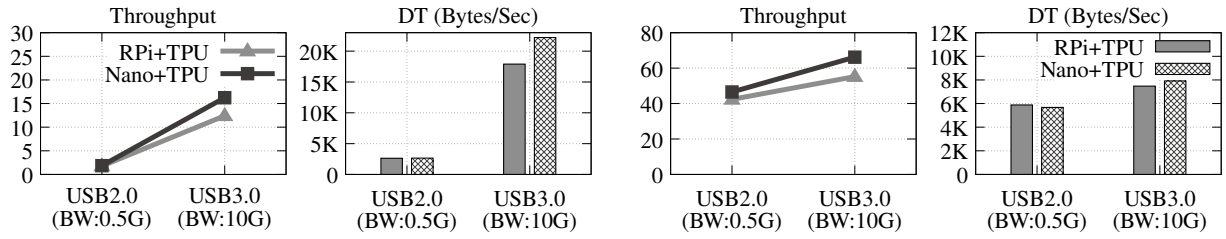
Fig. 4.8 shows the correlated factors for the DL inference throughput when using CPUs, GPUs, and EdgeTPUs. For the CPU-based inferences on RPi4, ODN2 (shown in Fig. 4.8a), the batch size, CPU, and memory strongly correlated with the inference throughput changes. This is because the CPU is the main computing resource for performing the DL tasks, and memory resources are used to load and store the DL models. The inference tasks with larger batch sizes naturally increase the input data for processing so that an increase in the batch sizes can improve the throughput until the limit of device resources. Table 4.4.3 summarizes the impact on throughput with increasing batch size and the corresponding increment in CPU usage on RPi4. Heavier models like DenseNet-161 and Inception-V3 did not show significant changes in throughput with increased batch size on CPUs because of the high processing demands, as highlighted by the 100% CPU usage. However, for the lighter models (AlexNet, MobileNet-V1/V2, SqueezeNet-V1), on average, there is a gain of 40% in throughput with a nearly 70% increase in CPU usage with increasing batch size.

For the GPU-based inference tasks on J.Nano, J.TX2, and J.Xavier (shown in Fig. 4.8b), memory, power consumption, and inference batch sizes had a relatively stronger correlation with the DL inference throughput. Specifically, the power consumption showed a strong correlation with the throughput as the GPU module in edge devices consumed more power than typical CPUs in edge devices. We further observed that, on average, a 15 – 20% increase in power usage corresponded to a 90 – 100% gain in throughput. Regarding the batch sizes, as we discussed in section 4.4.1, increasing batch size could significantly change the DL inference throughput, and it was clearly observed with the correlation analysis. On the other hand, CPU, as expected, showed a relatively weaker correlation with the DL inference throughput as CPU is primarily used for managing the device and non-DL tasks rather than performing GPU-based inference tasks.

For the inference tasks on EdgeTPU accelerators (especially USB-Accelerator), as shown in Fig. 4.8c, the USB bandwidth between a host edge device and the USB-Accelerator and memory usage on host edge devices had a strong correlation with the inference throughput. Both memory and USB IO were closely related to each other for executing DL models on USB-Accelerator. Because USB-Accelerator does not have main memory (RAM)[1], it relies on the host device's memory system to store models and uses context switching to swap models/parameters between the host device's RAM and EdgeTPU to perform DL inference tasks. Therefore, low USB IO bandwidth between the host device and USB-Accelerator limited data rates for switching models/parameters so that the throughput could be decreased.

Impact of USB Bandwidth on USB-Accelerator's DL Inference Throughput. To investigate the impact of the USB IO bandwidth, we measured the inference throughput changes on USB-Accelerator by connecting it to two edge devices (RPi4 and J.Nano) with different USB interfaces; (a) USB 2.0 with up to 0.5GB of bandwidth, (b) USB 3.0 with up to 10GB of bandwidth. Fig. 4.9 shows that USB's IO bandwidth could considerably change the inference throughput of EdgeTPUs. With larger IO bandwidth, RPi4 achieved 1.3× (MobileNet-V2) and 7× (Inception-V3) higher throughput when moving from

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



(a) `Inception-V3` (b) `MobileNet-V2`

Figure 4.9: Difference in DL inference throughput and data transfer with USB 2.0 and 3.0 interfaces. (DT: Data Transfer Rate)

USB 2.0 to 3.0. `J.Nano` also showed 1.4× (`MobileNet-V2`) and 8.7× (`Inception-V3`) higher throughput with USB 3.0 than that with USB 2.0. Larger USB IO bandwidth facilitated faster swapping of model parameters and input data between the host device and `USB-Accelerator`, enabling faster DL inferences.

4.5 Evaluation with AI Multi-Tenancy

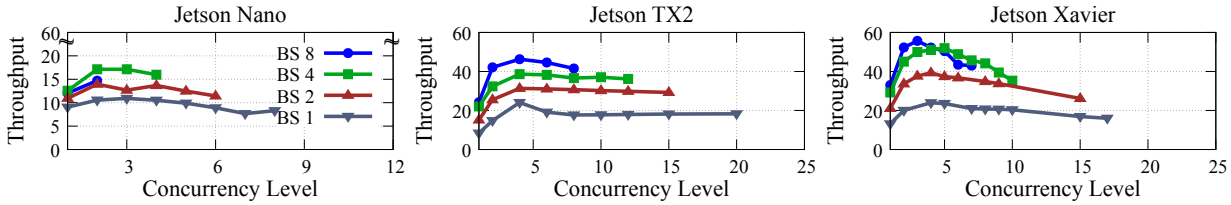
CME (Concurrent Model Execution) and DMP (Dynamic Model Placement) are two approaches to maximizing the DL inference throughput with AI multi-tenancy. This secction reports our measurement results with CME and DMP and discusses the benefits and limitations of both approaches.

4.5.1 Concurrent Model Executions (CME)

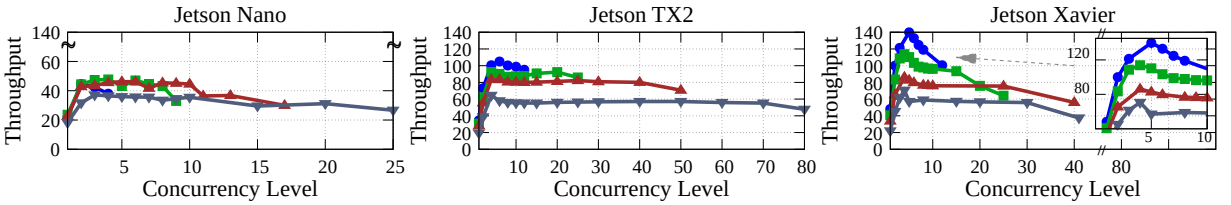
We first describe the evaluation procedure of CME and then report CME measurement results on GPUs on edge devices and EdgeTPUs. Finally, we will discuss CME results with a cluster of EdgeTPUs. In this evaluation, we seek answers to the following research questions:

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

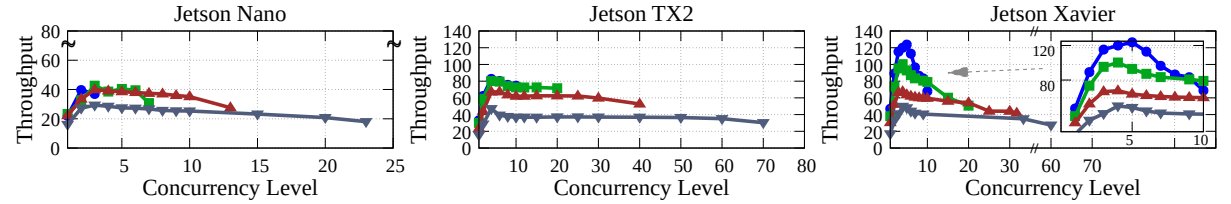
For the rest of this study, we only use three DL models, `Inception-V3`, `MobileNet-V1`, and `MobileNet-V2`, because pre-trained versions of these models are *officially* available for all the edge devices, including EdgeTPUs. Furthermore, we also exclude TensorFlow from this CME evaluation due



(a) `Inception-V3` with CME (PyTorch)



(b) `MobileNet-V1` with CME (PyTorch)



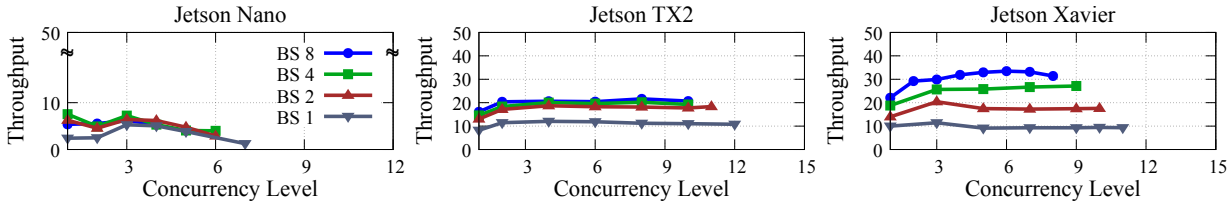
(c) `MobileNet-V2` with CME (PyTorch)

Figure 4.10: Concurreny measurement results on `J.Nano`, `J.TX2` and `J.Xavier` GPUs with PyTorch (BS: Batch Size)

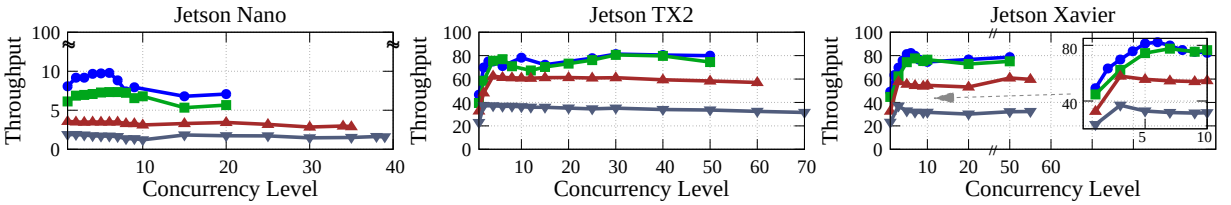
to software issues with `kerascv` [130] and `tf.Graph` [131] APIs, which do not fully support concurrent executions (*e.g.*, not thread-safe). Finally, regarding the throughput calculation with CME, we use equation-(4.3) to calculate DL inference throughput in section 4.3.

Evaluation Procedure

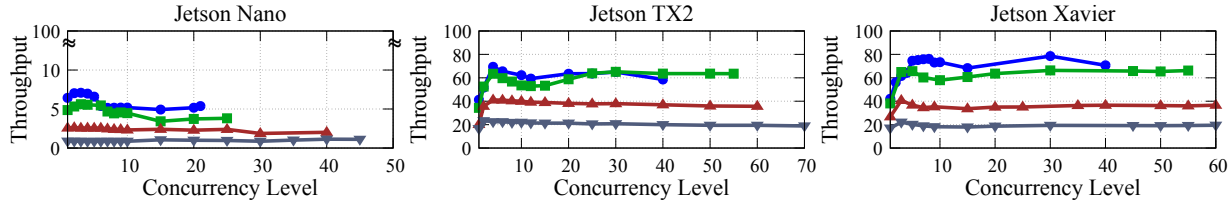
Based on the measurement results from single-tenancy cases (section 4.4), we gradually increase the number of co-running DL models ("*concurrency level*") on the devices and EdgeTPU to find the maximum level of concurrency and throughput improvement with CME. This process continues until the benchmarker fails to run for one of the following reasons. 1) the memory is fully saturated or 2) the device can no longer create more DL tasks. The concurrency level obtained from the last successful execution is considered as the maximum concurrency level supported by the edge devices and EdgeTPUs. In this measurement, we only report the results with leveraging CME on GPUs (`J.Nano`, `J.TX2` and `J.Xavier`) and EdgeTPUs (`DevBoard` and `USB-Accelerator`). We omit the measurement results from CPU-based inferencing (*e.g.*, `RPi4` and `ODN2`) because, while we could find some benefits of CME on CPUs, *e.g.*, six concurrent



(a) `Inception-V3` with CME (MxNet)



(b) `MobileNet-V1` with CME (MxNet)



(c) `MobileNet-V2` with CME (MxNet)

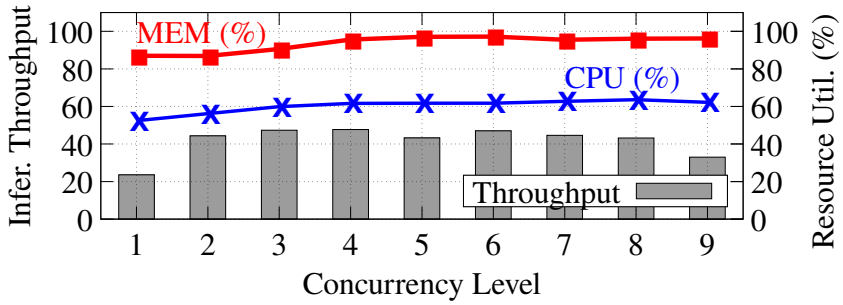
Figure 4.11: Concurreny measurement results on `J.Nano`, `J.TX2` and `J.Xavier` GPUs with PyTorch (BS: Batch Size)

models could be executed on CPUs of `RPi4` and `ODN2`, the throughput benefits were marginal. The measured throughput results were exceptionally lower than the results with CME on either GPUs or EdgeTPUs.

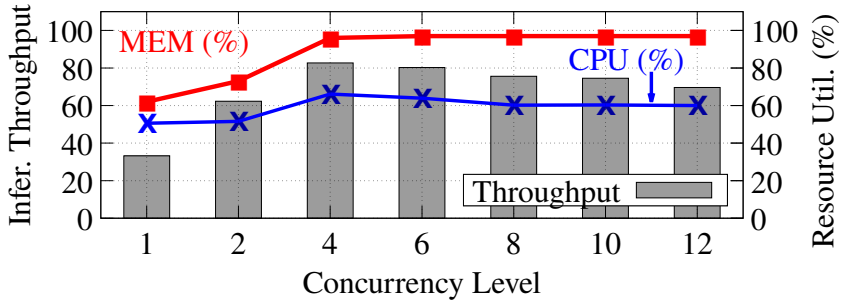
The benchmarking process described in Fig. 4.3 (section 4.3) is tweaked such that instead of running a model in the main thread (❻ in Fig. 4.3), new threads are created to run models concurrently (i.e., separate copies of the model are created for each thread). The main thread then waits for all the models to finish execution and finally terminates the script, followed by steps similar to the previous workflow.

CME Evaluation Results on GPU in Edge Devices.

The next evaluation is to measure the DL inference throughput of GPUs with CME and increasing concurrency levels using PyTorch (Fig. 4.10) and MxNet (Fig. 4.11), respectively, on `J.Nano`, `J.TX2`, and `J.Xavier`. As shown in the results, CME can significantly improve the overall DL inference throughput on GPUs by executing multiple DL inference tasks simultaneously. Similar to the results of batched inferencing, CME of lighter models like `MobileNet-V1/V2` (Fig. 4.10b, 4.10c, 4.11b, and 4.11c) yielded



(a) `MobileNet-V1` (`J.Nano`)



(b) `MobileNet-V2` (`J.TX2`)

Figure 4.12: Resource utilization and inference throughput changes with CME (PyTorch). `J.Nano` uses a batch size of 4, and `J.TX2` employs a batch size of 8.

higher gain in throughput while heavier models, *e.g.*, `Inception-V3` (Fig. 4.10a and 4.11a), showed minor improvement. In particular, compared to single-tenancy with bached inferencing, CME resulted in 1.4× – 2×, 1.8× – 2.7×, 1.7× – 3.0× increase in overall throughput on `J.Nano`, `J.TX2`, `J.Xavier` respectively, across all the three models with PyTorch (Fig. 4.10). The results with MxNet (Fig. 4.11) were less impressive in that relatively lower throughput improvements were observed; 1.3× – 1.5× on `J.TX2` and 1.5× – 1.8× on `J.Xavier`. `J.Nano`'s results with MxNet were particularly low. The worst-case we observed was 13% lower throughput than single-tenancy. The `J.Nano`'s low throughput was because the `J.Nano`'s experiments were performed by disabling MxNet's `cuDNN auto-tune` [132] process as `J.Nano`'s memory size (4 GB) was not sufficient enough to perform such memory intensive optimization. Enabling or disabling `auto-tune` option could significantly impact DL inference throughput. If `auto-tune` was enabled, MxNet first ran a performance test to seek the best convolutional algorithm, and the selected algorithm was then used for further inference tasks.

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

throughput improvement. After that, we observed increasing either of the two parameters (concurrency level and batch size) resulted in lower performance.

The level of concurrency was directly related to the size of the model and the available memory in edge devices. `J.Nano` could run 8 (`Inception-V3`) to 25 (`MobileNet-V1/V2`) models concurrently on GPU while `J.TX2` and `J.Xavier` were able to run approximately 25 (`Inception-V3`) to 80 (`MobileNet-V1/V2`) models on their GPUs simultaneously when using a batch size of 1. With increased batch sizes, the level of concurrency decreased as lesser memory became available. Fig. 4.12 shows that the maximum throughput highly correlated with memory utilization. As shown in the figures, after reaching the maximum throughput, the throughput was either decreased or stabilized with high memory utilization (or memory saturation). It is worth noting that the high correlation between memory utilization and throughput increase is consistent with our observation reported in Fig. 4.8.

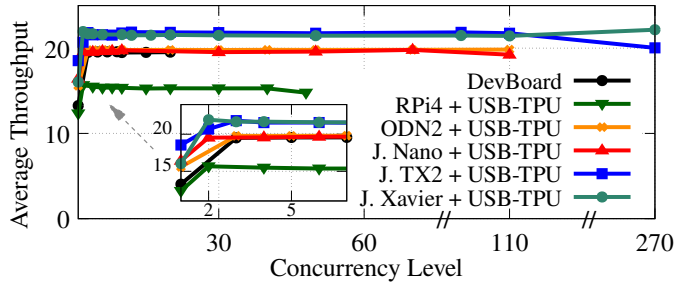
CME Evaluation Results on EdgeTPUs

The second evaluation for CME was to measure the DL inference throughput on EdgeTPUs, and Fig. 4.13 shows CME results on TPUs (both `DevBoard` and `USB-Accelerator`). The result includes all the combinations of edge devices and `USB-Accelerator` as well as `DevBoard`.

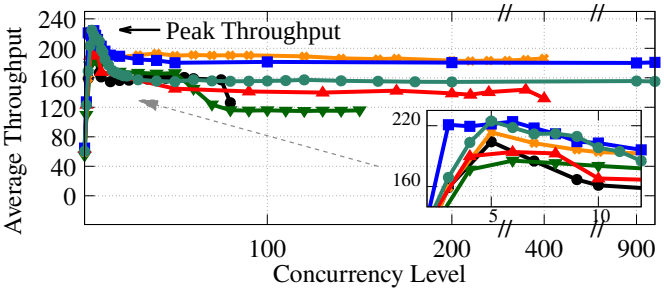
Like the previous GPU results, CME on EdgeTPUs also increased throughput over the single-tenancy cases. For `Inception-V3` (Fig. 4.13a), `DevBoard` showed $1.3\times$ higher throughput, and `USB-Accelerator` had $1.25\times$ improved throughput over single-tenancy cases. For both `MobileNet-V1` (Fig. 4.13b) and `MobileNet-V2` (Fig. 4.13c), EdgeTPUs showed $3.3\times$ higher throughput over the single-tenancy cases.

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

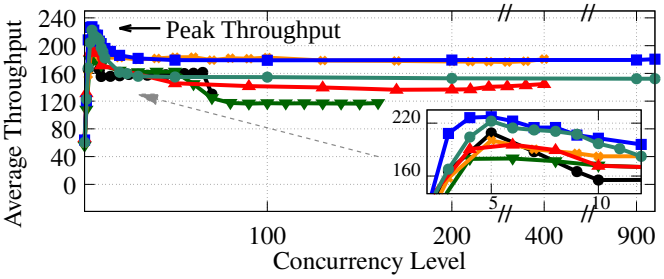
The second observation (Fig. 4.13b and 4.13c) – *the maximum throughput gain of `Mobile-Net-V1/V2` was achieved with a lower concurrency level* – was because the EdgeTPU cache was enough to load even multiple smaller models simultaneously. While EdgeTPU could leverage only one model at a time, other loaded models in its cache could obtain data from the host device's memory, hence minimizing the delay when switching models in EdgeTPUs. On the other hand, if the concurrency level was high, frequent



(a) `Inception-V3` (EdgeTPU)



(b) `MobileNet-V1` (EdgeTPU)

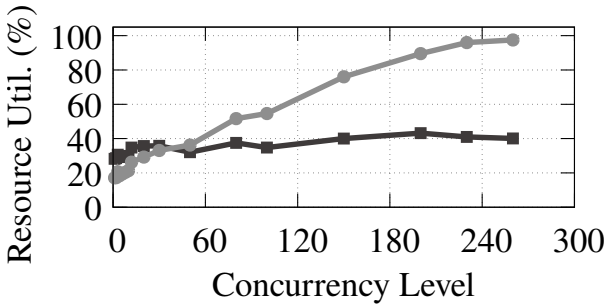


(c) `MobileNet-V2` (EdgeTPU)

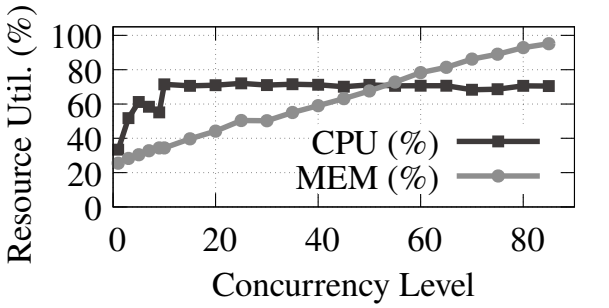
Figure 4.13: Results of CME measurement on EdgeTPU

model swaps needed to be frequently performed in EdgeTPU's cache, resulting in increased data transfer between EdgeTPU and the host edge device's memory. As a result, USB IO was quickly saturated, which in turn, decreased throughput. This was why both `MobileNet-V1` and `MobileNet-V2` reached the maximum throughput with a low concurrency level, and throughput could be decreased or stabilized with higher concurrency levels. This analysis suggests that, when using CME on EdgeTPU, model size and concurrency level should be carefully determined to maximize the throughput. Moreover, model compression techniques [84], *e.g.*, quantization and parameter pruning, should be considered for optimizing model size for EdgeTPUs.

Furthermore, all three models reported much higher concurrency levels on EdgeTPUs than on GPUs. For example, `DevBoard` supported the concurrency level of 20 for `Inception-V3` and 80 – 85 for both



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



(b) `MobileNet-V2` (DevB)

Figure 4.14: Resource utilization changes with increased concurrency level (EdgeTPUs)

`MobileNet-V1/V2` models. `USB-Accelerator` reached various maximum concurrency levels, and `USB-Accelerators`' concurrency levels vary considerably across different host edge devices. For example, for `Inception-V3`, the maximum concurrency level from `USB-Accelerator` with `RPi4` was 48, but the maximum concurrency level when it uses `J.TX2` or `J.Xavier` as the host device reached 270. For `MobileNet-V1/V2`, the maximum concurrency level from `USB-Accelerator` with `RPi4` was about 160, but it could be 1100 when leveraging `J.TX2` or `J.Xavier` as the host device.

Regarding the varying concurrency levels, Fig. 4.14 shows resource utilization changes with different concurrency levels measured on `USB-Accelerator` with `J.TX2` and `DevBoard`. We observed that memory utilization increased as the concurrency level increased. The maximum concurrency level was determined when the memory utilization reached close to 100%, indicating that memory size and bandwidth often limited the supported concurrency level DL models when enabling CME on `USB-Accelerator`.

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

Finally, compared with CME on GPUs, the maximum throughput of EdgeTPUs was nearly 230 inferences per second when running concurrent `MobileNet-V1/V2`. And this maximum throughput

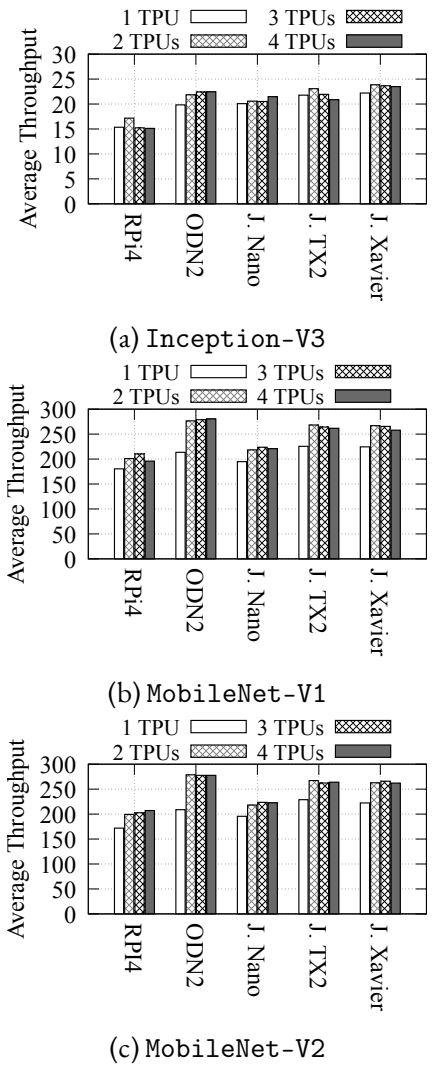
Figure 4.15: EdgeTPU-cluster composed of four EdgeTPUs (`USB-Accelerator`) connected with `J.Xavier`

was almost 2× higher throughput than CME on GPUs. However, this was not the case for larger models like `Inception-V3`. The maximum achievable throughput with `Inception-V3` using EdgeTPUs was nearly half the value when using GPUs. These observations suggest that careful consideration of model (size) and device (GPU or EdgeTPU) is critical to maximizing the overall throughput.

CME evaluation results on EdgeTPU Cluster

`USB-Accelerator` could significantly improve DL inference throughput with CME as shown in the previous section. To maximize the performance of `USB-Accelerator`, we performed further evaluations using multiple `USB-Accelerator` (called *EdgeTPU cluster*) with CME. The idea of EdgeTPU cluster is to connect *more than one* `USB-Accelerators` with a host edge device and employ CME on each `USB-Accelerator` (shown in Fig. 4.15). To this end, we used `Edge TPU Python API` [133] to load models in specific devices to ensure an equal number of models were running on all the `USB-Accelerators`.

We started the experiment by running each of three quantized models (`Inception-V3`, `MobileNet-V1`, `MobileNet-V2`) on two `USB-Accelerators` simultaneously on each device. Similar to the previous CME evaluations, the experiment was repeated for multiple levels of concurrency to find one that produced the maximum throughput. Once we found the maximum throughput with two accelerators, we gradually increased the number of `USB-Accelerator` (as well as the concurrency level) and repeated this procedure until the throughput reached the maximum with four `USB-Accelerators` with CME. The measurement results are reported in Fig. 4.16. In general, EdgeTPU-cluster could increase the DL inference throughput over a single accelerator, but the throughput differences among two, three, and four `USB-Accelerator` were marginal. In particular, when using EdgeTPU-cluster composed of two accelera-



(a) `Inception-V3`

(b) `MobileNet-V1`

(c) `MobileNet-V2`

Figure 4.16: DL inference throughput variation with multiple `USB-Accelerators` (EdgeTPU cluster)

tors, on average, we observed a 15 – 30% increase in throughput over a single accelerator for `MobileNet-V1` (Fig. 4.16b) and `MobileNet-V2` (Fig. 4.16c) across all devices. Such throughput increase was because two models could simultaneously be processed in two accelerators while, for one accelerator, the inference task had to wait for the single EdgeTPU to finish the current task before moving on to the next task. Moreover, with two `USB-Accelerators`, the throughput improvement could not reach $2\times$ as desired because the USB ports in all devices have an *internal shared hub* for the USB interface. This shared USB hub caused a certain delay (due to its serial processing characteristics) in data transfer, hence increasing the overall latency. For `Inception-V3` with two accelerators (Fig. 4.16a), the throughput improvement was smaller than the improvement observed with `MobileNet-V1/V2`. As mentioned earlier, `Inception-V3` was

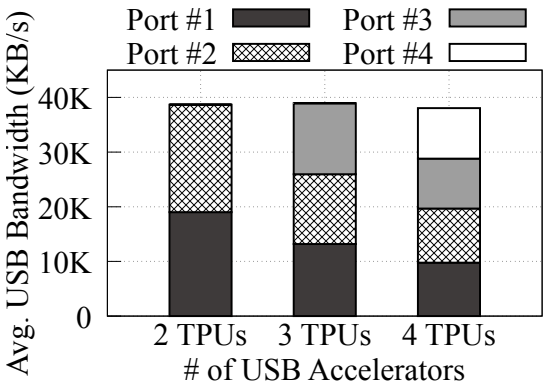


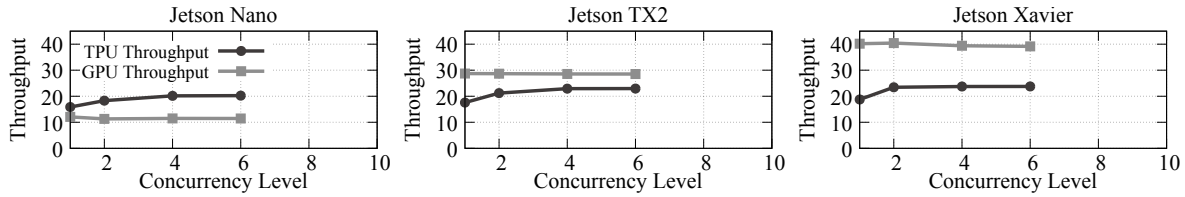
Figure 4.17: Total USB bandwidth usage and bandwidth consumed by each `USB-Accelerator` when using EdgeTPU-cluster with `J.TX2`.

too large to fit in a `USB-Accelerator` and thus requires constant swapping of model parameters with the host device. This, along with the serial nature of USB ports due to the presence of the shared hub, limited the parallelism that could have been achieved from multiple EdgeTPU-cluster with `USB-Accelerators`.

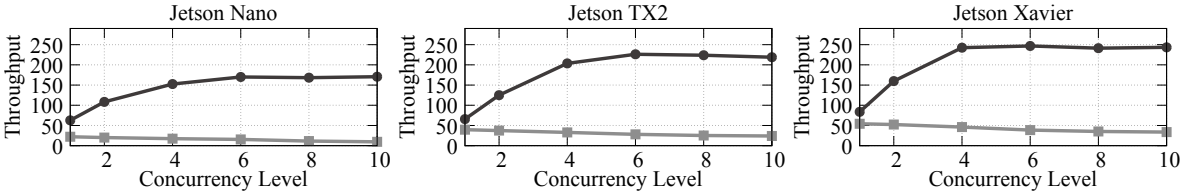
The EdgeTPU-cluster with three or four `USB-Accelerators` did not show meaningful throughput improvement over the cluster with two accelerators. Such limited throughput improvement was also due to the internal shared hub for the USB interface, which limited the total USB bandwidth while the device has the increased number of accelerators. Fig. 4.17 shows the overall bandwidth as well as the bandwidth consumed by each accelerator with `J.TX2` when running `MobileNet-V2`. It is observed that the bandwidth available to each accelerator decreases with every addition of a `USB-Accelerator`. This reduced bandwidth and data transfer rate directly degrade the overall performance of the `USB-Accelerators` and negate the benefits of having extra processing powers (EdgeTPU). Based on this evaluation, using two accelerators with CME appears to be the most effective approach to maximize DL inference thorughbput when using EdgeTPU-cluster.

4.5.2 Dynamic Model Placements (DMP)

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



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



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

Figure 4.18: Throughput changes with different concurrency level on both GPU and EdgeTPU when enabling DMP. We omit the results of `MobileNet-V2` because the results are similar to the results of `MobileNet-V1` (Fig. 4.18b)

1. What are the performance benefits (e.g., DL inference throughput) from DMP on heterogeneous resources?

2. What are the actual performance penalties of using DMP, compared to CME for AI multi-tenancy?

3. What are the performance benefits and limitations of using EdgeTPU-cluster for DMP?

Similar to the previous CME evaluations, we used three DL models (`Inception-V3`, `MobileNet-V1`, and `MobileNet-V2`) because these models could perform inference tasks on all resource types in edge devices and EdgeTPUs. Moreover, as CME showed significant throughput improvement in our previous evaluation, we enabled CME on both edge devices and `USB-Accelerator` when measuring the inference throughput with DMP. To calculate the throughput, we used equation-(4.4) in section 4.3.

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

DMP Evaluation Results on Edge Device and a single `USB-Accelerator`

As described above, CME was also enabled for DMP. The first step of this evaluation was to find an empirically optimal concurrency level that could produce the maximum DL inference throughput. While

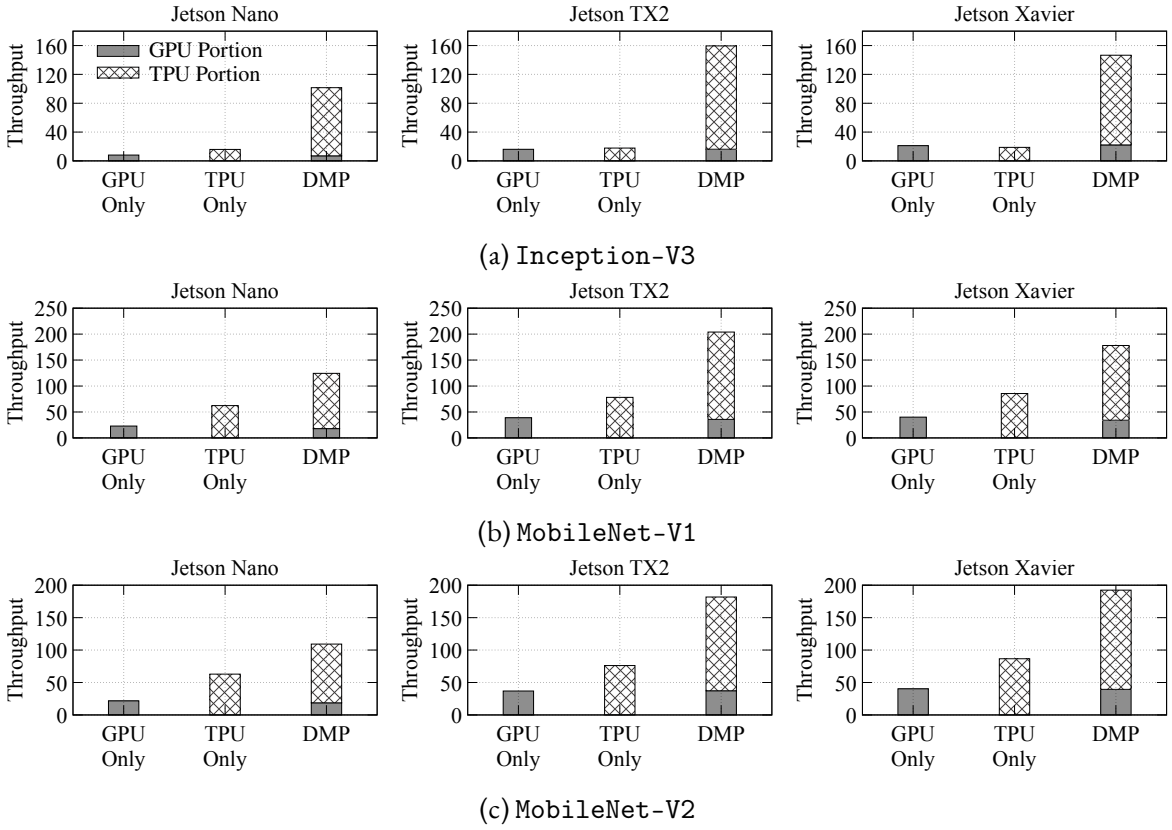


Figure 4.19: Comparison of DL inference throughput between DMP and single-tenancy

we reported the throughput changes with different concurrency levels on either GPU or EdgeTPU in section 4.5.1, such high concurrency levels *may* not be achievable for DMP. This is because the edge device needs to manage multiple inference tasks on both GPU and EdgeTPU, and there will be a contention of the edge device's resources (*e.g.*, memory). Therefore, we re-measured the throughput changes with different concurrency levels on both GPU and EdgeTPU for DMP. The evaluation results are shown in Fig. 4.18. As expected, much lower levels of concurrency were supported by edge devices and `USB-Accelerator`. We further observed that memory resources on the edge device were quickly saturated with lower levels of concurrency because the device needed to handle more inference tasks and frameworks for DMP.

Then, we measured the overall throughput with DMP, and the evaluation used to concurrency levels for GPU and EdgeTPU that could produce overall (accumulated) maximum DL inference throughput. Fig. 4.19 shows DMP's DL inference throughput improvement against the single-tenancy cases. All `J.Nano`, `J.TX2`, and `J.Xavier` showed significantly increased throughput compared to the single-tenancy GPU or EdgeTPU-based inferences. On average (across all three models), `J.Nano` had $6.2\times$ improved throughput over the single-tenancy on GPU and $2\times$ increased throughput over the single-tenancy on EdgeTPU. Both `J.TX2` and `J.Xavier` also showed significant throughput improvement for all three

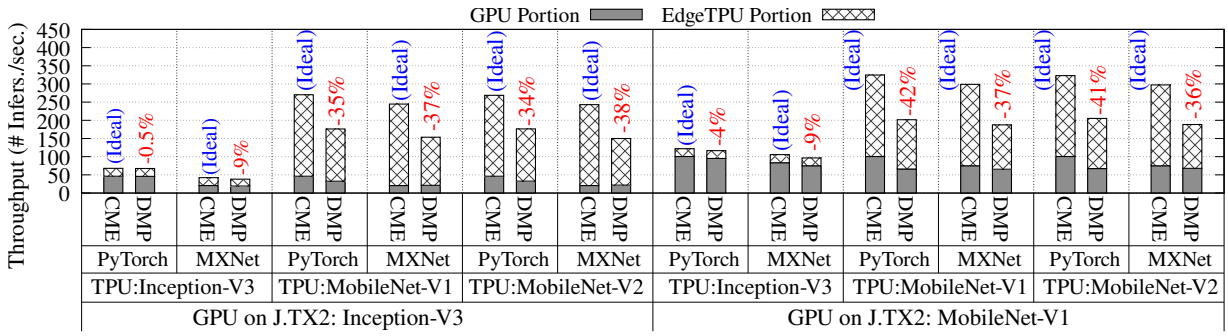
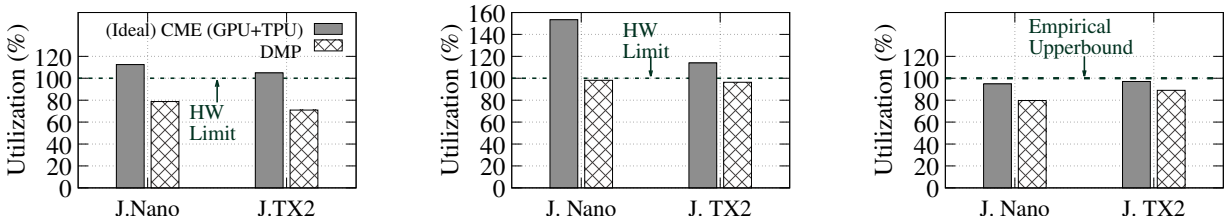


Figure 4.20: J.TX2's DL inference throughput comparison between (ideal) results from CME and DMP. The (ideal) results from CME are calculated by the sum of separately measured CME throughput on GPU and EdgeTPU.



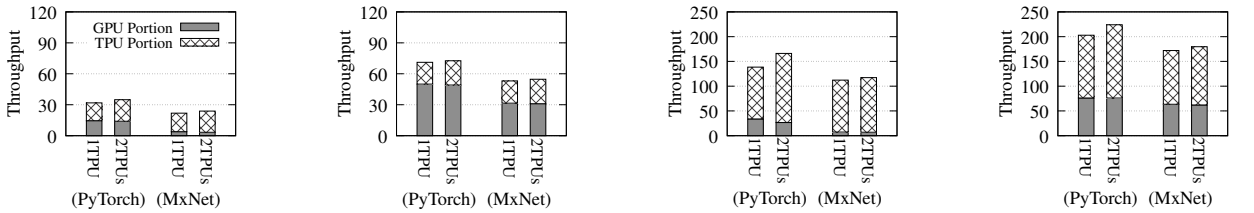
(a) CPU Usage (b) Memory Usage (c) USB-IO Usage

Figure 4.21: Resource usage comparison between (ideal) sum of CME on GPU/EdgeTPU and DMP.

models ranging from $2\times - 9.9\times$ over GPU-only or EdgeTPU-only inferencing with single-tenancy (with bached inferencing). However, this improved throughput can be in part due to leveraging both CME and DMP. Therefore, we also compare the DL inference throughput between the ideal throughput upper bound based on CME results (section 4.5.1) and DMP. Please note that the ideal throughput upper bound is calculated by accumulating GPU throughput with CME and EdgeTPU throughput with CME measured separately.

Fig. 4.20 reports the throughput comparison between (ideal) CME results and DMP results. The figure contains the results measured from J.TX2 when using PyTorch/MXNet (for GPU) and TFLite (for EdgeTPU). Please note that we omit the results from `J.Nano` and `J.Xavier`, but those omitted results show similar patterns with Fig. 4.20. As shown in the figure, while the differences between the ideal throughput and DMP's throughput varied with DL models and DL frameworks, `J.TX2` with DMP showed 34.6% and 25.3% lower DL inference throughput than the ideal throughput with CME (on both GPU and EdgeTPU). Such differences were mainly due to the resource contention and resource limits in the edge devices.

To understand the gap between the DMP's throughput and ideal throughput, we performed further analysis on resource consumption. Fig. 4.21 shows the resource utilization (CPU, memory, USB IO)



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

Figure 4.22: Throughput comparison between DMP with 1 EdgeTPU and DMP with 2 EdgeTPUs (EdgeTPU-Cluster)

between the ideal sum of CME on GPU/EdgeTPU (measured in section 4.5.1) and DMP. The figure shows that the ideal throughput often could not be achievable with current HW specifications. Specifically, CPU (Fig. 4.21a) and memory (Fig. 4.21b) utilization should exceed the HW limits of the edge devices (more than 100%) to reach such high throughput. Moreover, similar to the CME analysis, memory was identified as a critical resource when enabling DMP. Specifically, we observed that memory utilization reached 100% with DMP while CPU utilization did not reach 100%. Based on this observation, the DL inference throughput, when the memory resource is saturated, can be the empirical performance upper bound when enabling DMP. We also observed that resource contention could impact the DL inference throughput because the shared resources, such as memory and CPU, were needed to manage multiple DL models running on different resources. The decreased USB IO utilization (about 8% to 15%) with DMP (Fig. 4.21c) was because of such resource contentions, and the reduced USB IO utilization could decrease DL inference throughput from in USB-Accelerator.

DMP Evaluation Results on Edge Device and EdgeTPU-cluster

The next evaluation is to measure the throughput improvements of DMP when leveraging EdgeTPU-cluster. As shown in section 4.5.1, EdgeTPU-cluster with two accelerators showed almost maximum performance improvement due to the limitation of USB bandwidth. Therefore, in this evaluation, we used a EdgeTPU-cluster with two accelerators.

Fig. 4.22 reports DL inference throughput changes from DMP employing EdgeTPU-cluster. Please note that the figure only shows the results from `J.Nano` and `J.Xavier` due to the page limitation. Also, we only report the combination of `Inception-V3` models on GPU and EdgeTPU-cluster (Fig. 4.22a and Fig. 4.22b) and `MobileNet-V1` on GPU and EdgeTPU-cluster (Fig. 4.22c and Fig. 4.22d). Other omitted results showed similar results with Fig. 4.22. As shown in the figure, when using EdgeTPU-cluster on DMP, we observed a marginal improvement in DL inference throughput. On average, `J.Nano` and Xavier showed a 11% and a 5% increase in DL inference throughput over DMP with a single EdgeTPU accelerator.

More specifically, EdgeTPU cluster could increase 11% (`J.Xavier`) to 20% (`J.Xavier`) of DL inference throughput with multiple accelerators, but at the same time, `J.Nano`'s GPU showed a 12% decreased throughput and `J.Xavier`'s GPU showed 3% decreased throughput. This decreased GPU throughput was mainly due to the resource contention and the resource-constrained nature of edge devices. Because the edge devices do not have sufficient CPU and memory resources (of course limited USB bandwidth) to process many tasks for multiple accelerators, the devices naturally allocated less computing powers. As a result, the DL inference throughput of GPUs was directly impacted (and decreased).

4.6 Discussion and Lesson Learned

Through the study, we saw that, AI-enabling ASICs like EdgeTPU are superior to other edge devices for DL inferencing at the edge. The superiority is in terms of both performance (throughput) and cost-effectiveness (price and power). However, they impose restrictions such as support for only *quantized* models and no support for batched inferencing. Despite these limitations, the throughput provided by these devices far exceed the throughput of even GPU-embedded devices while also utilizing significantly lower resources. In addition, the ability to connect `USB-Accelerators` with TPU to existing edge devices allowed us to evaluate model placement strategies for higher throughput.

Multi-tenancy for DL inferencing was evaluated by employing multi-threading capabilities provided by modern operating systems and programming languages (*e.g.*, python). However, the restriction imposed by python's *Global Interpreter Lock(GIL)* to avoid deadlocks and race-condition by ensuring that only thread can hold the control of the python interpreter at a time, limits true concurrency. While workarounds like using multi-processing instead of multi-threading or using a different python interpreter (*e.g.*, Jython) can resolve the issue, these solutions have their own limitations and overheads. With python being the primary language for DL applications, this issue could pose to be a serious bottleneck for multi-tenant DL applications.

Despite the limitations associated with the programming language's design, we observed significant benefits in running multiple models simultaneously on edge devices. This is an important discovery, as many applications of DL at the edge necessitate the processing of multiple inputs concurrently. In most practical scenarios, these inputs are diverse in nature. For example, in drone-based surveillance, it is crucial to simultaneously process audio and video inputs to achieve comprehensive situational awareness.

In this study, however, we confined our evaluation to a single application domain, specifically image classification. This focused approach allowed us to deeply analyze the performance and benefits within this context. Nevertheless, our findings suggest that expanding the scope of evaluation to include the concurrent execution of a heterogeneous set of DL models across various application domains could yield even more insightful and impactful results.

By investigating how multiple models from different fields, such as audio processing, natural language understanding, and video analysis, perform when executed concurrently on edge devices, we can better understand the broader implications of AI multi-tenancy. Such an expanded evaluation could reveal novel insights into resource management, model interaction, and performance optimization, ultimately

pushing the boundaries of what is achievable with DL inferencing on resource-constrained edge devices. This comprehensive approach would not only validate our current findings but also pave the way for more versatile and robust implementations of deep learning at the edge, catering to a wider array of real-world applications.

4.7 Chapter Summary

This study conducted an in-depth investigation into various system approaches aimed at maximizing DL inference throughput on *resource-constrained* edge devices and EdgeTPU accelerators. It specifically focused on scenarios involving AI multi-tenancy to optimize performance and efficiency.

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

However, we also observed the limitations of the three approaches that will be future research explorations. For batched inferencing, the performance improvements started decreasing once the batch size exceeds a certain threshold (*e.g.*, the number of GPU cores). Besides, due to the limited memory size, there was a limit to the number of input images that could be simultaneously loaded into the memory. For CME and DMP with multi-tenancy, we started getting diminishing returns once the number of concurrently processed models exceeded the number of concurrent threads (or cores) supported by the CPUs. System memory is also turned out to be a bottleneck as we increased the number of concurrent models. Finally, since USB bandwidth drived the rate at which `USB-Accelerators` could process models, multi-tenancy on EdgeTPUs showed performance gain only when fewer data transfers (because of model parameters swapping) were involved. Inherently sequential hardware design, such as shared internal USB hubs, was also a restricting factor when using multiple `USB-Accelerators` simultaneously.

This study confirmed that implementing AI multi-tenancy on edge devices is a promising technique for enhancing the performance of DL tasks. The results demonstrated that AI multi-tenancy can significantly improve the efficiency and effectiveness of DL inferencing on resource-constrained devices. However, to fully realize the potential of this approach, further research is necessary. Specifically, investigating the strategic placement of models to minimize resource contention is crucial. Additionally,

developing robust isolation mechanisms for dynamic control of DL inference throughput can further push the performance boundaries of DL inferencing.

Moreover, given that multi-tenant applications share the same system memory, a thorough analysis of the security of individual applications becomes imperative. Ensuring isolation from other models is essential to maintain the integrity and security of the system. Techniques such as CME and/or DMP must be carefully evaluated to ensure they are suitable for deployment in environments where security is a paramount concern. This study underscores the importance of addressing these challenges to make AI multi-tenancy a viable solution for improving DL task performance on edge devices.

Chapter 5

Toward Low-Cost and Sustainable IoT Systems for Soil Monitoring in Coastal Wetlands

This chapter provides a field-deployable IoT-based soil monitoring system tailored for long-term coastal wetland monitoring. This study mainly introduces two core components of our system: energy management/optimization and the calibration/evaluation of low-cost soil sensors. We develop a novel energy management mechanism powered by multiple predictors, optimizing sensing efficiency and supporting long-term operability, and our contribution incorporates a calibrated set of cost-efficient sensors, ensuring precise soil property estimation. We evaluated our energy management with long-term, real-world weather data. The results show our approach allows more frequent sensing without interruptions, outperforming two common energy management mechanisms and ensuring sustained soil sensor operation in wetlands without a power grid. We also explored the use of low-cost sensors to timely and efficiently estimate complex soil health indicators, particularly bulk density, which is traditionally determined via labor-intensive lab tests.

5.1 Introduction

Coastal wetlands and nearshore environments are important natural carbon storage systems, containing approximately half of the world's blue carbon [11]. They also serve as highly efficient carbon sequestration systems, capable of isolating carbon at rates 55× higher than mature tropical rain forests annually [134, 135]. Beyond this, wetland ecosystems store vast amounts of organic carbon in the soil, often called soil organic carbon (SOC). In particular, while coastal wetlands only cover 5% of the global land area, they account for 20% – 25% of the global SOC stock [134].

Coastal wetland ecosystems face increasing *vulnerabilities* from threats, such as coastal development, invasive species, rising global temperatures, and sea-level rise [12, 13]. Therefore, it is imperative to preserve such wetland areas by continuously monitoring and improving soil health in the coastal wetlands

[14]. However, traditional soil sampling methods are not only costly but also labor-intensive and time-consuming [15]. Specifically, the traditional approach requires pre-processing of soil samples, such as air-drying, sieving (wetland soils get very hard after drying), and pulverizing, which takes even longer for wetland soils. Furthermore, the traditional approach lacks space and time scalability for soil properties measurements. While remote sensing offers a more convenient and cost-efficient alternative, this approach has its own challenges, including weak signal detection and difficulties in accurately observing non-exposed soil and surface water [136, 137].

Due to the advancements in IoT (Internet-of-Things) technologies, various soil sensors have been developed. These sensors are now widely used in various domains, such as agricultural systems [16, 17, 18]. These IoT-based soil sensors play an important role in optimizing water and nutrient application to crops. Given their capability to monitor crop yields and diverse soil parameters, they present a promising approach for soil monitoring. Moreover, soil sensors are cost-effective, offering a more economical way than traditional soil sampling methods and proving more accurate than conventional remote sensing. In coastal wetlands, these sensors can be used to measure various soil/environmental parameters, including soil moisture, electrical conductivity, redox potential, soil salinity, and soil pH [16].

When building an IoT system with soil sensors to monitor soil health in coastal wetlands, there are significant research and engineering challenges. Many of these challenges arise due to the unique environmental dynamics of the coastal wetlands [27]. For example, coastal wetlands often lack stable power supply to support the IoT system's long-term sustainability and operability. To address this challenge, energy harvesting (EH) emerges as a promising approach as it generates electricity from natural resources [28, 29]. Specifically, solar power is a widely used energy source for these EH systems. Various solar panels are compatible with IoT computing boards, such as Raspberry Pi and Arduino, and with sensors. However, due to the weather dynamics, solar power can exhibit non-uniform energy generation patterns [30]. Thus, EH-based IoT systems need intelligent energy management for long-term operations at varying energy harvesting rates.

Additionally, while current sensors can measure multiple soil parameters, there is, to our best knowledge, no IoT sensor capable of directly measuring essential information for evaluating soil health. For example, soil health can be deduced from the variance and quantity of soil organic carbon (SOC) [31, 32], bulk density [33], and microbial biomass [34]. Therefore, these properties need to be estimated from other environmental parameters (*e.g.*, temperature, salinity, and pH) that can be collected by soil sensors. To this end, domain scientists should determine the most appropriate sensors (with respect to cost, accuracy, and data quality) that can measure environmental/soil parameters, and then they should create a (machine learning or empirical) model to infer SOC, bulk density, and/or microbial biomass for determining soil health.

This chapter details our preliminary findings in building a cost-effective, sustainable, field-ready IoT platform to monitor soil health in coastal wetlands. Our insights stem from two distinct research domains: computer science and domain science (*e.g.*, soil science and remote sensing). We first outline the design of our IoT sensing platform, focusing on its overall architecture and its energy scheduler. This scheduler, designed to optimize energy management across various sensors, stands as a key novelty of our platform,

Figure 5.1: **Soil coring activities by our interdisciplinary research team in a wetland near the Savannah area, Georgia, United States.**

ensuring long-term operational capability. Following this, we delve into selecting and evaluating various soil sensors integrated into the platform. Moreover, we introduce our machine-learning model designed to predict one of the soil health indicators *e.g.*, soil bulk density, using data measured by the soil sensors.

5.1.1 Chapter Organization

The rest of this chapter is organized as follows: Section 5.2 describes the background of soil monitoring. Section 5.3 details our IoT-based soil monitoring system, focused on its design, energy management mechanism, and low-cost sensors. Section 5.4 presents evaluation results, emphasizing the performance of the energy management mechanism and our machine learning models for soil bulk density estimation. Section 5.5 summarizes the chapter.

5.2 Background of Soil Monitoring

We now describe the background, advantages, and challenges of soil monitoring approaches, including traditional sampling methods, remote sensing, and soil sensors.

5.2.1 Soil Sampling Method

The traditional and most accurate method of measuring soil health/properties is based on field soil sampling through labor-intensive coring, as shown in Fig. 5.1. Subsequent soil property determinations are made through laboratory testing. While several essential parameters for determining soil health can be directly obtained using traditional soil sampling methods, these techniques are often expensive, labor-intensive, and time-consuming. For example, some laboratory tests, *e.g.*, soil carbon testing, necessitate air-drying the samples. This process can often take months, especially for wetland soils, as per the guidelines from United States Department of Agriculture (USDA) [138].

5.2.2 Remote Sensing Method

Given the high costs and labor-intensive nature of conventional soil sampling, remote sensing has emerged as a more practical and cost-effective alternative. As illustrated in Fig. 5.2, remote sensing utilizes drones or satellites to obtain visible and near-infrared surface reflection images. These image datasets are subsequently leveraged to build either empirical or machine-learning-based models for predicting soil properties. Several studies have found bandwidths in the visible and near-infrared regions of the electromagnetic spectrum that correlate with soil attributes [139, 140, 15]. Yet, achieving a robust signal from the soil via remote sensing remains a challenge. This issue predominantly stems from the weak signal-to-noise ratio (SNR) observed with soils [15]. The challenge is even more complex to address in coastal wetlands. The consistent year-round vegetative cover means that bare soil exposure is infrequent, hindering the precision of remote sensing. Additionally, the regular influx of tidal water can blur soil signatures with those from surface water, further diminishing the clarity of the soil signal.

5.2.3 IoT Soil Sensing Method

The advancement in IoT technologies has catalyzed the development of cost-effective sensing methods, enhancing the monitoring, collection, and storage of environmental data [61]. These IoT systems offer efficient, automated, and non-intrusive monitoring, proving invaluable for traditionally understudied ecosystems like coastal wetlands [141]. Regarding soil sensing, various, low-cost soil sensors are now available, capable of measuring a set of soil parameters such as soil salinity, redox potential, electrical conductivity, pH, and temperature [142, 143, 144, 145]. By integrating these sensors into an IoT system, we can directly deploy the system onto the wetland soil surface. This IoT-based sensing system addresses challenges (*e.g.*, weak SNR) of remote sensing and significantly reduces the time and labor efforts required by traditional sampling methods. Furthermore, this IoT system can be enhanced with advanced deep learning, edge devices, and AI accelerators, such as Raspberry Pi, Jetson Nano, and Google's edge TPUs [146, 147, 66], allowing for *in situ*, lightweight AI analysis of collected soil parameters for supporting timely decision-making.

However, building and deploying an IoT system for soil monitoring presents unique challenges: 1) *determining the most suitable sensors and creating an accurate (empirical or ML) model to assess soil*

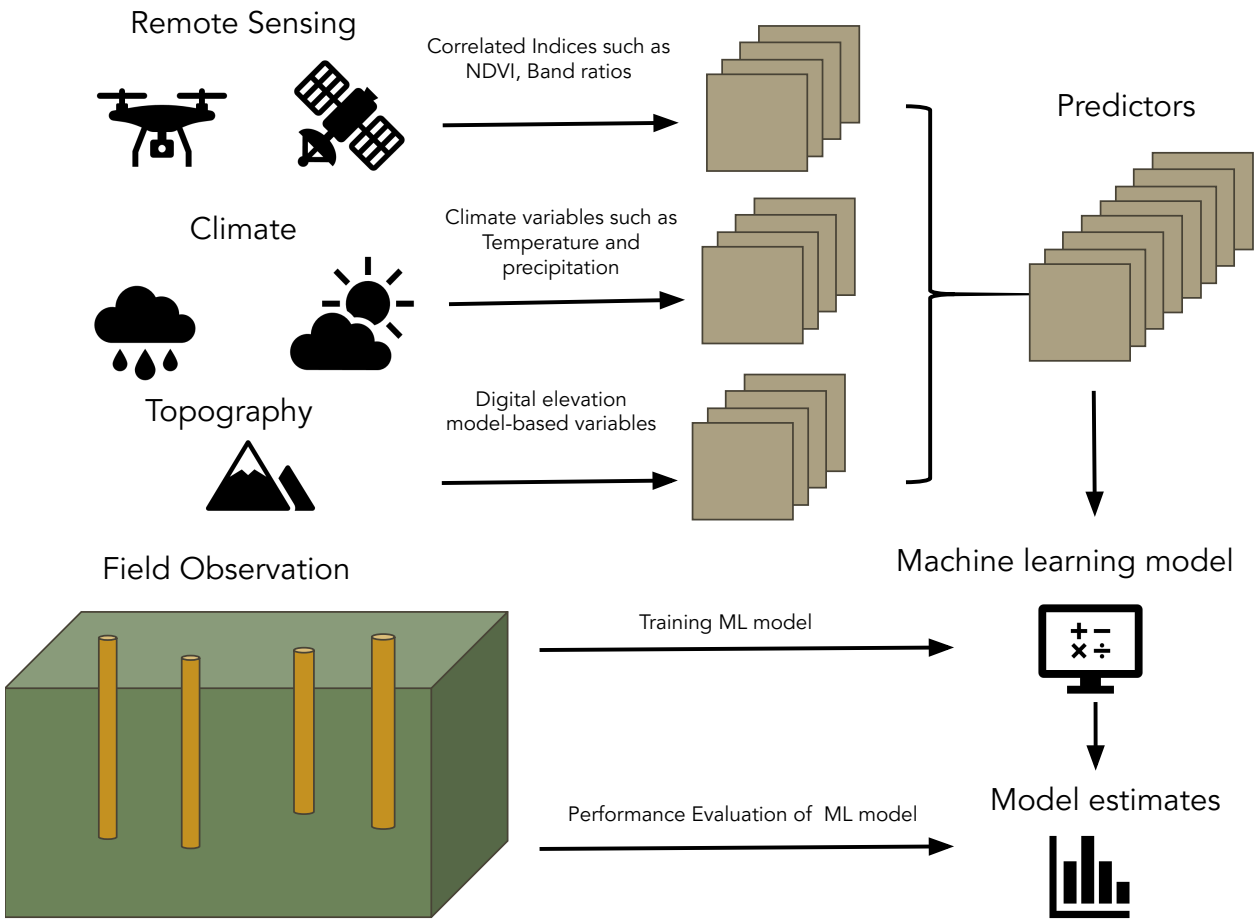


Figure 5.2: **Remote-sensing workflow to monitor and predict soil properties.** *Note that the left-bottom part of this figure also includes coring for soil sampling. Both remote sensing and soil sampling can be used together to enhance the accuracy and validate the results from remote sensing.*

health indicators such as SOC and bulk density and 2) *ensuring a consistent and reliable energy supply and management* for enabling the sustainable system's operations.

For the first challenge, while various soil sensors are available, their accuracy, precision, and reliability vary. More importantly, there is no specific sensor designed to measure essential soil health indicators (SOC and bulk density) directly. These essential metrics need to be estimated from the data captured by the available sensors. In this chapter, we present our empirical study to determine the best-suited soil sensors for collecting base parameter parameters. Additionally, we introduce our ML models designed to accurately estimate bulk density trained on the data collected by low-cost soil sensors.

For the second challenge, energy harvesting (EH) techniques [28, 29], such as solar or wind power, can be a promising approach. However, EH methods can often be unreliable due to weather dynamics such as rain, snow, or cloudy days. Therefore, in this chapter, we introduce a method to predict future solar

Figure 5.3: **IoT soil monitoring platform.**

energy gains without relying on external services (but rather using local environmental parameters that sensors can collect). We also present a novel scheduling strategy to manage energy resources efficiently, ensuring the long-term operability of IoT soil sensing systems.

5.3 IoT-based Soil Monitoring

The goal of this study is to establish an efficient and practical solution to soil monitoring by creating long-term, field-deployable IoT platform. This chapter elaborates on our initial way to achieve this objective. We first discuss **the design of our soil monitoring system** (section 5.3.1), with an emphasis on its

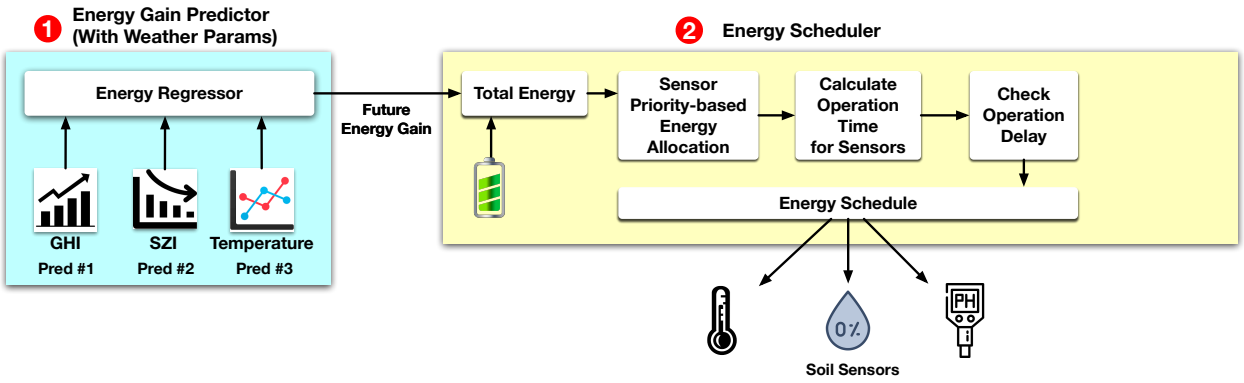


Figure 5.4: **Overview of energy management mechanism for IoT soil monitoring system.**

energy harvesting and management features. Specifically, our energy management algorithm can be readily applied to other EH IoT systems across various domains. We then discuss **our initial performance study of various low-cost soil sensors** (section 5.3.2) with the high-end sensors widely used in domain-specific scientific projects. We also delve into our feasibility study on building ML models. These ML models, trained using data collected from our selected soil sensors, are designed to estimate bulk density, a key indicator of soil health.

5.3.1 IoT Soil Monitoring System

Coastal wetlands have unique environmental characteristics like soft soil, clear water, and dense vegetation, requiring tailored platforms for IoT devices and soil sensors for long-term deployment. Our IoT platform, shown in Fig. 5.3, has a central post with a solar panel on top and a camera (color sensor) below. We place soil sensors connected to a Raspberry Pi at the base for protection against ater exposure and excessive heat. To ensure stability in the soft wetland terrain, our design employs a four-way chassis to prevent sinking or tilting.

Moreover, wetlands are typically located away from conventional power grids, we face significant challenges in ensuring a consistent energy supply. Solar power, as highlighted in §5.2, has become a widely-adopted approach, capturing sunlight and storing it in batteries for ongoing energy needs. Nonetheless, solar power's reliability can often be affected by changing weather conditions. Accurate weather forecasts are essential for energy planning, and many online services provide this information [41]. However, wetlands often lack consistent and reliable networks, making it difficult to access these forecasts.

To ensure a stable power supply, we developed an *energy management mechanism that performs energy gain prediction and scheduling* tailored to our IoT soil monitoring system. Specifically, this mechanism does not rely on external, online weather forecasting services by considering unique environmental challenges (frequent network disconnectivity) in coastal wetlands. Instead, our insight is that we use *in situ* measurable weather parameters that can estimate the total energy gain from solar energy. Specifically, we

focused on three weather parameters, *e.g.*, global horizontal irradiance (GHI) [73], solar zenith angle (SZA) [74], temperature, and their correlation with the strength of sunlight. Eventually, we target to predict the total amount of energy the solar panel in the IoT platform will harvest. Regarding these three weather parameters' correlation with the energy gain, we refer the interested reader to [30] for further details. The workflow of this energy management mechanism is illustrated in Fig. 5.4 and consists of two components 1) *energy gain predictor* using weather parameters and 2) *energy scheduler* for soil sensor management.

Energy gain predictor

To predict the future energy gain from solar energy, we focused on three weather parameters: GHI, SZA, and temperature, which can be collected through *in situ* measurements. Among them, we found that GHI is particularly important for this prediction as it is directly correlated with cloudiness [73]; therefore, we consider this parameter as a key feature of this prediction task. Moreover, GHI has a strong correlation with other SZA and temperature parameters. Therefore, achieving accurate GHI predictions becomes a vital problem for this energy gain prediction. To obtain accurate future GHI values, we compared the prediction accuracy, overhead, and power consumption of three models: Holt-Winters Exponential Smoothing (HWES) [77], NeuralProphet [76], and Long Short-Term Memory (LSTM) [75], testing them on a Raspberry Pi, which is the main computing board installed in our IoT sensing platform. HWES was the fastest but least accurate, while LSTM was the slowest but most accurate. Given GHI's importance, we selected LSTM for GHI prediction. For SZA and temperature predictions, we chose HWES because of its decent accuracy and low power consumption.

We use the predicted values of the three weather parameters as input features for our energy regressor. This regressor is designed to estimate the future energy that the solar panel on our IoT platform will harvest. We evaluated various ML models for this purpose, including random forest (RF) [78], support vector machine (SVM) [77], and linear regression [77]. The RF model was chosen for the energy regressor because it surpassed the others in prediction accuracy.

Energy scheduler

With the estimated future energy gain, the energy scheduler allocates energy to soil sensors. This scheduler has five steps to ensure soil sensors' operations under a contained energy budget.

The first step in this scheduling process involves calculating the total available energy. This includes the energy predicted to be gained during the upcoming time interval and the energy stored in the sensor's battery. Considering these factors, the scheduler obtains a comprehensive overview of the energy resources. With the available energy information, the scheduler allocates energy to each sensor within the system. This allocation is done by the sensors' priorities, which are determined based on their importance and relevance. Sensors with higher priority receive a larger share of the available energy, ensuring that critical sensing tasks are adequately supported. Once the energy allocation is completed, the scheduler calculates the number of sensing operations that each sensor can undertake during the specified time period. This calculation is based on the energy allocated to each sensor and the energy consumption associated with

individual sensors. The next critical aspect of the scheduler's role is the distribution of sensing operations over time. To minimize the overall sensing interval and ensure consistent data collection, the scheduler evenly spreads out the sensing operations across the defined time frame. This even distribution helps maintain a steady pace of data acquisition. However, a potential challenge arises when scheduled sensing operations coincide with low available energy periods, such as at night when solar power generation is minimal. To address this issue, the scheduler incorporates a thoughtful time delay mechanism. This mechanism identifies the earliest feasible time for conducting sensing operations while guaranteeing that the available energy is sufficient to support them.

Soil Property	Vendor/Model	Price ($)	Image
pH	Atlas Scientific-pH	$162.99	
	Grove - pH Sensor	$17.50	
EC (Electrical Conductivity)	Gravity EC Sensor - K10	$79.90	
	Groove EC Sensor Kit	$35.90	
ORP (Oxidation-Reduction Potential)	Vernier - ORP Sensor	$89.00	
	Grove - ORP Sensor Kit	$43.90	
Moisture	Gikfun Capacitive Soil Moisture Sensor	$7.48	
Temperature	DS18B20 Temperature Sensor	$9.99	

Table 5.1: Vendor details and cost of soil sensors deployed on IoT soil sensing system.

5.3.2 Preliminary Performance Evaluation of Soil Sensors

As we discussed in section 5.2, IoT soil sensing can be a promising direction by addressing the limitations of existing methods. As a prerequisite, we should perform rigorous testing on various soil sensors and

select sensors that will be integrated into our IoT soil monitoring system. We first focus on low-cost soil sensors (as shown in Table 5.1), ranging in price from $7.48 to $162.99, capable of measuring five soil properties: pH, electrical conductivity (EC), oxidation-reduction potential (ORP), temperature, and moisture.

We evaluate the feasibility of these sensors, particularly in their ability to infer one key soil health metric: bulk density. And this evaluation compares the bulk density results calculated from measurements using the Hanna HI98196 multiparameter sensor[1]. The Hanna HI9819 sensor is a widely used, high-end handheld device in domain-specific science, is priced at approximately $3,000.

In this evaluation, we used soil samples collected from seven carefully chosen study sites along the Georgia coast, as depicted in Fig. 5.5. These sites are situated on Skidaway Island (2 study sites), Sapelo Island (2 study sites), and near salt marshes in South Newport (1 study site) and Hinesville, GA (2 study sites). Each location represents different water flow patterns from upland to coastal wetlands. Besides recording soil sensor measurements, we collected surface soil samples to determine soil bulk density and soil organic carbon (SOC) content. Our data collection spanned five field expeditions conducted between March 2022 and May 2023.

To infer soil bulk density, we employed various machine learning models, including random forest (RF), support vector machine (SVM), extreme gradient boosting (XGBoost) [148], and neural networks (NN) [149]. We compared the estimated bulk density with soil properties collected by both the low-cost soil sensors and the baseline Hanna sensor, alongside lab test results. Our performance metrics include mean absolute error (MAE) and the coefficient of determination (R^2). Detailed evaluation results are discussed in section 5.4.2 and section 5.4.3.

5.4 Evaluation Results

This chapter presents the evaluation results of our IoT soil monitoring system. We first discuss the performance of the energy management mechanism for the system (section 5.4.1). We then delve into the calibration and lab test results of the low-cost soil sensors (section 5.4.2). Finally, we report our initial evaluations regarding the estimation of soil bulk density through real-world deployment of the IoT soil monitoring system (section 5.4.3).

5.4.1 Evaluation Results of Energy Management for IoT Soil Monitong System

We conducted an evaluation to assess both the **energy gain predictor** and the **energy scheduling component** (the first and second component shown in Fig. 5.4). The accuracy of the energy gain predictor is particulary important as it establishes the basis for predicting future energy gains, which are essential for scheduling operations. To evaluate this accuracy, we measured the root mean square error (RMSE) and

[1] `https://www.hannainst.com/hi98196-multiparameter-ph-orp-do-temperature-waterproof-meter.html`

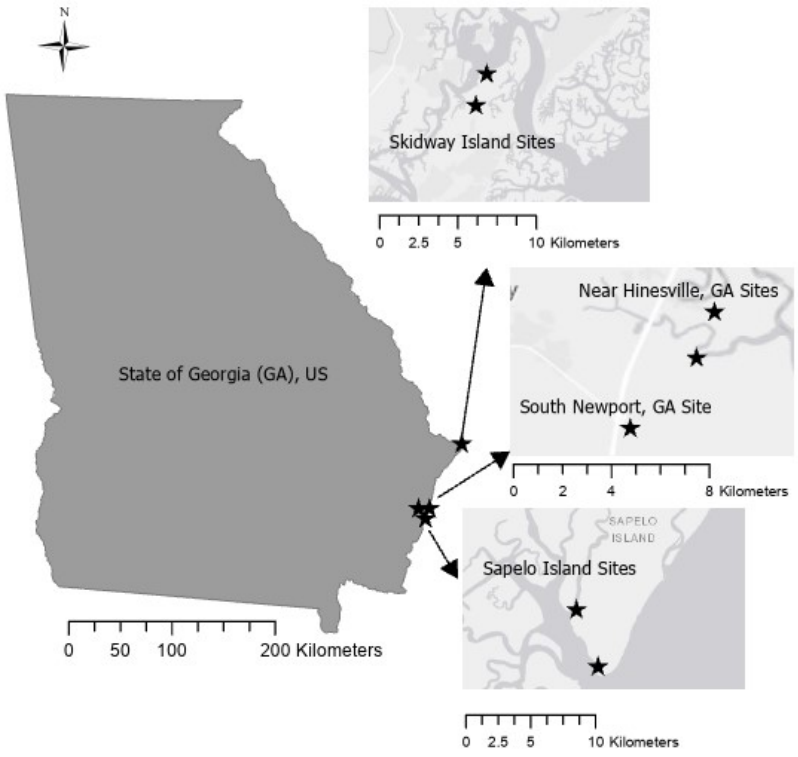


Figure 5.5: **Study sites along the GA coast. In the north Skidway Island, and Near Hinesville, South Newport, and Sapelo Island sites in the mid-Georgia coast.**

over-prediction rate of the DC power estimator, investigating how different parameters affect prediction precision.

Additionally, we evaluated the performance of our energy scheduler through simulations that reflected real-world scenarios for EH environmental sensors. This evaluation involved observing changes in battery charging levels, variations in sensing operation frequencies, and adjustments in sensing intervals for various sensor configurations. Furthermore, we compared the performance of our energy scheduling approach with two benchmark strategies: static and adaptive energy schedulers.

Evaluation dataset

We combined two openly available datasets [72, 71] from NREL (National Renewable Energy Laboratory) to create new datasets for our evaluation. These datasets cover a two-year period from 2017 to 2018 and include various meteorological parameters like GHI, temperature, and DC power gain data captured by solar panels installed in Denver, Colorado, USA. The baseline schedulers used in this evaluation are a static scheduler (es-stat) and an adaptive scheduler (es-adap). 'es-stat' employs a fixed sensing interval for sensing operations, while 'es-adap' adjusts the sensing interval based on the battery level.

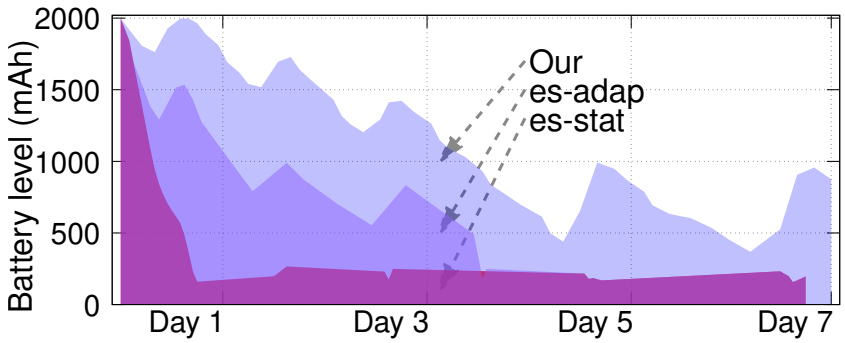


Figure 5.6: **Battery level changes in three energy schedulers.**

We evaluated the energy management mechanism's performance using a trace-based simulator with real-world power consumption data from soil sensors and the energy gain predictor. These experiments took place over five random weeks in 2017 and 2018, focusing on changes in battery levels and sensing frequency.

Evaluation results

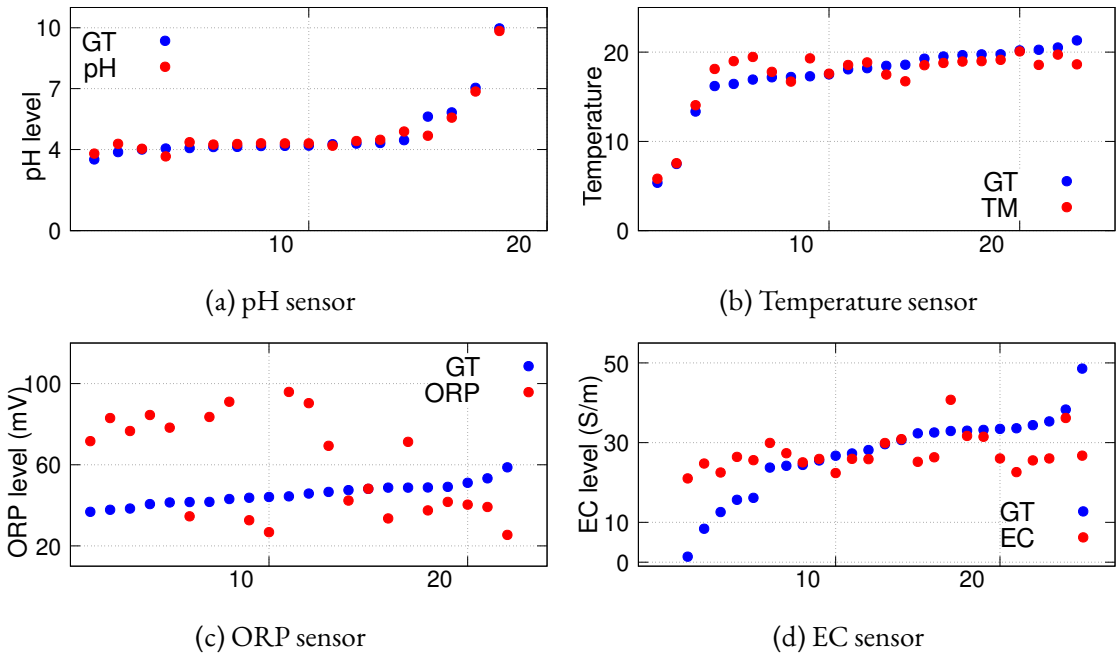
Fig. 5.6 reports the results for one of those weeks due to space constraints, but similar outcomes were observed in the other weeks. This figure demonstrates that our scheduler effectively managed battery levels, allowing for more frequent sensor operations without completely draining the battery. The fluctuations in battery level were primarily influenced by changes in weather conditions.

In contrast, the baseline approaches faced challenges in maintaining EH sensor operations. For instance, the 'es-stat' scheduler experienced complete battery depletion on the first day of the week, resulting in minimal sensing operations thereafter. Although 'es-adap' performed better than 'es-stat,' it also encountered complete battery depletion on specific days, limiting its capacity to support future operations due to insufficient stored energy.

These evaluation results confirm that ***our energy management mechanism enabled more frequent and reliable sensing operations, ultimately enhancing the overall quality of EH sensor data***. For comprehensive evaluation results of our energy management mechanism, please refer to [30].

5.4.2 Lab Test and Calibration Results of Low-Cost Soil Sensors

We evaluated the performance of the low-cost sensors (listed in Table 5.1) intended for integration into our IoT soil monitoring system through laboratory testing. Fig. 5.7 shows the soil property measurement results obtained from the low-cost soil sensors compared to the Hanna sensor, represented as 'GT' in the figure.



(a) pH sensor

(b) Temperature sensor

(c) ORP sensor

(d) EC sensor

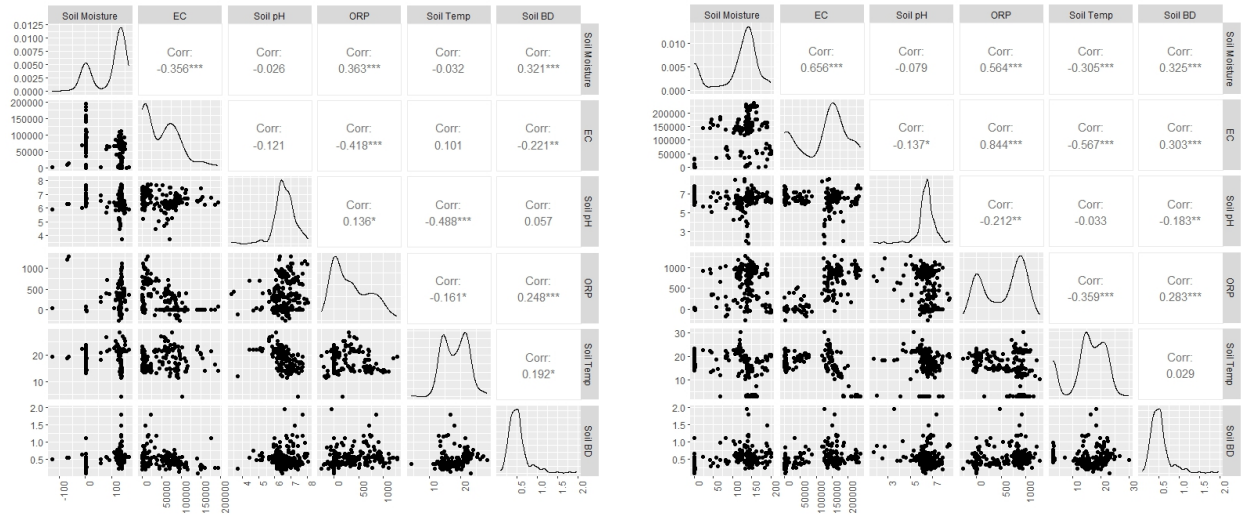
Figure 5.7: **Low-cost soil sensor calibration based on test solutions and Hanna sensor.** *GT: ground truth (Hanna sensor); pH: pH sensor; TM: Temperature sensor; ORP: Oxidation-Reduction potential sensor; EC: Electrical conductivity sensors. All x-axes are the number of data samples.*

During the pH and temperature tests, the pH sensor (Fig. 5.7a) and temperature sensor (Fig. 5.7b) closely matched the readings of the Hanna sensor when testing pH solutions and known temperatures. However, we also obeserved that there were variations in the measurements of the EC (Electrical Conductivity) sensor (Fig. 5.7d) and ORP (Oxidation-Reduction Potential) sensor (Fig. 5.7c) between the low-cost sensors and the Hanna sensors. This discrepancy might be attributed to their sensitivity to temperature changes since both EC and ORP readings rely on temperature sensor data. Errors from temperature sensors in both the low-cost and Hanna sensors could propagate to EC and ORP measurements. Nevertheless, the EC and ORP sensors generally followed a similar trend to that of the Hanna sensor.

For further testing, we collected *in situ* soil measurements to analyze the overall trends in soil properties along the salinity gradient and to estimate soil bulk density using data from our IoT soil sensors.

5.4.3 In Situ Test Results of IoT Soil Sensors

We conducted a preliminary analysis of *in situ* soil properties collected from IoT soil sensors, as shown in Fig. 5.8. These figures report the distribution of soil properties along the diagonals, scatter plots depicting relationships between soil properties (bottom-left of diagonal), and indicate the strength and significance of the linear correlations between these soil properties. Please note that all soil properties, except for



(a) Evaluation Set #1

(b) Evaluation Set #2

Figure 5.8: **Linear correlation matrix for soil properties measured from IoT soil sensors** *The diagonal represents the distribution of soil properties; the top-right of the diagonal shows linear correlation strength and significance; the bottom-left of the diagonal shows a scatter plot between the soil properties.*

soil bulk density, were gathered from IoT soil sensors. Soil bulk density was determined based on lab measurements.

Soil pH and soil temperature values showed similar distributions for our evaluation sets. However, soil properties, such as soil moisture, EC, and ORP, had different distributions for both sets. There are some expected correlations between the soil properties. For example, Soil ORP and EC are highly correlated to each other. Other studies have found a correlation between soil ORP and EC [150]. However, the directionality of correlation was different for experiment set #1 (Fig. 5.8a) and set #2 (Fig. 5.8b). This could be because of sensor-to-sensor fluctuations. All collected soil properties had a strong and statistically significant correlation with soil bulk density. This confirms our hypothesis that soil properties collected from IoT soil sensors can be used for predicting complex soil properties (bulk density).

Next, we perform the evaluation of various ML models to estimate soil bulk density with two datasets (evaluation set #1 and #2) collected by IoT soil sensors and a dataset from the baseline Hanna sensor. In this evaluation, we found that our ML models' outcomes were influenced by the choice of training and testing data divisions. This might be due to higher unpredictability in the soil bulk density estimates. To address this, we also included average statistics along with a 90% confidence interval (CI) for MAE and R^2, derived from a Leave-One-Out Cross-Validation. Table 5.2 reports the MAE and R^2 for models trained using data from IoT soil sensors (evaluation set #1 and #2) versus the Hanna sensor, which serves as our baseline.

Table 5.2: **Performance of ML models for BD estimation based IoT soil sensors (Evaluation Set #1 and #2) and Hanna sensor.**

	Evaluation Set #1			
	Average MAEs (g/cm3)	CI of MAEs (g/cm3)	Average R^2 (%)	CI of R^2 (%)
RF	0.30	0.28 – 0.31	51.5	49 – 54
SVM	0.24	0.23 – 0.25	59.9	58 – 61
XGBoost	0.40	0.35 – 0.44	46.8	42 – 51
NN	0.73	0.45 – 1.2	22.2	1 – 43
	Evaluation Set #2			
	Average MAEs (g/cm3)	CI of MAEs (g/cm3)	Average R^2 (%)	CI of R^2 (%)
RF	0.34	0.33 – 0.35	67.1	64 – 69
SVM	0.32	0.31 – 0.33	57.8	57 – 59
XGBoost	0.45	0.41 – 0.50	45.2	38 – 54
NN	1.09	0.59 – 1.9	39.7	12 – 61
	Hanna Sensor (Baseline)			
	Average MAEs (g/cm3)	CI of MAEs (g/cm3)	Average R^2 (%)	CI of R^2 (%)
RF	0.42	0.39 – 0.44	60.3	56 – 64
SVM	0.47	0.46 – 0.48	37.3	36 – 38
XGBoost	0.83	0.71 – 0.97	41.3	31 – 54
NN	1.23	0.75 – 1.99	12.0	0 – 35

From our results, both RF and SVM outperformed XGBoost and NN for our dataset derived from both IoT sensors and the Hanna sensor. The better performance of RF and SVM aligns with other research findings for soil property estimation [140, 151, 152]. Moreover, models trained on both IoT-based evaluation sets had lower error rates than the Hanna sensor-based model. This difference might be due to their measurement methods: IoT sensors measure directly from the surface soil, while the Hanna sensor only reads from soil pore water.

Comparing SVM and RF, SVM consistently outperformed with the lowest MAE for both sets. Moreover, SVM showed greater stability, having the narrowest 90% CI width for MAE in both sets. Interestingly, models trained on the evaluation set #1 had slightly better accuracy in error estimates and R^2. Among the IoT soil sensors, all the models trained on the evaluation set #1 performed slightly better than the models on the evaluation set #2. This difference is highlighted by distinct CIs for both MAE and R^2. However, please note that, due to the sensitivity of our ML models to the specific training-testing data splits, further testing is required to make more reliable conclusions about sensor set performance. This sensitivity prompted our choice of leave-one-out cross-validation to reduce the bias of dataset splits, but some bias inevitably remained. Moreover, the potential impact of extended use on the IoT soil sensor performance remains an open question.

5.5 Chapter Summary

Coastal wetlands are invaluable carbon repositories. However, monitoring and collecting data from such regions has historically been labor-intensive and time-consuming. IoT-based systems present a promising solution, but two primary challenges arise: 1) ensuring a continuous energy supply for long-term IoT operation and 2) selecting precise soil sensors to accurately determine essential metrics like soil bulk density and organic carbon.

This study addressed the first challenge by developing a novel energy management mechanism designed explicitly for solar-powered IoT systems. This mechanism does not rely on accessing online weather services. Instead, it employs LSTM and statistical methods to predict changes in critical weather parameters, such as GHI, SZA, and temperature. These predictions enable accurate DC power gain forecasts. With these predictions, our energy management mechanism can dynamically allocate power to the sensors based on their importance, thereby enabling the long-term sensing operation of the system.

For the second challenge, we thoroughly evaluated various low-cost soil sensors. The objective was to evaluate their potential in estimating complex soil health indicators, specifically bulk density, a metric usually determined through time-intensive lab tests. By harnessing field metrics like ORP, EC, temperature, and pH, we tested the performance of various ML models to estimate this bulk density. Our preliminary results confirm the need for extended testing to ensure accuracy. However, promisingly, our ML models trained on data from these low-cost sensors outperformed those based on the high-end Hanna sensors.

In the future, we aim to deploy our IoT soil system for long-term soil monitoring and enhance our energy management mechanism by considering both sensor data quality and *in situ* analytics. Additionally, we will refine and extend our ML models to achieve more accurate inferences of key soil metrics, including

both bulk density and SOC. Ultimately, our goal is to provide a non-invasive and cost-effective solution for monitoring critical, yet challenging-to-study, ecosystems like coastal wetlands.

Chapter 6

Future Directions and Conclusions

This chapter delves into potential future research directions and presents the conclusions derived from this dissertation.

6.1 Future Research Considerations

This dissertation sets the stage for real-world deployment and comprehensive validation of IoT-edge systems. There are two future directions for this work.

The first future direction involves exploring the extensive impact of environmental factors on the performance of edge devices when deploying AI applications. Environmental factors such as temperature, humidity, dust, air pressure, X-ray radiation, and exposure to various weather elements can significantly influence the reliability and efficiency of hardware used in edge computing. For instance, extreme temperatures might induce thermal throttling in processors, directly reducing the operational speed of AI tasks, while high humidity levels can lead to corrosion or electrical shorts in circuitry, causing system malfunctions. Additionally, the presence of dust and particulate matter can obstruct cooling mechanisms, further degrading device performance. Understanding how these and other environmental factors, like salt air in coastal areas or vibration in industrial settings, affect edge device performance could lead to the development of more resilient systems. These systems would not only maintain high efficiency and reliability under diverse environmental stresses but also adapt in real-time to optimize performance and longevity. This research would be important in ensuring that edge computing devices can robustly support AI applications across a wide range of challenging and variable environments.

The second future direction for enhancing edge computing performance includes two approaches that aim to expand the computational capabilities of edge devices for processing heavy deep learning models. The first approach involves creating and optimizing clusters of edge devices. By interconnecting multiple edge devices into an edge cluster, computational tasks can be distributed across the network, significantly increasing the overall processing power available for complex AI computations. This distributed pro-

cessing framework not only improves the computational throughput but also enhances system resilience and fault tolerance, making it particularly valuable in scenarios where quality of service is critical. Moreover, this setup can dynamically allocate resources based on workload demands, ensuring optimal use of computational resources.

The second approach focuses on the optimization of various advanced model compression techniques to reduce the size and computational requirements of deep learning models. Techniques such as quantization, which reduces the precision of the numbers used in computations; pruning, which eliminates unnecessary information in the neural network; and knowledge distillation, where a smaller model is trained to replicate the performance of a larger one, are critical in making it feasible to run large AI models on resource-constrained edge devices. These methods help in creating a balance between performance and computational efficiency, minimizing memory usage and enhancing the speed of model inference.

Incorporating these techniques can significantly transform edge computing into a robust platform capable of executing more complex AI tasks directly at the data source. This not only reduces the latency associated with data transmission to centralized cloud servers but also improves data privacy by processing sensitive information locally. As edge computing continues to evolve, these advanced approaches will enable a wide range of applications, from autonomous vehicles and smart cities to IoT in industrial settings, to leverage the full potential of AI without the traditional constraints imposed by limited edge resources. These future directions will pave the way for broader and more effective deployment of AI capabilities in edge computing environments, pushing the boundaries of what these technologies can achieve.

6.2 Conclusions

This dissertation consolidates findings across three areas: energy management for IoT-edge systems, optimization of DL performance on edge computing devices, and the deployment of IoT-edge systems for environmental sensing.

An advanced energy scheduler is designed for environments lacking access to weather information networks. Utilizing a combination of deep learning and statistical time-series methods, DynaES accurately predicts energy gains from solar inputs and dynamically allocates energy based on sensor priority, enhancing operational frequency and extending battery life. The integration of these forecasts allows for efficient scheduling of energy distributions, ensuring that the IoT-edge systems can operate longer and more reliably without needing constant recharging.

In edge computing, this work explored the enhancement of DL inference throughput on resource-constrained devices through innovative system approaches such as AI multi-tenancy (CME and DMP) and batching. These strategies have shown substantial improvements in handling parallel processing tasks on GPUs and EdgeTPUs. However, the research also highlighted the limits of scaling such methods, including memory bottlenecks and inefficiencies in USB bandwidth, which restrict the performance gains from multi-tenancy.

For the environmental monitoring of coastal wetlands, this dissertation evaluated multiple low-cost soil sensors to determine key soil health indicators. This approach addressed the challenge of the accuracy

of soil metrics like bulk density, traditionally assessed through labor-intensive lab tests. The study's findings advocate for further testing and refinement of machine learning models to enhance the predictive accuracy of soil health indicators based on data from these sensors.

This dissertation represents a pioneering effort in optimizing in situ IoT-edge sensing systems, focusing on enhancing their energy efficiency, deployment flexibility, and AI performance. By addressing key challenges in sensing systems, this research holds immense potential to improve sustainability, performance, and deployability in diverse environments, ranging from the challenging conditions of wetlands and forests to the controlled settings of farms and greenhouses.

References

[1] Yuanhao Cui et al. "Integrating sensing and communications for ubiquitous IoT: Applications, trends, and challenges". In: *IEEE Network* 35.5 (2021), pp. 158–167.

[2] Silvia Liberata Ullo and Ganesh Ram Sinha. "Advances in smart environment monitoring systems using IoT and sensors". In: *Sensors* 20.11 (2020), p. 3113.

[3] Mohammad Ghiasi et al. "Resiliency/cost-based optimal design of distribution network to maintain power system stability against physical attacks: A practical study case". In: *IEEE Access* 9 (2021), pp. 43862–43875.

[4] Sandro Nižetić et al. "Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future". In: *Journal of cleaner production* 274 (2020), p. 122877.

[5] MG Sarwar Murshed et al. "Machine learning at the network edge: A survey". In: *ACM Computing Surveys (CSUR)* 54.8 (2021), pp. 1–37.

[6] Haozhao Wang et al. "A comprehensive survey on training acceleration for large machine learning models in IoT". In: *IEEE Internet of Things Journal* 9.2 (2021), pp. 939–963.

[7] Jiasi Chen and Xukan Ran. "Deep learning with edge computing: A review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[8] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. "Edge machine learning for ai-enabled iot devices: A review". In: *Sensors* 20.9 (2020), p. 2533.

[9] Rong Yu and Peichun Li. "Toward resource-efficient federated learning in mobile edge computing". In: *IEEE Network* 35.1 (2021), pp. 148–155.

[10] Sherali Zeadally et al. "Design architectures for energy harvesting in the Internet of Things". In: *Renewable and Sustainable Energy Reviews* 128 (2020), p. 109901.

[11] ML Kirwan and LK Blum. "Enhanced decomposition offsets enhanced productivity and soil carbon accumulation in coastal wetlands responding to climate change". In: *Biogeosciences* 8.4 (2011), pp. 987–993.

[12] Léa Lorrain-Soligon et al. "Long-term trends of salinity in coastal wetlands: Effects of climate, extreme weather events, and sea water level". In: *Environmental Research* 237 (2023).

[13] Shuxin Luo and Ting Fong May Chui. "Sea-level rise predicted to reduce exotic mangrove distribution and biomass in coastal wetlands in southern China". In: *Journal of Hydrology* 612 (2022).

[14] Ren-Min Yang and Wen-Wen Guo. "Modelling of soil organic carbon and bulk density in invaded coastal wetlands using Sentinel-1 imagery". In: *International Journal of Applied Earth Observation and Geoinformation* 82 (2019), p. 101906.

[15] Rajneesh Sharma et al. "Remote Sensing of Surface and Subsurface Soil Organic Carbon in Tidal Wetlands: A Review and Ideas for Future Research". In: *Remote Sensing* 14.12 (2022), p. 2940.

[16] Heyu Yin et al. "Soil sensors and plant wearables for smart and precision agriculture". In: *Advanced Materials* 33.20 (2021), p. 2007764.

[17] Bhuwan Kashyap and Ratnesh Kumar. "Sensing Methodologies in Agriculture for Soil Moisture and Nutrient Monitoring". In: *IEEE Access* 9 (2021), pp. 14095–14121.

[18] Raphael A. Viscarra Rossel and Johan Bouma. "Soil sensing: A new paradigm for agriculture". In: *Agricultural Systems* 148 (2016), pp. 71–74.

[19] Mohammadreza Gholikhani et al. "A critical review of roadway energy harvesting technologies". In: *Applied Energy* 261 (2020), p. 114388.

[20] Mahmuda Khatun Mishu et al. "Prospective efficient ambient energy harvesting sources for IoT-equipped sensor applications". In: *Electronics* 9.9 (2020), p. 1345.

[21] Bharat Bhushan et al. "Unification of Blockchain and Internet of Things (BIoT): requirements, working model, challenges and future directions". In: *Wireless Networks* 27 (2021), pp. 55–90.

[22] Keyan Cao et al. "An overview on edge computing research". In: *IEEE access* 8 (2020), pp. 85714–85728.

[23] Qianyu Guo et al. "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 810–822.

[24] Rongjie Yi et al. "Boosting DNN Cold Inference on Edge Devices". In: *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. MobiSys '23. Helsinki, Finland: Association for Computing Machinery, 2023, pp. 516–529. ISBN: 9798400701108. DOI: 10.1145/3581791.3596842. URL: https://doi.org/10.1145/3581791.3596842.

[25] Qipeng Wang et al. "Melon: Breaking the memory wall for resource-efficient on-device machine learning". In: *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 2022, pp. 450–463.

[26] In Gim and JeongGil Ko. "Memory-efficient DNN training on mobile devices". In: *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 2022, pp. 464–476.

[27] Abdallah Yussuf Ali Abdelmajeed et al. "Cloud-Based Remote Sensing for Wetland Monitoring—A Review". In: *Remote Sensing* 15.6 (2023), p. 1660.

[28] Naomi Stricker et al. "Joint Energy Management for Distributed Energy Harvesting Systems". In: *ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 2021.

[29] Kai Geissdoerfer et al. "Getting more out of energy-harvesting systems: energy management under time-varying utility with PreAct". In: *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2019.

[30] Jianwei Hao et al. "DynaES: Dynamic Energy Scheduling for Energy Harvesting Environmental Sensors". In: *IEEE International Performance Computing and Communications Conference (IPCCC)*. 2023.

[31] DE Stott. "Recommended soil health indicators and associated laboratory procedures". In: *Soil Health Technical Note* 450-03 (2019).

[32] Márcio R Nunes et al. "The soil health assessment protocol and evaluation applied to soil organic carbon". In: *Soil Science Society of America Journal* 85.4 (2021), pp. 1196–1213.

[33] MK Shukla, R Lal, and M Ebinger. "Soil quality indicators for reclaimed minesoils in southeastern Ohio". In: *Soil Science* 169.2 (2004), pp. 133–142.

[34] Elke Jurandy Bran Nogueira Cardoso et al. "Soil health: looking for suitable indicators. What should be considered to assess the effects of use and management on soil health?" In: *Scientia Agricola* 70 (2013), pp. 274–289.

[35] Alessandro Cammarano, Chiara Petrioli, and Dora Spenza. "Pro-Energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks". In: *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*. 2012.

[36] Aman Kansal et al. "Power management in energy harvesting sensor networks". In: *ACM Transactions on Embedded Computing Systems (TECS)* 6.4 (2007), 32–es.

[37] Mawloud Guermoui et al. "Support vector regression methodology for estimating global solar radiation in Algeria". In: *The European Physical Journal Plus* 133 (2018), pp. 1–9.

[38] Seyed Mohammad Jafar Jalali et al. "Automated Deep CNN-LSTM Architecture Design for Solar Irradiance Forecasting". In: *IEEE Trans. on Systems, Man, and Cybernetics: Systems* 52.1 (2022), pp. 54–65.

[39] Giacomo Capizzi, Christian Napoli, and Francesco Bonanno. "Innovative Second-Generation Wavelets Construction With Recurrent Neural Networks for Solar Radiation Forecasting". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.11 (2012), pp. 1805–1815.

[40] Navin Sharma et al. "Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems". In: *IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 2010.

[41] NOAA. *National Weather Services*. `www.weather.gov`. 2023.

[42] Antonio Caruso et al. "A Dynamic Programming Algorithm for High-Level Task Scheduling in Energy Harvesting IoT". In: *IEEE Internet of Things Journal* 5 (2018), pp. 2234–2248.

[43] Pierpaolo Loreti, Lorenzo Bracciale, and Giuseppe Bianchi. "StableSENS: Sampling time decision algorithm for IoT energy harvesting devices". In: *IEEE Internet of Things Journal* 6.6 (2019), pp. 9908–9918.

[44] Fan Yang et al. "AsTAR: Sustainable Energy Harvesting for the Internet of Things through Adaptive Task Scheduling". In: *ACM Transactions on Sensor Networks* 18.1 (2021).

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

[46] 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.

[47] Colin Samplawski et al. "Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators". In: *Int'l Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AUChallengeIoT)*. New York, USA, Nov., 2019.

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

[49] Ramyad Hadidi et al. "Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices". In: *IEEE International Symposium on Workload Characterization (IISWC)*. Orlando, FL, USA: IEEE, November, 2019, pp. 35–48.

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

[51] 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.

[52] *Intel Neural Compute Stick*. https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. [ONLINE]. 2022.

[53] Bhuwan Kashyap and Ratnesh Kumar. "Sensing methodologies in agriculture for soil moisture and nutrient monitoring". In: *IEEE Access* 9 (2021), pp. 14095–14121.

[54] Mahammad Shareef Mekala and P Viswanathan. "A Survey: Smart agriculture IoT with cloud computing". In: *2017 international conference on microelectronic devices, circuits and systems (ICMDCS)*. IEEE. 2017, pp. 1–7.

[55] HR Bogena et al. "Potential of wireless sensor networks for measuring soil water content variability". In: *Vadose Zone Journal* 9.4 (2010), pp. 1002–1013.

[56] HR Bogena et al. "Hybrid wireless underground sensor networks: Quantification of signal attenuation in soil". In: *Vadose Zone Journal* 8.3 (2009), pp. 755–761.

[57] Juan Vera et al. "Towards irrigation automation based on dielectric soil sensors". In: *The Journal of Horticultural Science and Biotechnology* 96.6 (2021), pp. 696–707.

[58] Hugo de Moura Campos et al. "Low-cost open-source platform for irrigation automation". In: *Computers and Electronics in Agriculture* 190 (2021), p. 106481.

[59] Hugo de Moura Campos et al. "Low-cost open-source platform for irrigation automation". In: *Computers and Electronics in Agriculture* 190 (2021), p. 106481.

[60] Maria Knadel et al. "Soil organic carbon and particle sizes mapping using vis–NIR, EC and temperature mobile sensor platform". In: *Computers and Electronics in Agriculture* 114 (2015), pp. 134–144.

[61] Ali Saffari et al. "Smart Pallets: Toward Self-Powered Pallet-Level Environmental Sensors for Food Supply Chains". In: *ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 2022.

[62] Long Liu et al. "Promoting Smart Cities Into The 5G Era With Multi-field Internet of Things (IoT) Applications Powered With Advanced Mechanical Energy Harvesters". In: *Nano Energy* 88 (2021).

[63] Madeleine I. G. Daepp and et al. "Eclipse: An End-to-End Platform for Low-Cost, Hyperlocal Environmental Sensing in Cities". In: *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2022.

[64] Lea Dujic Rodic et al. "Machine Learning and Soil Humidity Sensing: Signal Strength Approach". In: *ACM Transactions on Internet Technology* 22.2 (2022), 39:1–39:21.

[65] Ju Wang et al. "Soil moisture sensing with commodity RFID systems". In: *ACM International Conference on Mobile Systems Applications, and Services (MobiSys)*. 2020.

[66] Andy Rosales Elias et al. "Where's The Bear?: Automating Wildlife Image Processing Using IoT and Edge Cloud Systems". In: *IEEE/ACM International Conf. on Internet-of-Things Design and Implementation (IoTDI)*. 2017.

[67] *Open Weather*. `openweathermap.org`. 2023.

[68] Xiaojie Wang et al. "Future Communications and Energy Management in the Internet of Vehicles: Toward Intelligent Energy-Harvesting". In: *IEEE Wireless Communications* 26.6 (2019), pp. 87–93.

[69] Deniz Gunduz et al. "Designing Intelligent Energy Harvesting Communication Systems". In: *IEEE communications magazine* 52.1 (2014), pp. 210–216.

[70] Vikrant Bhatnagar and Philip Owende. "Energy Harvesting for Assistive and Mobile Applications". In: *Energy Science & Eng.* 3.3 (2015), pp. 153–173.

[71] National Renewable Energy Laboratory. *NSRDB: National Solar Radiation Database*. nsrdb.nrel.gov. 2023.

[72] Chris Deline and et al. *Photovoltaic Data Acquisition Public Datasets*. www.osti.gov/dataexplorer/biblio/dataset/1846021. 2021.

[73] P.I. Raptis et al. "Measurements and model simulations of solar radiation at tilted planes, towards the maximization of energy capture". In: *Energy* 130 (2017), pp. 570–580.

[74] Mark Z. Jacobson. "Fundamentals of Atmospheric Modeling". In: (2005).

[75] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[76] Oskar Triebe et al. "NeuralProphet: Explainable Forecasting at Scale". In: *CoRR* abs/2111.15397 (2021).

[77] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The Element of Statistical Learning: Data Mining, Inference, and Prediction". In: (2011).

[78] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[79] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *9th Python in Science Conference (SciPy)*. 2010.

[80] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[81] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

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

[83] PiSupply. *PiJuice HAT*. https://github.com/PiSupply/PiJuice. 2023.

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

[85] 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.

[86] 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)*. Santa Clara, CA, USA: USENIX Association, July, 2015.

[87] Yiping Kang et al. "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge". In: *Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Xi'an, China, April, 2017.

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

[89] 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.

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

[91] 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.

[92] Wazir Zada Khan et al. “Edge computing: A survey”. In: *Future Generation Computing Systems* 97 (2019), pp. 219–235.

[93] *Raspberry Pi 4*. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [ONLINE]. 2022.

[94] Koustabh Dolui and Soumya Kanti Datta. “Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing”. In: *Global Internet of Things Summit (GIoTS)*. Geneva, Switzerland: IEEE, June, 2017, pp. 1–6.

[95] Kim M. Hazelwood and et al. “Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective”. In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Vienna, Austria, February, 2018, pp. 620–629.

[96] Carole-Jean Wu and et al. “Machine Learning at Facebook: Understanding Inference at the Edge”. In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Washington DC, USA, February, 2019, pp. 331–344.

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

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

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

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

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

[102] 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.

[103] 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.

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

[105] Yu Cheng et al. “A Survey of Model Compression and Acceleration for Deep Neural Networks”. In: *CoRR* (2017). eprint: 1710.09282.

[106] Yihui He et al. “AMC: AutoML for Model Compression and Acceleration on Mobile Devices”. In: *15th European Conference Computer Vision (ECCV)*. Vol. 11211. Lecture Notes in Computer Science. Munich, Germany: Springer, September, 2018, pp. 815–832.

[107] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”. In: *4th International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, May, 2016.

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

[109] 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. URL: http://arxiv.org/abs/1602.07360.

[110] 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.

[111] 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, June, 2018.

[112] Chuang Hu et al. “Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge”. In: *IEEE Conference on Computer Communications (INFOCOM)*. Paris, France: IEEE, April, 2019, pp. 1423–1431.

[113] Thaha Mohammed et al. “Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading”. In: *IEEE Conference on Computer Communications (INFOCOM)*. Toronto, ON, Canada: IEEE, July, 2020, pp. 854–863.

[114] 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)*. Phoenix, AZ, USA, June, 2019.

[115] Xiaotang Jiang et al. “MNN: A Universal and Efficient Inference Engine”. In: *3rd Conference on Machine Learning and Systems (MLSys)*. Austin, TX, USA, March, 2020.

[116] *Jetson Nano | Nvidia Developer*. https://developer.nvidia.com/embedded/jetson-nano. [ONLINE]. 2022.

[117] *Nvidia Jetson TX2*. https://developer.nvidia.com/embedded/jetson-tx2. [ONLINE]. 2022.

[118] *NVIDIA Jetson Xavier NX*. https://developer.nvidia.com/embedded/jetson-xavier-nx. [ONLINE]. 2022.

[119] *Coral USB Accelerator Datasheet*. https://coral.ai/docs/accelerator/datasheet/. [ONLINE]. 2022.

[120] Haotian Zhang et al. "Eye in the Sky: Drone-Based Object Tracking and 3D Localization". In: *ACM International Conference on Multimedia (MM)*. Nice, France, October, 2019.

[121] Zhiting Zhu et al. "Understanding the security of discrete GPUs". In: *The General Purpose GPUs*. 2017, pp. 1–11.

[122] 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.

[123] *INA219 – 26V, 12-bit, i2c output current/voltage/power monitor*. https://www.ti.com/product/INA219. [ONLINE]. 2022.

[124] *pi-ina219 1.4.0*. https://pypi.org/project/pi-ina219/. [ONLINE]. 2022.

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

[126] Tianqi Chen and et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Carlsbad, CA, USA, October, 2018.

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

[128] Luyu Gao et al. "Scaling deep contrastive learning batch size under memory limited setup". In: *arXiv preprint arXiv:2101.06983* (2021).

[129] 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.

[130] *kerascv 0.0.40*. https://pypi.org/project/kerascv/. [ONLINE]. 2022.

[131] *tf.Graph – TensorFlow v2.4.1*. https://www.tensorflow.org/api_docs/python/tf/Graph. [ONLINE]. 2022.

[132] *Environment Variables – MXNet v1.7.0*. https://mxnet.apache.org/versions/1.7.0/api/faq/env_var. [ONLINE]. 2022.

[133] *Edge TPU Python API overview*. https://coral.ai/docs/edgetpu/api-intro/. [ONLINE]. 2022.

[134] Christine Bertram et al. "The blue carbon wealth of nations". In: *Nature Climate Change* 11.8 (2021), pp. 704–709.

[135] Peter I. Macreadie, A Randall Hughes, and David L Kimbro. "Loss of 'blue carbon' from coastal salt marshes following habitat disturbance". In: *PloS one* 8.7 (2013), e69244.

[136] C Kuenzer et al. "Remote Sensing of mangrove Ecosystem". In: *A Review Remote Sensing* 3.5 (2014).

[137] Kevan B Moffett et al. "Multiple stable states and catastrophic shifts in coastal wetlands: Progress, challenges, and opportunities in validating theory using remote sensing and other methods". In: *Remote Sensing* 7.8 (2015), pp. 10184–10226.

[138] United States Department of Agriculture. *Soil Health Technical Note No. 450-03: Recommended Soil Health Indicators and Associated Laboratory Procedures*. `https://directives.sc.egov.usda.gov/OpenNonWebContent.aspx?content=44475.wba`. [ONLINE]. 2019.

[139] Cécile Gomez, Raphael A Viscarra Rossel, and Alex B McBratney. "Soil organic carbon prediction by hyperspectral remote sensing and field vis-NIR spectroscopy: An Australian case study". In: *Geoderma* 146.3-4 (2008), pp. 403–411.

[140] Caiyun Zhang, Deepak R Mishra, and Steven C Pennings. "Mapping salt marsh soil properties using imaging spectroscopy". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 148 (2019), pp. 221–234.

[141] Quoc-Viet Pham et al. "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art". In: *IEEE access* 8 (2020), pp. 116974–117017.

[142] James R Holmquist et al. "Accuracy and precision of tidal wetland soil carbon mapping in the conterminous United States". In: *Scientific reports* 8.1 (2018), p. 9478.

[143] Wendi Qu et al. "Effect of salinity on the decomposition of soil organic carbon in a tidal wetland". In: *Journal of Soils and Sediments* 19 (2019), pp. 609–617.

[144] Craig Rasmussen et al. "Beyond clay: towards an improved set of variables for predicting soil organic matter content". In: *Biogeochemistry* 137 (2018), pp. 297–306.

[145] David Bruce Lewis, Jewel A Brown, and Kristine L Jimenez. "Effects of flooding and warming on soil organic matter mineralization in Avicennia germinans mangrove forests and Juncus roemerianus salt marshes". In: *Estuarine, Coastal and Shelf Science* 139 (2014), pp. 11–19.

[146] Jianwei Hao et al. "Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy". In: *ACM Transactions on Internet Technology* 23.1 (2023), 2:1–2:33.

[147] Piyush Subedi et al. "AI Multi-Tenancy on Edge: Concurrent Deep Learning Model Executions and Dynamic Model Placements on Edge Devices". In: *IEEE International Conference on Cloud Computing (CLOUD)*. 2021.

[148] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2016, pp. 785–794.

[149] Diego B. Haddad et al. “Brazilian Soil Bulk Density Prediction Based on a Committee of Neural Regressors”. In: *International Joint Conference on Neural Networks (IJCNN)*. 2018.

[150] Bernard F Tano et al. “Spatial and temporal variability of soil redox potential, pH and electrical conductivity across a toposequence in the Savanna of West Africa”. In: *Agronomy* 10.11 (2020), p. 1787.

[151] Jianli Ding et al. “Machine-learning-based quantitative estimation of soil organic carbon content by VIS/NIR spectroscopy”. In: *PeerJ* 6 (2018), e5714.

[152] Yongxing Ren et al. “Investigating spatial and vertical patterns of wetland soil organic carbon concentrations in China’s Western Songnen plain by comparing different algorithms”. In: *Sustainability* 12.3 (2020), p. 932.