University of California

Santa Barbara

Power Adaptive and Resource Aware Deployments for Remote Edge Systems

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy

in

Computer Science

by

Animesh Narayan Dangwal UCSB TR 2026-03

Committee in charge:

Professor Chandra Krintz, co-Chair

Professor Rich Wolski, co-Chair

Professor Shiyu Chang

August 2026

The Dissertation of Animesh Narayan Dangwal UCSB TR 2026-03 is approved.

Professor Shiyu Chang

Professor Chandra Krintz, Committee Co-Chair

Professor Rich Wolski, Committee Co-Chair

August 2026

Power Adaptive and Resource Aware Deployments for Remote Edge Systems

by

Animesh Narayan Dangwal UCSB TR 2026-03

Acknowledgements

Formally, this thesis has been supported in part by NSF award CNS-2107101, Norwegian Research Council DILUTE award 262854/F20, AirQMan award 322473, Oracle monetary and hardware support, NSF awards CNS-2444318, CNS-2107101, and DoE award DE-SC0025541. We also gratefully acknowledge the support and guidance from Eric Sedlar (Oracle Labs) and Chris Bensen (Lawrence Livermore National Lab), and for excellent technical support from Vic Breen and Justin Maness (University of California, Santa Barbara). And the support and help from my committee members throughout my PhD.

Informally, this thesis has been supported by my advisors, Chandra and Rich, who much to my surprise, got me to the finish line. I'm forever grateful to be their student whether they like it or not. Along with my friends, and family, who unquestionably supported all my questionable decisions. The staff across campus and Santa Barbara for the sustenance (coffee) provided. To all the professors who taught me everything I know.

A series of special thanks, in no particular order, rhyme or reason save for being part of my shenanigans: To my parents, Devna and Praveen, by virtue of everything. To my Nani, Indra, by virtue of her blessings. To Ziad, Elizabeth, Maritza, Jenna, Bella, Greta, Karen, Elliot, and the CS, IEE grad admin staff, by virtue of UCSB. To Abhe, Camille, Itir, Nazerke, Yeganeh, Pranjali, Sarah, Naomi, Parker, Madison, "non-unc"-Animesh, Guy, Minh, Juan, Guillem, Jon, Tim, and Arch Lab, by virtue of my sister. To tinytart, Ria, Karen, Shruthi, Emily, by virtue of ERSP. To Nevena, Shereen, Sierra, Liu, Sahiti, Nazmus, Tyler, Gautam, Peiyuan, Vinayak, Ben, Towhid, by virtue of being lab members past and present, I owe y'all 4 billion cups of coffee. To Nicole, Meghan, Valeria, Rhys, Nicholas, Dawson, Kai, by virtue of our lease agreements past and present.

And finally to my sister, Deeksha, who has easily been responsible for every good thing that's happened to me past, present and future.

Curriculum Vitæ

Animesh Narayan Dangwal UCSB TR 2026-03

Education

2026	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2020	B.Tech. in Computer Science, PES University

Selected publications

- Plumber: Scheduling Deep Learning Inference Pipelines on Resource-Constrained Edge Clusters, **Dangwal, A. N.** and Singh, R. and Unnithan, S. and Yuan, K. and Zheng, E. and Krintz, C. and Wolski, R., in-submission, 2026.
- Reliable Unreliability in Edge Deployments, **Dangwal, A. N.** and Singh, R. and Unnithan, S. and Yuan, K. and Zheng, E. and Krintz, C. and Wolski, R., published in International Conference On Distributed Computing In Smart Systems And The Internet Of Things (DCOSS), 2026.
- Designing Co-operation in Systems of Hierarchical, Multi-objective Schedulers for Stream Processing, **Dangwal, A.** and Jiang, Y. and Arnold, C. and Fan, J. and Bassem, M. and Rajagopal, A., arXiv, 2025.
- Energy-Aware IoT Deployment Planning, Guan, P. and **Dangwal, A.** and Taherkordi, A. and Wolski, R. and Krintz, C., published in Proceedings of the 21st ACM International Conference on Computing Frontiers, 2024.

Awards

2023-2024	Fellowship, Oracle Labs
2021, 2022	Fellowship, University of California, Santa Barbara

Service

2021, 2023	Graduate Mentor, Early Research Scholar's Program, University of California, Santa Barbara
2021-2022	Teaching Assistant, Computer Science, University of California, Santa Barbara

Note, this thesis contains text and figures from prior publications of the author.

Abstract

Power Adaptive and Resource Aware Deployments for Remote Edge Systems

by

Animesh Narayan Dangwal UCSB TR 2026-03

The Internet of Things (IoT) consists of low-cost, highly-specialized, resource constrained devices deployed in technologically limited environments to collect data, perform computations and actuate over the results. These deployments co-locate computers and storage systems at the "edge" of the network or as close to the environment they actuate over. Together these "edge deployments" localize communication, computation, and storage for security, with increased efficiencies (e.g. lower latency response), and reliability. To keep these deployments low-cost and sustainable, edge clusters increasingly rely on commodity hardware and shared power, network infrastructures.

Increasingly, advancements in applications like Machine Learning (ML), Artificial Intelligence (AI), and device hardware, have allowed these systems to go beyond just sensing. Applications that used to require offloading heavy computations to compute, network-rich environments like the cloud, can now sufficiently operate locally at the edge. However, unlike the cloud, with its abundant and stable compute, network and power resources, resource-constrained edge environments exhibit dynamic network and compute conditions, leading to fluctuating workload performance. This body of work showcases the impact of such computations, communications under realistic edge deployment conditions. Specially, by modelling and designing edge deployments with a power first approach, the dissertation shows how to account for these novel power-compute interactions on real edge clusters, and finally how to both efficiently and effectively utilize real edge clusters under these dynamic conditions for ML/AI applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) describes a computing fabric that enables ordinary, physical objects to sense, analyze, actuate, and control their environment automatically. These objects are embedded with computation and communication capabilities to enable intelligent decision-making, at or close to, the environment they operate in, also called the "edge" of the network. Here, these edge devices enable low-latency, low-cost, low-power decision-making, when compared to their cloud computing counterparts. Historically, these devices had low computational capabilities leading to applications offloading their compute to resource-rich environments like the cloud, with stable and sufficient sources of compute, network and power to support such applications.

For remote environments, IoT deployments co-locate compute and storage systems at the edge of the network, far from stable power and communication infrastructure. To sustainably add such infrastructure at scale, with low-costs, deployments share network and power, primarily relying on battery powered devices or running on renewable energy sources. While the advancements in hardware and fabrication technology, along with applications evolving to better suit IoT/edge devices, have made these devices ubiquitous and capable of handling a more diverse set of computations. The main study for these

improved hardware-software designs are edge applications like smart homes, smart cities, wearables, AR/VR, where pre-existing infrastructure can be extended to support these devices. Their interactions in remote edge deployments such as smart farming, wildlife conservation and forest fire monitoring are underexplored. Here, it is essential for cluster usability and shared infrastructure maintenance to take into account:

- Communication management, where allocating the limited available bandwidth across edge devices is paramount for reducing energy utilization and application performance,
- Computation deployment, where applications must select and schedule computations on edge devices to run both effectively and efficiently utilize the full system while balancing the limited power available to it.

These remote edge systems must take into account, not only the power expenditure and budget of IoT applications, but also the fluctuations in performance caused by the contention of their shared power and network infrastructure. Especially since these newer application optimizations and hardware systems were never designed to consider these fluctuations. Given these limitations, and gaps in the literature, our thesis statement is presented below.

1.1 Thesis Statement

This dissertation demonstrates that treating power infrastructure as a first-class systems resource throughout the lifecycle of an edge deployment enables efficient planning, accurate performance characterization, and effective runtime scheduling.

This statement is investigated for remote edge systems under shared network and power infrastructure, using three foci:

1. Developing a model for edge system deployments,
2. Characterizing these power-variable interactions in real edge systems
3. Designing an end-to-end scheduler accounting for dynamic resource conditions at the edge

1.2 Dissertation Outline

An initial approach to treating power as a first-class resource is by modelling major sources of expenditure in edge clusters and how to design such clusters to supply adequate power under different distribution infrastructures. Chapter 3 explores how power management dictates deployment topology for wireless communication, a major source of power drain, across edge devices in a remote environment for both shared and individual power sources. Specifically, the chapter uses popular resource allocation techniques, like constraint solvers, to minimize the maximum power spent in network transfers given an individual or shared power source for said devices. The communication is a simulated channel model, where given a total bandwidth and power budget, the distance between devices and amount of data to be transferred are inputs to the solver which outputs the bandwidth allocation per device to minimize power used. Additionally, the chapter show how to extend these solvers to decide device placement, data allocation, and batter size, for these deployments.

But these modelling, and planning techniques explored in Chapter 3, while useful, fail to consider device performance fluctuations caused by shared power contention in real edge deployments. Especially in long-lived edge deployments, behavior frequently deviates from these assumptions as infrastructure ages, components are replaced, workloads evolve, and resource contention changes over time. Consequently, deployment planning alone

is insufficient for maintaining efficient operation throughout the lifetime of an edge system. Chapter 4 addresses these concerns by characterizing the hardware and software interactions of real edge devices with shared power infrastructure. Using a 1050+ device, Raspberry Pi 3B+, cluster where ≈ 42 devices share a power distribution unit (PDU), the work illustrates contention between devices sharing a PDU by how compute performance is affected by the number of devices used within a shared PDU subcluster. This property of throttling due to compute load is exhaustively profiled, modelled and validated, into a regression equation, while also describing how to avoid exhaustive profiling and still deriving a valid regression formulation to describe system behavior.

From the findings of the previous chapters, Chapter 5 accounts for the varying performance of real edge devices caused by resource contention related to power and network to design an adaptive scheduler. Specifically the chapter explores dynamically decomposing ML models into sequential operations across a set of edge devices, for remote image processing applications, such as wildlife monitoring and forest fire detection. The workloads tested reflect real traces from Azure's public dataset and explore the scheduler's adaptability for different workload compositions as well when the ML models used have varying communication and computation requirements.

Finally, Chapter 6 presents the future work and broader impacts of the dissertation. A common limitation presented in the previous chapters is how the application-power interactions are explored for compute-bound applications specifically. Extending the methodology to include other operation interactions such as network and memory usage with power is required to expand the types of applications we can schedule. The chapter also presents ways to integrate these resource contention aware scheduling techniques into existing ecosystems in a plug and play fashion, while still achieving Pareto-optimality. The methodology presented provides a real example for large-scale datacenter computing [15] to demonstrate its scalability and potential application to edge computing as well.

1.3 Attributions

- Parts of chapter 3 include data and text from Energy-Aware IoT Deployment Planning [31] from authors Guan, P. and **Dangwal, A.** and Taherkordi, A. and Wolski, R. and Krintz, C., published in Proceedings of the 21st ACM International Conference on Computing Frontiers, 2024.

- Parts of chapter 4 include data and text from Reliable Unreliability in Edge Deployments [14] from authors **Dangwal, A. N.** and Singh, R. and Unnithan, S. and Yuan, K. and Zheng, E. and Krintz, C. and Wolski, R., published in International Conference On Distributed Computing In Smart Systems And The Internet Of Things (DCOSS), 2026.

- Parts of chapter 5 include data and text from Plumber: Scheduling Deep Learning Inference Pipelines on Resource-Constrained Edge Clusters [13] from authors **Dangwal, A. N.** and Singh, R. and Unnithan, S. and Yuan, K. and Zheng, E. and Krintz, C. and Wolski, R., publication pending.

- Parts of chapter 6 include data and text from Designing Co-operation in Systems of Hierarchical, Multi-objective Schedulers for Stream Processing [15] from authors **Dangwal, A.** and Jiang, Y. and Arnold, C. and Fan, J. and Bassem, M. and Rajagopal, A., available on arXiv.

Chapter 2

Background

In remote IoT applications, managing resources to keep the edge cluster usable for as long as possible before intervention of power or device replacement is an indication of effective and efficient usage of said resources. These resources are broken down into communication and computation requirements where an application is assigned a certain compute or communication budget. This chapter presents how resource management is explored for IoT, how compute-power interactions are underexplored and how existing adaptive schedulers for ML on the edge miss out on handling dynamic resource contentions.

2.1 Resource Management on Edge Clusters

Past work has focused on resource allocation for wireless sensor networks (WSNs) [76, 77, 87], given that communication is shown to be a major source of power consumption for edge devices. The authors of [48, 79, 84] show that numerical optimization solvers can be used effectively for scheduling resources, optimizing energy use, and improving quality of service for edge systems. Our work in chapter 3 differs from this past work in that we use independent objectives (versus sum-weighted) to minimize power consumption,

and we show how to use solver-based solutions to co-design power infrastructure and IoT deployments (sensor placement and duty cycle) at the edge. In later chapters that while useful, these models fail to account for power-induced variability in compute, caused by real infrastructure-device interactions. We also show how previous work in this area over-compensates for these interactions in potentially device-damaging ways.

2.2 Infrastructure-Application Interactions for Edge Clusters

Infrastructure degradation studies are common in hardware reliability and storage systems research [32, 68, 75], but are rare in edge and IoT computing contexts. Here, low-cost single-board computers such as Raspberry Pi are widely used to construct edge and IoT computing test beds for research and education [11, 18, 50, 72]. Prior work has explored the use of Raspberry Pi clusters for distributed computing, federated learning, sensor data processing, and cyber-physical systems experimentation [4, 44, 74, 85]. These platforms offer a cost-effective alternative to traditional datacenter infrastructure but typically rely on commodity power and networking hardware. To obtain stable and repeatable measurements, many studies intentionally pin CPU frequency, disable DVFS or supply elevated voltages, accelerating hardware degradation [5, 33, 42, 51, 70]. While these studies demonstrate the feasibility of large-scale edge clusters, they generally assume stable power delivery and do not examine how to handle power-induced performance effects – that are intrinsic to and critical for using real-world deployments.

Most existing Pi cluster studies evaluate freshly deployed hardware over short time horizons. Our study, in chapter 4 provides a characterization of a large-scale, long-serving cluster infrastructure that reveals systematic power-induced performance degradation

and emergent behavioral classes. This perspective is critical for understanding real-world IoT and edge deployments, which often operate for extended periods with minimal maintenance.

Power-aware computing has been extensively studied in datacenter and mobile systems, where voltage and frequency scaling (DVFS) is used to manage energy consumption and thermal constraints [35, 80, 81]. In edge and IoT systems, several studies have characterized power consumption and energy efficiency of single-board computers under varying workloads [5, 42]. However, prior work largely treats power as a controllable resource or a static constraint, rather than a degradable shared infrastructure that introduces heterogeneous and non-deterministic performance behavior across devices. In this vein, the next chapters explore how to extend this work outside the explored matrix-multiply workloads in chapter 4, for applications such as edge ML/AI.

2.3 Adaptive Scheduling for ML on Edge Clusters

We now explore how these power interactions impact similar computations, specifically, ML/AI applications. Improved edge AI/ML deployment strategies like pruning [88], compression [10], slicing [37], or quantization [34], typically target devices in isolation, on a single device [1], or set of devices [37], [38], [89]. Our work specifically targets realistic workload traces across multiple edge devices in a real edge cluster under shared power infrastructure. Past work assume static deployment conditions, unintentionally providing "one-size-fits-all" results, even if the optimizations to the models are dynamic for other parameters such as application accuracy [89], or device heterogeneity [38]. Another popular methodology to fit large models on small devices is pipelining, where a monolithic DL model is split into multiple parts and pipelined (Figure 5.1) to speed up training and inference [83], and dynamically creating these pipelines based on environment conditions have been

explored in cloud or data centers with stable network and power conditions [56], [52], or even on lower-end accelerators [39] or across low-network conditions [90], According to our knowledge, the work in chapter 5 is unique as, there is no prior work that is, energy efficient, adapts to shared resource contentions during workload execution, and where the models stay completely on the edge.

Chapter 3

Modelling Edge Deployments For Power Efficient Communication

3.1 Introduction

In this chapter, we investigate using power management as an approach for modelling and designing remote edge deployments. Specifically, we target deployment environments where electrical power infrastructure is lacking, forcing the edge devices or sensors to rely on renewable energy and battery storage for power. Recent research has shown that it is possible to formulate these resource management problems (e.g. for scheduling, energy optimization, improving QoS, etc.) as multi-objective, numerical optimization problems and to use optimization software (i.e. solvers) to efficiently produce possible solutions [16, 48, 64, 79, 84]. In this work, we use multi-objective optimization to efficiently allocate radio bandwidth to sensors in an edge deployment to minimize power consumption. We focus on radio communication because it is a primary consumer of power for wireless sensor networks that are remotely deployed and spatially distributed [40, 58, 60] – key characteristics (see Section 3.2) of the edge deployments that we target.

We model the power consumed for wireless communication between edge devices and their access points by using their physical deployment properties such as distance to access point, amount of data transferred, available bandwidth per device and time limit for data transfer, i.e. their duty cycle. Furthermore, we investigate three popular heuristic approaches: MIDACO [64], a Mixed Integer Programming (MIP) and Mixed Integer Non-Linear Programming (MINLP) solver, Z3 [16] a Satisfiability Modulo Theory (SMT) solver, and a Genetic Algorithm (GA) that we specifically designed for this project. The heuristics allocate radio bandwidth to devices placed at different distances from the access point such that the worst case energy use is minimized. We consider per-node and shared battery infrastructure and empirically compare the approaches using different experimental settings. We evaluate the impact of limiting computational budget and the use of different device duty cycles (which impose deadlines on data transfer completion). Finally, we present a methodology for using these techniques to guide energy efficient, configuration planning and sensor placement for remote, edge-based IoT deployments.

Our results show how power infrastructure, being shared or individual per device, impacts communication optimizations and how to design deployment topologies to better suit communication conditions to minimize power spent. We also see from our heuristic methods evaluated, MIDACO outperforms SMT and GA, and that SMT outperforms GA in terms of solution quality. In our study, MIDACO is also always able to generate feasible solutions, even with very limited computational effort budgets. However, in some cases where the computation effort budget is limited, SMT is able to find better solutions compared to MIDACO. It is just not possible to predict whether SMT will do so for a specific deployment and effort budget. This result indicates that it is "best" to expend a fraction of the available computational budget on SMT and if it fails to find a feasible solution in using that fraction, invoke MIDACO with the remainder. We also find that this approach can be used to size of the battery infrastructure for a given deployment

and that infeasible solutions can be intelligently adjusted (adapting sensor placement or duty cycle) while minimizing power use.

3.2 Edge Deployment Optimization

Our model of operation is inspired by real-world remote deployments that our group maintains as part of the UCSB SmartFarm project [22, 30, 41, 73]. In this setting, sensors monitor the environment, periodically transmit their data on a duty cycle over radio links to one or more base stations, and deactivate (e.g. go into "deep sleep") to conserve battery power between duty cycles. These deployments are employed by farms for a dynamically changing set of purposes related to precision agriculture, requiring them to be periodically updated for different use cases (by humans or ground robots). The base station is often grid-powered and part of a resource rich edge cloud (with intermittent Internet connectivity) that is sited on-farm in an outbuilding. In this work, however, we also explore the use case where the base station or stations are also solar and battery powered using off-grid power infrastructure with greater capacity than the infrastructure supporting the sensors. The sensor nodes are battery powered and each sensor can have its own battery or multiple sensors can draw from a shared battery (both alternatives are charged via solar power).

A key challenge of these deployments is the need to co-design the IoT device network *with* the power infrastructure which can impose incompatible constraints. The goal of our work is to develop tools that simplify this co-design process for edge deployments consisting of sensors, a communication access point, and base station that executes a deployment planning tool.

Toward this end, we present a model for radio communication between the sensors nodes and base stations in an edge deployment. We use multi-objective numerical optimization

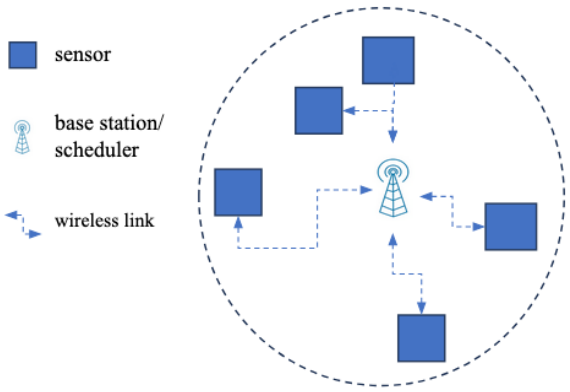

Figure 3.1: The figure illustrates an abstract model for the edge deployment topology where each block is a sensor, connected to the base station, and each sensor sends data of size D MB from a distance of d m to the base station.

to allocate the bandwidth to sensors in a way that minimizes power consumption of the sensors in the deployment. Our approach is unique in that we use it to facilitate comparison of popular optimization approaches as well as to support the co-design of power infrastructure, sensor placement, and data delivery for edge deployments. We employ two objective functions that we believe are useful in these settings. The first, called `min-max`, minimizes the maximum energy consumption, $max\{E_i^t\}$, in each data transfer cycle t. The objective corresponds to deployments in which each sensor node has its own battery and our goal is to minimize the maximum energy draw by any individual node. The second, called `min-sum`, minimizes the total energy consumption, $\sum^i E_i^t$, in each cycle of data transfer t. This objective corresponds to deployments that use a shared power source (e.g. a single solar powered battery infrastructure that each node uses as their power source).

3.2.1 System Model and Optimization Problem

The system model consists of a set of N nodes, each of which communicates using wireless radio communication with a single access point. As seen in Figure 3.1, each node

is assigned a data size D that it must transmit to the access point, and a distance d that gives its physical distance from the access point. All nodes share a common duty cycle during which time they must begin and complete the transfer of the data assigned to them. Each node can be configured to use a fraction of a total fixed bandwidth budget available during a duty cycle and to adjust its signal power according to the assigned fraction.

The main tunable parameters are bandwidth allocation and signal power usage to reduce the energy used per data transfer. Both the sensor and access point placement are known *a priori*, as is the amount of data that must be communicated in each duty cycle, meaning data and distance aren't tunable parameters and act as input to our model instead. All of these assumptions are from our previously mentioned execution model (SmartFarm micro-climate monitoring) where all sensors "wake up" simultaneously (so that readings are time correlated), transmit sensor readings over a radio communication link to the access point during a fixed duty cycle, and then "sleep" to conserve battery power until the start of the next duty cycle. This system model presents the problem where, given a set of nodes with assigned data transfer requirements and distances from the access point, the scheduler must provide the apportionment of bandwidth that minimizes the signal power each node must expend to transfer its data within the duration of a specified duty cycle.

3.2.2 Data Transmission Model

To model the wireless communication scenario, we use the path-loss function [28] shown in Equation 3.1 to calculate the influence of distance for each sensor i. In Equation 3.1, d_i is the distance (in meters) between sensor i and the base station, whereas f_i is the frequency (we set this to 5.9 GHz in this work) of the transmission signal. The units of

path loss (PL) computed by Equation 3.1 are decibels (dB). Parameters, such as 38.77, 16.7, and 18.2, are from the experiences in [28].

$$PL_i = 38.77 + 16.7 * \log_{10} d_i + 18.2 * \log_{10} f_i \tag{3.1}$$

This path loss needs to be overcome by the signal power used to transmit the information. The acceptance signal power, in decibel-milliwatts (dBm), is given by Equation 3.2 where P_i is the wireless signal power (in the range of 0.1 to 21 dBm) per the wireless standard [28]. Note that dB and dBm are on a log scale making the units of $P_i^{'}$ dBm. Converting $P_i^{'}$ to milliwatts (mW) using Equation 3.3 allows the computation of the signal-to-noise ratio (SNR) for each node using Equation 3.4 where N_0 is the power of environment noise, and it equals $10^{-11.4}$ mW [46]. Given the Signal-Noise-Ratio (SNR_i) and Bandwidth (B_i in MHz), the transmission speed (TS) of sensor i, is computed using the Shannon equation shown in Equation 3.5.

$$P_i^{'} = P_i - PL_i. \tag{3.2}$$

$$P_i^{''} = 10^{\frac{P_i^{'}}{10}} \tag{3.3}$$

$$SNR_i = \frac{P_i^{''}}{N_0 * B_i} \tag{3.4}$$

$$TS_i = B_i * \log_2 (1 + SNR_i) \tag{3.5}$$

With the given signal power and SNR, we can calculate the transmission speed (TS) from Equation 3.5, and dividing by data size D (in MB) gives transmission time (TT), from Equation 3.6. This helps determine bandwidth allocations, as we specify TT must

be less than or equal to the deadline imposed by the duty cycle for the deployment for all sensors. All together the sensor energy consumption i is obtained via Equation 3.7. It is worth noting that there is a trade-off between P_i and TT_i. If B_i is fixed, then when P_i increases TT_i decreases. If P_i is fixed, then the allocation of B_i is non-linear. Thus, in terms of the data transmission model, the problem of determining the values of B_i and P_i that yield a minimized set of E_i values, subject to the constraint that all TT_i are less than or equal to a fixed deadline (imposed by a given duty cycle), is an optimization problem we must solve.

$$TT_i = \frac{D_i}{TS_i}. \tag{3.6}$$

$$E_i = TT_i * 10^{\frac{P_i}{10}}. \tag{3.7}$$

3.2.3 Optimization Solvers

Given the formal system and communication models from the previous sections, we define two different optimization problems to represent different power supply infrastructures. The first, termed `min-max`, is shown in Equation 3.8. It minimizes the maximum value of E_i for a set of sensors. This objective function corresponds to a deployment in which each sensor has its own separate power source that can be exhausted (e.g. a battery). The second function, termed `min-sum` is given by Equation 3.9. It uses the sum of E_i as an objective function. This function represents the case where the set of sensors share a common power source that can be exhausted. We investigate three approaches for minimizing $F(O)$: mixed-integer non-linear programming, satisfiability modulo theories (SMT), and a genetic algorithm (GA).

$$F(O) = max\{E_i\}, \tag{3.8}$$

$$F(O) = sum\{E_i\}. \tag{3.9}$$

MIDACO

Our problem formulation natively supports a numerical solver that can represent individual sensors and handle range based constraints to specify valid values. MIDACO is a commercial solver for numerical optimization that is based on the Ant Colony algorithm [63], which can be applied to continuous non-linear optimization (NLP) problems, discrete/integer (IP) optimization problems, and mixed integer (MINLP) problems [64]. In our scenario, the deployment is specified as a set of data transmission requirements D_i, a set of distances d_i and a deadline for a set of sensors indexed by i which is an integer value in the range 1 to N. To better suit the problem structure MIDACO solves, we formulate our objectives (minimize $F(O)$) specified by Equation 3.8 or Equation 3.9, as a set of internal constraints that any feasible solution (i.e. one that meets the deadline) must also obey. These internal constraints are shown in Equation 3.10 where G(0)=0 and other G()≥0. N is the number of sensors.

$$\begin{aligned} G(0) &= B_1 + B_2 + ... + B_N - total\ bandwidth \\ G(1) &= deadline - TT_1 \\ &... \\ G(i) &= deadline - TT_i, \end{aligned} \tag{3.10}$$

SMT and Z3

Another approach to solving our formulation is by leaning towards the constraints and their relationship to instead repeatedly find valid combinations and present the most optimal ones based on the effort put into the search. For this, we select Z3, an SMT solver,

suited for boolean constraints and optimization capabilities for equational constraints that are linear [6]. As an SMT solver, Z3, includes limited capabilities for determining the satisfiability of constraints that require the evaluation of numerical variables. In particular, the support in Z3 for log functions, exponentiation, and real-valued variables either performs poorly or is non-existent. Thus, to use Z3 for optimizing energy usage, we transform the real-valued, non-linear problem formulation described in the previous subsection into an integer-valued, linear problem.

We first discretized the integer representation and represent the specific proportional relationships between PL and TS. As seen in Equation 3.11 shows a discretized integer representation of P_i. We limit the range of signal power values that the access point and sensors can adopt the range 0.01 to 21 dBm. We then multiply this value by 100 so that it is expressed as an integer with a value between 1 and 2100. Equation 3.12 captures the total bandwidth budget (shown as an internal constraint in the MIDACO formulation in Equation 3.10) as an independent and explicit constraint for the SMT formulation. To capture the inverse relationship between a signal power and bandwidth allocation shown in Equations 3.1 through 3.5, we add a constraint that forces the solver to minimize the maximum signal power P_i while, at the same time, maximizing the minimum bandwidth B_i for all sensors, as shown in Equations 3.13 and 3.14, respectively. Similarly, for the objective of minimizing the sum of energy usage, we substitute Equations 3.15 for Equation 3.13 in the constraint set to compute `min-sum`.

Because Z3 cannot evaluate non-linear, real-valued, log functions, it cannot incorporate a deadline constraint because it cannot solve Equations 3.1 through 3.6. However, it is able to find multiple unique solutions for a single constraint problem. We exploit this feature to "search" the constraint space for the best solution (the one with the lowest energy objective function value). To do so, we run the solver until it finds a solution and then apply Equations 3.1 through 3.6 to the resulting values to determine if the solution

meets the deadline. We also compute $F(O)$ using Equation 3.7 for the solution. If the solution meets the deadline, it is saved in a set of feasible solutions. If it does not, but its $F(O)$ value is lower than the $F(O)$ value of a previous infeasible solution, it is saved as the "best" infeasible solution. We then continue the solver and repeat the process for the next solution it finds. The search terminates after either 900 iterations (the maximal case) or a fixed time limit (see Section 3.3). We explore the use of feasible and infeasible solutions in Section 3.4.

$$1 \leq P_i \leq 2100,\ 1 \leq B_i \leq 100 \tag{3.11}$$

$$\sum_{i=1}^{N} B_i == total\ bandwidth \tag{3.12}$$

$$\forall_{i=1}^{N} P_{max} \geq P_i,\ minimize(P_{max}) \tag{3.13}$$

$$\forall_{i=1}^{N} B_{min} \geq B_i,\ maximize(B_{min}) \tag{3.14}$$

$$minimize(\sum_{i=1}^{N} P_i) \tag{3.15}$$

To guide the solver toward better solutions in the space, we add a "weighting" constraint that creates a relationship between the product of the data size and distance and the sum of the signal power and allocated bandwidth. This search process can be improved by adding a constraint that captures the influence of the data size d_i that each sensor must send and the distance D_i from the access point for each sensor. Specifically, we would like the solver to avoid solutions where nodes sending large amounts of data that are located

far from the access point are assigned less bandwidth than nodes closer to the access point with smaller data transfer requirements. There may be many such solutions that satisfy the constraints but result in infeasible (and poor) allocations. Equation 3.16 shows this "guidance" constraint. Note that this is the only constraint in the SMT formulation that relates bandwidth allocation B_i to signal power P_i – without it the solver is free to find solutions that assign values to these variables independently. As such, it constitutes a linear multi-objective function for Z3 to optimize.

To speed up the iterative searching, we also use a Z3 feature that prioritizes Pareto-optimal solutions to multi-objective functions [6]. Doing so ensures that the relationship in Equation 3.16 is favored by Z3. This optimization to the constraint search introduces additional complexity, But using Pareto priority, Z3 can return repeated solutions when it cycles through all the Pareto points the solver can find. When the solver returns repeated solutions, we introduce a "soft" constraint that allows it to consider additional possible solutions. When the solver stops and offers a repeated solution, we compute a *target* P_i for each node based on the d_i, D_i values for the node from the deployment and the B_i in the current (and repeated) solutions. We then add the soft constraint shown in Equation 3.17 if *target* P_i satisfies the constraint shown in Equation 3.11. The soft constraint forces Z3 internally to search for new solutions using Pareto priority but in the course of that search, if it can find a lower value for the power as an optimal point, it can select it even if it violates the soft constraint. Thus, the soft constraint allows Z3 to make an "up-hill" climb in the search space in its search for Pareto optimal solutions.

$$\forall_{i=1}^{N}(D_i * d_i \leq D_{i+1} * d_{i+1} \implies B_i + P_i \leq B_{i+1} + P_{i+1}) \tag{3.16}$$

$$\begin{aligned} &\#target\ P_i\ computed\ using\ Equations\ 3.1\ through\ 3.6 \\ &\#from\ the\ values\ of\ B_i,\ D_i, d_i \\ &(add\ soft)\ P_i \geq target\ P_i \end{aligned} \tag{3.17}$$

Genetic Algorithm

The final heuristic we approach to solve our problem formulation is based on customizable greedy numerical optimizers. For this we develop a custom Genetic Algorithm (GA) for comparison in this study given it is a widely used heuristic-based optimization approach. In our GA formulation, bandwidth $\{B_i\}$ and signal power $\{P_i\}$ are represented as separate chromosomes. The GA first creates a group of random chromosomes as parents and crosses each *father* in the group with a random *mother*. A cross is performed by selecting a random position for each chromosome and concatenating the mother with the father using this position. For example, a child bandwidth chromosome using random position k consists of father bandwidth items B_1–B_k followed by mother bandwidth items B_{k+1}–B_n for list length n. After the crossing, the sum of child bandwidth may differ from the total bandwidth. We update the items to conform to this constraint. Specifically, we compute $\Delta = sum\{child\_B_i\} - total\ bandwidth$. If $\Delta > 0$, we select a random item in the $child\_B_i$ and subtract 1 from it (if $\Delta < 0$, we add 1). We repeat this process $\Delta - 1$ times. After crossing, we perform mutation to introduce chromosome diversity. We choose a random number $\mu \in (0, 1)$ and use it to mutate the child items. For bandwidth, if $0.5 < \mu < 0.85$, we do not mutate. If $\mu \geq 0.85$, we left-rotate the items one position. If $\mu \leq 0.5$, we subtract 1 from the maximum item and add 1 to the minimum item in $child\_P_i$. For power, we use a different range for $\lambda \in (0, 1)$. For each item in $child\_P_i$, if $0 < \lambda < 0.15$, there is no mutation. When $0.15 \leq \lambda \leq 0.6$, we add or subtract 1 to/from $child\_P_i$ using a 50-50 probability, and we only update the value if doing so results in a

value that is in the range (0.1 dBm, 21 dBm). Similarly, we mutate each item ± 0.1 and ± 0.01, when λ is in the range (0.6, 0.85) and (0.85, 1), respectively.

3.3 Solver Evaluation

In this section, we evaluate three different solvers: MIDACO, Z3 (SMT), and a genetic algorithm (GA) written specifically for this work. The overall goal of this investigation is to develop a tool for aiding deployment design in remote IoT settings (see Section 3.4). Doing so requires answering the following research questions:

- RQ1: Are all solvers capable of minimize energy consumed for both individual and shared power supply infrastructures?
- RQ2: Does quality of solution change based on computational effort across solver heuristics used?
- RQ3: How do we use these solvers? Is there always one heuristic that performs the best or do we require a system of solvers per deployment problem?
- RQ4: Can these solvers determine valid deployment topologies, and in-turn suggest feasible designs to generate valid deployments?

3.3.1 Experimental Setup

To explore solver efficacy, we generate a set of 100 randomized deployments and compare the solvers using this set as an input. Each deployment in the set consists of 10 sensor nodes, where a node's data transmission requirement is drawn from a uniform distribution on the interval $(50, 150)$ and its distance from the access point is drawn from a uniform distribution on the interval $(50, 200)$. For a specific individual deployment, we

measure solution quality as the energy resulting either from minimizing the `min-max` or the `min-sum` objective function. All the experiments were run using an AMD Ryzen 7 5800H @3.20 GHz CPU with 16GB RAM and 4 cores. For the mixed-integer programming solver, we use MIDACO Version 6.0 under a commercial license, for SMT, we use Z3 Version 4.12.2, which is available as open source from [86], and we developed the genetic algorithm as a bespoke approach specifically for this chapter. In all experiments, the units of data transmission size are converted to megabits (from MB), the distance units are meters, and we measure energy consumption in millijoules (mJ). When reporting aggregate solution quality statistics (e.g. minimum, maximum, mean), we set a fixed duty cycle (the units are seconds) for the 10 nodes and aggregate over the 100 randomized deployments.

Finally, to capture the "effort" each solver requires creating a solution, we define the "computational effort budget" to be the maximum wall-clock time the solver is allowed to use to compute a solution. A solver may use less than this budget, but not more. We define an "infinite" computational effort budget to be 5 hours. Note that it is possible to generate a deployment for which there is no allocation of bandwidth that will allow all 10 nodes to finish transmission within the specified duty cycle. It is also possible that a solution exists, but that a solver is not able to "find" it within its computation effort budget. Finally, it is possible that multiple solutions exist, in which case, we expect that the solver will return the "best" one (i.e. the one that minimizes the objective function). We term a solution that allows all 10 nodes to complete transmission before the end of the duty cycle a "feasible" solution. We select the 100 deployment evaluation set such that all 100 deployments have feasible solutions given a 2 second duty cycle (however, a particular solver may not be able to generate a feasible solution for all 100 deployments).

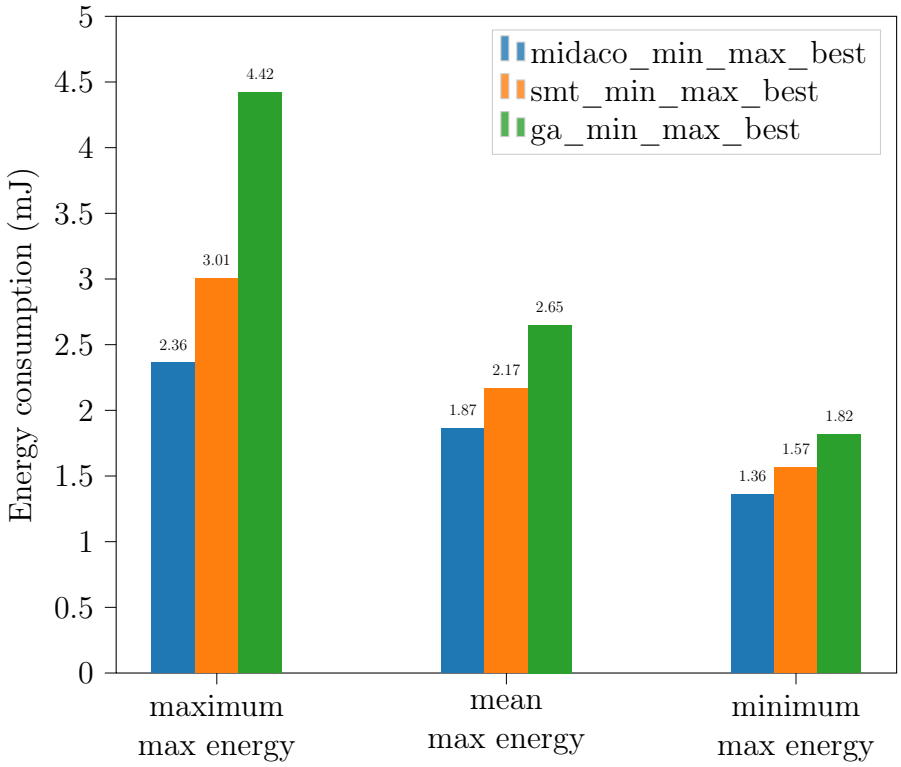

Figure 3.2: The figure compares the solvers for the `min-max` objective function and an unlimited computational effort budget. The y-axis is maximum energy consumption over the 10 nodes (in mJ) and x-axis represents the maximum, mean and minimum of over 100 randomized experiments. The duty cycle is 3 seconds. The lower the energy consumed the better. MIDACO's solutions always generate the best allocation followed by SMT and then GA.

3.3.2 Unlimited Solver Computational Effort

Figure 3.2 shows the efficacy of each solver for the 100-deployment test set with respect to the `min-max` objective function (see Equation 8): a practically unlimited computational effort budget, and a 3-second duty cycle. In the figure, the best (lowest) maximum energy, mean energy, and minimum energy were generated by the MIDACO solver. The SMT solver generated solutions that were somewhat less energy efficient, and the GA solutions were less efficient than the SMT solutions. This comparison illustrates the effectiveness of

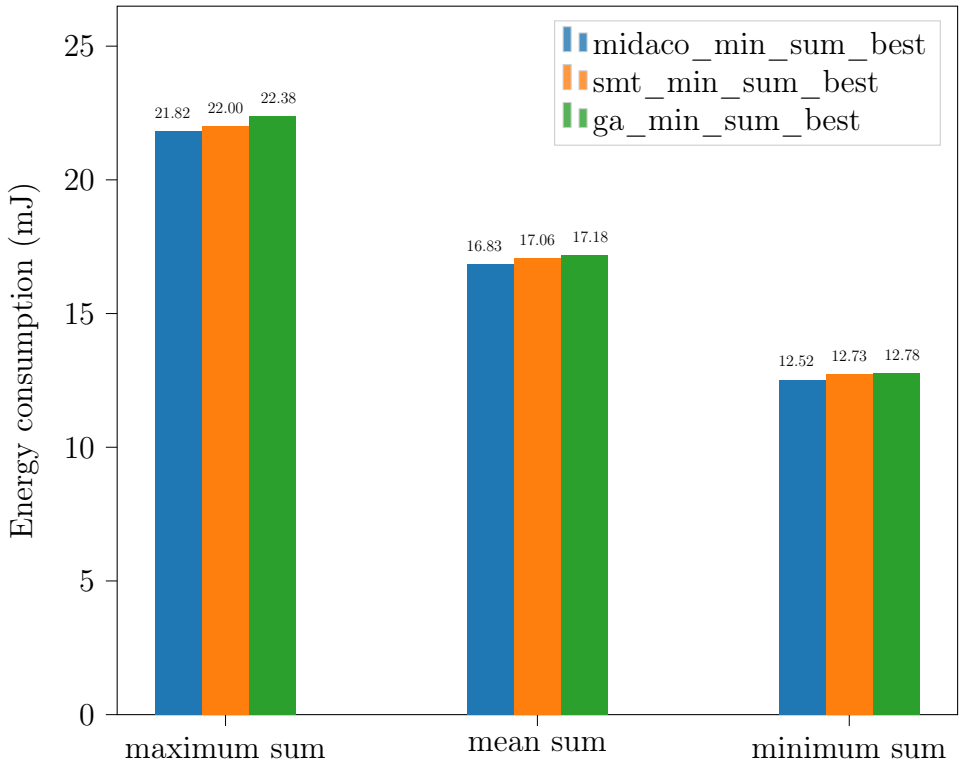

Figure 3.3: The figure compares the solvers for the `min-sum` objective function and an unlimited computational effort budget. The y-axis is the maximum energy consumption over the 10 nodes (in mJ) and x-axis represents the maximum, mean and minimum of over 100 randomized experiments. The duty cycle is 3 seconds. The lower the energy consumed the better. All solutions from all solvers are extremely close to each other, indicating a simpler problem space than `min-max` in Figure 3.2.

using a high-quality, commercial solver over perhaps a more intuitive constraint-based approach (SMT) or a solution that is easy to conceive, develop, and debug (GA), given an unconstrained computational effort budget.

Figure 3.3 shows the same comparison as depicted in Figure 3.2, but for the `min-sum` objective function (see Equation 3.9). These results indicate that with "infinite" computational effort, each optimization technique generates solutions that are approximately equivalent in terms energy efficiency in each deployment. That is, it is "easier" to optimize the total energy consumption (i.e. `min-sum`) for the collection of sensors compared to

optimizing the energy consumption for individual sensors (i.e. `min-max`). As such, this result argues for deployment architectures in which the power infrastructure is shared rather than implemented separately for each sensor. Note that in our experiments, GA substantially underperforms MIDACO and SMT for the `min-max` objective function (and all three are approximately equivalent for the `min-sum` objective function). Thus, for brevity, we focus in comparing MIDACO to SMT using `min-max` for the remainder of this investigation.

3.3.3 Limited Solver Computational Effort

We next compare the solvers in terms of their computational efficiency relative to the energy efficiencies of the solutions they produce. In these experiments, we limit the computational effort budget given to each solver and look at the quality of the solutions that are generated within the budget.

The Figure 3.4 illustrates the average "efficiency" of the MIDACO and SMT solvers in terms of solution quality. It plots the average maximum energy consumption of the 10 nodes across the 100 deployments in the test set on the y-axis, as a function of the computational effort budget shown on the x-axis. Note that SMT is not able to find at least one feasible solution for all 100 test deployments with a computational effort budget of less than 17 seconds (the graph only depicts averages calculated over 100 test deployments). In contrast, MIDACO is able to find *some* feasible solution for all 100 deployments using a computational effort budget of as little as 1 second. Moreover, the average quality of the solutions (the average energy consumption) improves rapidly with additional effort budget until approximately 6 seconds, after which the improvement per additional second of solver computation increases far more slowly until it is near zero.

Overall, as each solver is given more computational effort budget the quality of the

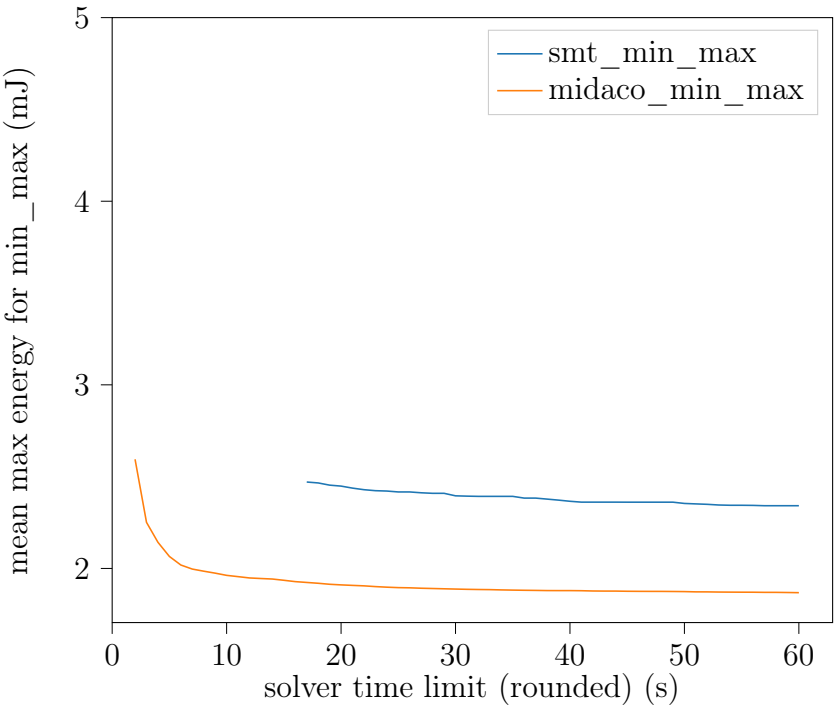

Figure 3.4: The figure compares the solvers for the *min-max* objective function when each solver is given the same computational effort budget. The y-axis is the average maximum energy consumption over the 10 nodes (in mJ) and x-axis represents the computational effort budget used to generate the corresponding y-axis value. The duty cycle is 3 seconds. The lower the energy consumed the better, and closeness to the left indicates faster solution generation. MIDACO is both lower and closer to the left, indicating on average, better solution quality and solution generation speed compared to SMT.

solution improves (i.e. the mean energy consumption is lower). Moreover, for the same computational effort budget of 17 seconds or greater, the mean solution quality generated by MIDACO is higher (i.e. the mean maximum energy consumption is lower) compared to SMT. Thus, in terms of average expected performance against a fixed computational effort budget, MIDACO is more performant than SMT. However, while SMT does not a produces a feasible solution for all 100 test deployments when the effort budget is less than 17 seconds, it does find feasible solutions for a subset of the 100 test deployments. Table 3.1 shows a sample of 5 deployments from the deployment set. Column 1 shows the deployment index, column 2 shows the maximum energy consumption (in mJ) over the 10 nodes in the deployment and the time (in seconds) taken by MIDACO (in parentheses) which was limited to 4 seconds. Column 3 shows the energy consumption (in mJ) and the time (in seconds) taken by SMT to find its *first* feasible solution for that particular experiment. In column 4, we show the "chase" time – the additional time (in seconds) MIDACO required to find a solution equivalent or better than the first solution found by SMT shown in column 3. For example, in *Exp #11*, MIDACO achieved a solution in which the maximum energy consumption was 6.02 mJ in 4 seconds. SMT found its first feasible solution, corresponding to a maximum energy consumption of 2.80 mJ in 6 seconds. MIDACO required an additional $12 seconds$ to achieve a solution in which the maximum energy consumption was less than or equal to the 2.80 mJ achieved by SMT.

Thus, when SMT finds a solution with little computational effort budget, it can be considerably better than the solution MIDACO finds with the same computational effort budget. These results suggest that we can use SMT and MIDACO in concert when the computational effort is small. Specifically, we devote one half of the computational effort budget to SMT and the other to MIDACO. If SMT returns a solution, we compare it to the solution generated by MIDACO and use the solution that results in less energy consumption. Otherwise, we use the MIDACO solution.

Table 3.1: Solution energy (mJ), solver duration (seconds), and chase time for 5 specific randomized deployment solutions. Unlike the Figures 3.5, 3.4 the table illustrates when SMT can be both faster and generate better quality solutions than MIDACO.

Exp #	MIDACO solution mJ (sec)	SMT solution mJ (sec)	chase sec
11	6.02 (4)	2.80 (6)	12
51	2.04 (4)	1.99 (1)	11
52	3.21 (4)	2.40 (2)	14
61	2.66 (4)	2.21 (12)	11
76	2.55 (4)	2.25 (<1)	1

Finally, the "complexity" of the optimization problem each solver must solve is roughly proportional to the deadline imposed by the duty cycle. As duty cycle decreases, feasible solutions become more "difficult" to find. Figure 3.4 shows results for a duty cycle of 3 seconds and Figure 3.5 shows results for a duty cycle of 2.8 seconds. Comparing Figures 3.4 and 3.5 which show the mean solution quality (in mJ), SMT is unable to find 100 feasible solutions in less than 35 seconds of solver time with a duty cycle of 2.8 seconds compared to 17 seconds of solver time when the duty cycle is 3 seconds. When the solver time is sufficient in both cases, the average quality of the corresponding solutions is approximately the same. Thus, the length of the duty cycle affects the ability of SMT to find a feasible solution with a fixed computational effort budget and shorter duty cycles require larger budgets.

3.3.4 Discussion of Solver Comparison

For minimizing maximum energy consumption, the problem is a non-linear mixed integer programming optimization problem where only one quantity (communication bandwidth) is integer-valued (described in Section 3.2). Given this formulation of the optimization problem, it is perhaps unsurprising that MIDACO is almost always more effective than SMT. MIDACO is a high-quality commercial solution designed specifically

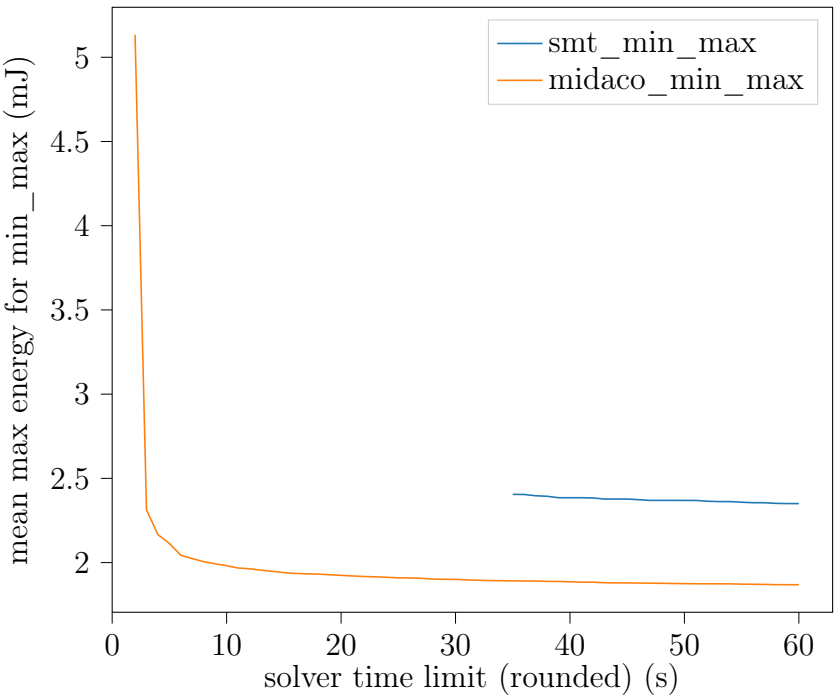

Figure 3.5: The figure compares the solvers for the *min-max* objective function when each solver is allowed the same computational effort. The y-axis is the average maximum energy consumption over the 10 nodes (in mJ) and x-axis represents the computational effort budget used to generate the corresponding y-axis value. The duty cycle is 2.8 seconds. The lower the energy consumed the better, and closeness to the left indicates faster solution generation. No clear winner here, while on average, MIDACO is both lower and closer to the left eventually, depending on how fast the solution is generated SMT could potentially be better as seen in Table 3.1

for such problems, answering RQ1.

For computational effort correlating with solution quality, it depends on the scenarios and solvers used. The SMT solver can find a "good" solution with limited computational effort for *some* deployments and that solution may be considerably better than the one generated by MIDACO with the same effort budget. From a practical perspective, when the computational effort budget is small, these results militate for "trying" SMT for a given deployment using some fraction of the effort budget to possibly generate a "good" feasible solution. And then giving the remainder of the budget to MIDACO (which is sure to generate a feasible solution) to exploit the strengths of each solver, answering RQ2 and RQ3.

3.4 Solver-aided Deployment Design

The results discussed in Section 3.3 for a randomized test set indicate that the MIDACO solver is generally more performant (in terms of solution quality) compared to SMT or GA for the individual power-source scenario. In this section, we outline our use of a solver to aid in deployment planning.

Our overall approach is for the deployment engineer or maintenance staff member to generate a list of data transfer requirements (one per sensor node), a set of distances between each node and the access point, and a duty cycle. Solving the `min-max` or `min-sum` problem yields a list of signal power requirements and bandwidth allocated to each sensor for each duty cycle. From this information, the battery "drain" (in units of power) required by each node in the deployment can be computed by multiplying the energy required by the length of the duty cycle and computing how many duty cycles take place during the worst case hours of darkness. Using the charging rate and the worst-case hours of available solar, it is possible to size the solar capacity to match or slightly exceed

the battery drain such that both are minimal. At present, deployment engineers typically overprovision the power infrastructure in these settings using the manufacturer-reported rates and maximal energy use by all devices. Using a solver-based approach, the expense and infrastructure footprint, compared to a maximal approach, can be reduced.

For example, consider the deployment transfer requirements and distances shown in Equation 3.19. For a 3-second duty cycle, the MIDACO solver generates the energy consumption estimates shown in Equation 3.20 for the 10 sensors using the `min-max` objective function. Converting mJ to Watts (W) for a 3 second duration (the length of the duty cycle) yields the value in Equation 3.21 for each duty cycle. In California (at one of our test site locations), the longest nighttime duration in 2024 will be 14.12 hours (on December 23). The battery drain for each sensor due to communication, then, is the product of the values shown in Equation 3.21 and the number of 3-second duty cycles that will take place during the hours of darkness. As an example, if the sensors are "awake" for a 3-second duty cycle every minute, the batteries must sustain 848 duty 3-second cycles in 14.12 hours. Multiplying the values in Equation 3.21 by 848 would yield the battery drain, in Watt-seconds, that the battery must sustain and recharge in every diurnal cycle due to communication.

3.4.1 Making Infeasible Solutions Feasible

Similarly, it is also possible to adapt our methodology to cases where the solver cannot find a feasible solution. As discussed in Section 3.3, longer duty cycles (at the possible expense of additional battery capacity) improves optimization performance. However, if the deployment functionality depends on a specific duty cycle for which the solver yields no feasible solution, it still may be possible to employ our approach. In particular, if the sensor nodes can be moved closer to the access point or the amount of data they

are required to transfer per duty cycle can be reduced, it is possible to take the "best" infeasible solution and adjust either the distances or the transfer sizes to make that solution feasible.

For example, consider again the deployment configuration shown in Equation 3.19, the `min-max` objective function, and a duty cycle of 2 seconds. The MIDACO solver does not generate a feasible solution for this deployment using an unlimited computational effort budget (and SMT does not either). However, it does allow us to rank the infeasible solutions in terms of the objective function and the "best" of the infeasible solutions is shown in Equation 3.18. This solution is infeasible because the first and fourth sensors in the list cannot complete their communication transfers within the 2-second duty cycle. That is, in Equation 3.18, the first and fourth sensor completes their data transfer in 2.53 and 2.08 seconds, which is longer than the 2-second duty cycle.

$$\begin{aligned} energy(mJ) &= [2.58, 1.67, 1.33, 2.13, 1.55, \\ &\quad 1.6, 1.17, 1.99, 1.55, 1.34] \\ transfer\_time(s) &= [2.53, 1.63, 1.30, 2.08, 1.51, \\ &\quad 1.56, 1.14, 1.94, 1.51, 1.31] \end{aligned} \tag{3.18}$$

There are two main ways to change this to a feasible deployment, first is to change the data transfer size, and the other being the distance from base station. We initially focus on changing the size of the data, to satisfy the duty cycle constraint. In Equation 3.6, we keep the value of deadline as 2 seconds, the data size as an unknown variable to solve for, and the transmission speed is the transmission speed we derive from Equations 3.1 through 3.5. Here, we use the values of signal power and bandwidth from the "best" infeasible solution and keep the same distance value from Equation 3.19 for sensor 1 and sensor 4. After deriving the transmission speeds for each sensor, we solve for the data sizes

that can satisfy the duty cycle constraint, using the other values from "best" infeasible solution generated by MIDACO. This means the worst case energy consumption is still minimized to the extent it could be by the "best" infeasible solution MIDACO generated. Using this adjustment generates data size values shown in Equation 3.22. Note that a comparison of Equation 3.22 to the data list in Equation 3.19 for sensors 1 and 4 indicates that reducing the data transfer of sensor 1 from 106 megabits to 86.76 megabits, and sensor 4 from 133 megabits to 127.33 megabits creates a feasible solution for a duty cycle of 2s.

Alternatively, if the data size cannot be changed, but the sensor can be moved, it is possible to perform a similar adjustment for distance using Equation 3.1. This process of derivation is a bit more involved as we start backwards from Equation 3.6 to calculate the target transmission speed that would meet the duty cycle constraint of 2 seconds, i.e. the transmission time would be 2 seconds and the data size would be the same as the input (from Equation 3.19) for sensors 1 and 4. Consider sensor 1 as an example. From its target transmission speed, we can derive the value of the target SNR using Equation 3.5. Here we keep the bandwidth value B_1 as the value from the "best" infeasible solution generated from by solver. Using this value, we derive the target $P_1^{''}$ from Equation 3.4 and then calculate the target $P_1^{'}$ from Equation 3.3. Finally, we calculate the path loss (PL_1) for sensor 1 from Equation 3.2, keeping target $P_1^{'}$ from the previous equation and the signal power P_1 as the value received from the "best" infeasible solver solution. We substitute the value for path loss in Equation 3.1 and solve the equation for d_1, the distance value for sensor 1. The same steps apply for sensor 4 as well. The modified distance values are shown in Equation 3.23. As with adjusting the data requirements, we can compare Equation 3.23 to the distance list in Equation 3.19 for sensors 1 and 4. We find that moving sensor 1 from 156 meters from the access point to 86.76 meters, and sensor 4 from 67 meters to 58.74 meters, creates a feasible solution for a duty cycle of 2 seconds.

The methodology works for either scenario as the data size transfer, distance are independent across sensors (i.e. changing a value in one does not impact the value of another). Thus, for those applications where the duty cycle cannot be extended, and the solver cannot find a feasible solution, the deployment engineer can choose to move sensors or to re-code sensors to transfer less data per duty cycle (or to move some and re-code others) in the "best" infeasible solution to create a feasible one. If the engineer's goal is to make the smallest possible change to the deployment that is infeasible, understanding whether changing data transfer or distance requires a larger perturbation.

$$\begin{aligned} data(MB) &= [106, 91, 136, 133, 121, 96, 113, 85, 78, 52] \\ distance(m) &= [156, 97, 29, 67, 55, 75, 35, 199, 184, 178] \end{aligned} \tag{3.19}$$

$$\begin{aligned} energy(mJ) = [&1.74, 1.67, 1.75, 1.70, 1.67, \\ &1.60, 1.64, 1.67, 1.75, 1.75] \end{aligned} \tag{3.20}$$

$$\begin{aligned} power(W) = [&0.58, 0.56, 0.58, 0.57, 0.56, \\ &0.53, 0.55, 0.56, 0.58, 0.58] \end{aligned} \tag{3.21}$$

$$new\_data(MB) = [83.77, 91, 136, 133, 121, 96, 113, 85, 78, 52] \tag{3.22}$$

$$new\_dist(m) = [86.76, 97, 29, 58.74, 55, 75, 35, 199, 184, 178] \tag{3.23}$$

These results illustrate how solver-aided deployment design can be used to "co-design" a deployment. In the cases where a deployment results in a feasible solution, the power infrastructure capacity that is necessary can be directly computed. When no feasible solutions can be found, the approach allows a deployment engineer to understand what duty cycle (and resulting power infrastructure capacity) is feasible (by experimentally

extending the duty cycle). When an acceptable duty cycle cannot be found, the engineer can use the method to determine a feasible solution from the best infeasible solution. Thus, when there is a choice, it is possible to determine whether changing the data payload for one or more sensors or moving one or more sensors (or some combination) will perturb the deployment less and fit the goals of the deployment best. Engineers can then make informed decisions about how to change a deployment to meet the objectives, thus presenting an answer to RQ4.

3.5 Related Work

We investigate three popular heuristics for allocating radio bandwidth to sensors in an edge deployment, such that power consumption is minimized. They include MIDACO [64], Z3 [16], and a custom genetic algorithm (GA; see Section 3.2.3). MIDACO is a numerical high-performance solver for single- and multi-objective optimization. MIDACO is based on a derivative-free, evolutionary hybrid algorithm that treats the objective and constraint functions as a black-box which may contain critical function properties like non-linearity, non-convexity, discontinuities, or stochastic noise.

Satisfiability Modulo Theories (SMT) solvers can also be used for constraint solving and optimization. They are used for application scheduling [2, 3, 7, 62, 67], resource allocation for the edge-cloud continuum [7, 25, 55, 67], and network optimization in Wireless Sensor Networks (WSNs) and Software Defined Networks(SDNs) for time sensitive applications [20, 27, 55, 66]. SMT is also considered a competitive alternative [6, 27, 62] to Mixed Integer Programming(MIP)/Mixed Integer Linear Programming (MILP) solvers (like MIDACO). SMT allows for more flexibility in constraint declaration compared to MIP/MILP solvers [6, 7, 27, 62]. We evaluate the use of the Z3 [16] SMT solver in this chapter, and empirically compare its performance against MIDACO and GA.

Similarly, Gosh et al. explore the use of the optimization in a Satisfiability Modulo Convex Optimizer (SMC) using the Z3 SMT solver for WSNs in [27]. In this prior work, the authors compare Z3 against the CPLEX MILP solver [12], for synthesizing "pop up" IoT networks which are temporary, short lived WSNs used during crisis events [21]. They target constraints for coverage, visibility, and connectivity. They also investigate how to scale the solvers to construct larger networks hierarchically, and perform power optimization by controlling the number of sleeps/wake up cycles. While we also explore topics on sensor placement and power optimization, our approach differs in both target application and placement formation. Our focus is on permanent deployments for remote and highly resource constrained settings, as well as on tools that help deployment operators size their battery and renewable power generation, while minimizing worst case energy across deployments.

For more permanent sensor network deployments, the authors of [20, 59, 66] optimize node placement using network coverage and connectivity and conserve energy by manipulating sleep/wake cycles (similar to Gosh). In [59], the authors use a hybrid approach that combines mathematical modeling and Genetic Algorithms (GA) to do so. In our work, we optimize energy use across an edge deployment by focusing on radio communication (distance and payload size) and use the solutions to guide configuration of energy-efficient edge deployments.

3.6 Chapter Summary

Finally, in this chapter, we present our methodology to design and optimize edge deployments for power consumption under two different power supply infrastructures, shared and individual. Here we focus on optimizing power consumption from communication using wireless networks, where bandwidth and signal power allocation between

sensors and a wireless access point determines energy efficient communication. We define a non-linear optimization problem that minimizes radio energy used by a set of sensors based on the amount of data transmitted by each sensor to a wireless access point and its distance from the access point. Additionally, we investigate three methods for solving this optimization problem and detail how the problem must be formulated to fit the requirements of each.

Our results illustrate specific heuristics and their properties, where for randomized scenarios commercial mixed-integer non-linear programming solver (MIDACO) is generally more effective than both an SMT solver (Z3 in this study) and a bespoke genetic algorithm. However, for a specific deployment, SMT can produce the best solution with the least amount of computational budget even though it is not possible to predict *a priori* whether it will produce any solution at all. We also find that for randomized deployments, infeasible solutions are often "close" to feasible ones. Thus, in a practical system, when a solver is unable to generate a feasible solution, it is possible to modify the best infeasible solution to produce a feasible one using only small perturbations.

This work illustrates the process of co-design that is necessary for remote IoT in which the sensing and actuation functionality, the networking functionality, the electrical power infrastructure, and spatial infrastructure placement must all be designed together. It also outlines the practical considerations associated with building tooling to facilitate this co-design process for edge deployments. However, this is still a simulation model, and need not be reflected by real edge systems. In the next chapters, we specifically explore and characterize how real edge device performance can deviate due to hardware-software interactions caused by shared power and network infrastructure.

Chapter 4

Characterizing Power-induced Variability in Real Edge Systems

4.1 Introduction

In the previous Chapter 3, we understand how to utilize simulation models for power optimization and cluster co-design, but in this chapter, we show how performance deviates from these models under realistic scenarios and characterize them. As edge systems prioritize affordability (often through long-term amortization) and compute density, these cluster deployments often utilize cheap, Linux-capable, single board computing devices like Raspberry Pis, Jetson Nanos, or BeagleBones, for programmability and rely on consumer-grade componentry for power delivery [23, 30, 42, 50, 72]. This is in contrast to datacenters where power delivery is carefully engineered and monitored. Moreover, these platforms dynamically scale voltage and frequency in response to power instability [80]. If the power delivery is variable or below operating specification, these systems exhibit significant performance reductions.

The simulation models do not take this power-induced performance variability into

account which is particularly problematic for edge clusters as it causes frequent deviations from model expectations. Edge clusters increasingly support time-sensitive distributed IoT applications, including sensor data aggregation, distributed analytics, and cyber-physical control loops, where predictable execution timing is required for synchronization and correctness. However, many IoT deployments operate in environments with aging, shared, or opportunistic power infrastructure and lack fine-grained power monitoring or management. As a result, operators and researchers lack tools to reason about how power-delivery affects observable application performance, complicating scheduling, capacity planning, and experimental reproducibility. Unsurprisingly, experimental evaluations in the literature frequently disable dynamic voltage and frequency scaling (DVFS) and other power management mechanisms to reduce performance unreliability and improve experimental repeatability [5, 33, 42, 51, 70], thereby masking power-induced performance effects intrinsic to many practicable deployments. A scalable methodology to characterize and predict power-induced performance variability is therefore essential for managing large-scale IoT and edge platforms.

In this chapter, we present a performance characterization of a large-scale, long-lived, Raspberry Pi 3B+ cluster (originally constructed in 2019 for remote deployment) with 451 devices, organized into 12 subclusters, with uneven device counts per subcluster, where each subcluster shares a commodity 60A/300W USB power supply. The Linux installed on the devices is configured for minimal power draw (i.e. all unused subsystems are powered down by the kernel) and each subcluster draws power within operational range of the individual Raspberry Pi 3B+ devices (1.4A - 2.5A and 4.63V - 5V [24, 49]). Each subcluster represents a *power domain*, analogous to a rack-level power domain in datacenters. As part of its operation, we observe systematic power-induced performance degradation as "CPU throttling" rather than a power-off shutdown. Specifically, when a device detects supplied voltage, through the USB interface, falls below a threshold,

the device reduces the system clock speed to a "safe" minimum speed to avoid electrical damage. As computational speed of a device (i.e. cycles per second) is proportional to the clock speed, a throttled device is slower than an unthrottled one.

We present (in Section 4.2) profiling measurements to generate a repeatable class of Pi personalities, and provide evidence supporting the investigation of a particular modeling approach. In Section 4.3 we detail how different models suggested by this approach "fit" the Pi personality data (using R^2 and Bayesian Information Criteria [65]), and compare their predictive power using a reduced, scalably collected, profile. Our results indicate only three representative power domains profiles (subclusters with the minimum, median, and maximum active and functional device counts) are sufficient to accurately predict performance for the remaining subclusters. Holdout validation shows the modeling approach achieves low prediction error while reducing profiling complexity from exponential in concurrency configurations to constant-factor sampling.

More generally, by characterizing power-induced unreliability and providing a scalable predictive methodology, this work contributes empirical insights and practical tools for managing large-scale commodity edge clusters. We present a large-scale empirical characterization of power-induced performance degradation for an aging Raspberry Pi cluster. Here we introduce a behavioral taxonomy of edge devices under shared power constraints, identifying three repeatable throttling classes (unthrottled, volatile, throttled). Furthermore, we explore a family of linear predictive models and develop a multivariate piecewise-linear model that predicts throttling operational regimes. And finally we propose and validate a sparse profiling methodology (with this model) to predict subcluster performance by profiling only three representative power domains, achieving near-optimal estimations with minimal measurement overhead, and representative profiling data and traces available at *https://github.com/MAYHEM-Lab/rpi_throttle_dataset*. Overall, our results indicate power infrastructure degradation should be treated as a first-class

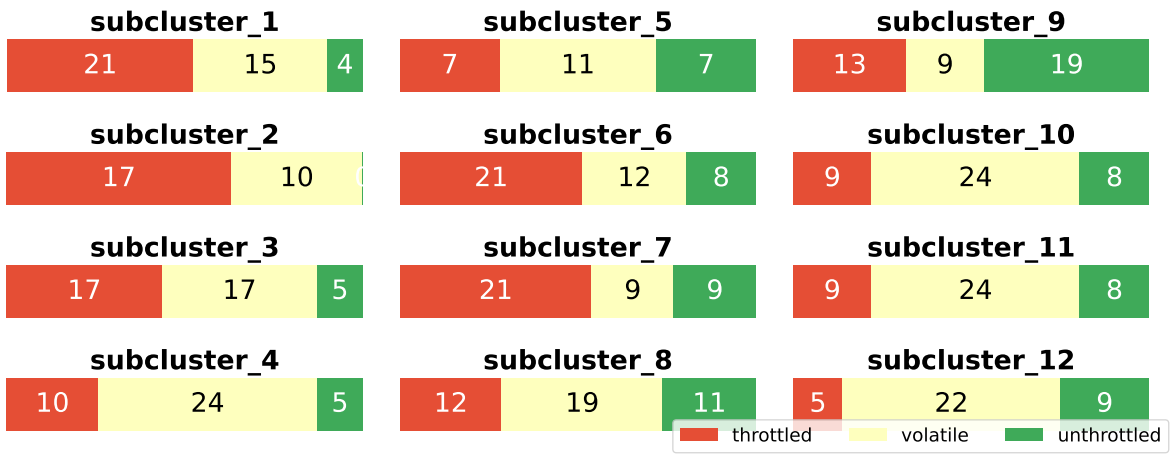

Figure 4.1: Pi Personalities per subcluster. Figure shows the device counts for each personality type (throttled, volatile, unthrottled) in the subclusters under study.

uncertainty source in distributed IoT systems, and scalable performance modeling is feasible even in large, low-cost edge clusters.

4.2 Pi Personalities

We begin by characterizing power-induced performance variability exhibited by the individual Raspberry Pi machines comprising a large-scale (and aging) cluster. We describe the cluster and power-domain layout, present our benchmarking methodology, and analyze how concurrency influences execution time and voltage throttling. From these measurements, we derive a taxonomy of device throttling behaviors and a methodology for categorizing the behaviors according to the taxonomy. This methodology provides the empirical foundation for the predictive models introduced in the final subsection.

We consider 451 Raspberry Pi 3B+ devices, a large scale cluster constructed in 2019 for edge deployment. Subclusters of devices share a commodity 60A/300W USB power supply; we consider each of 12 subclusters as a separate power domain. Each subcluster consists between 25 and 42 active working devices (failed devices are powered

off). Each Raspberry Pi has 4 cores at 1.4GHz, 1GB RAM, with operational range of current, 1.4A - 2.5A, and voltage, 4.63V - 5V [24,49]. The Raspberry Pi 3B+ dynamically imposes internal clock limits designed to slow down the CPU in an attempt to save the device from operating under potentially harmful conditions such as high temperatures or unstable power supplies. We focus on the latter in this chapter as the cluster is housed in a temperature-controlled setting, and observe no temperature triggered throttling events. To protect against unstable power delivery, the device automatically adjusts CPU frequency from 1.4GHz to 0.6GHz when the internally-measured CPU voltage is < 4.63V, with a $\pm 5\%$ error in measurement [49]. We monitor dynamic voltage and frequency scaling (DVFS) every second (on average) using `vcgencmd`, which reports voltage and frequency changes with corresponding throttling causes.

To profile each subcluster, we construct two compute-bound stress tests using a single, compute-bound microbenchmark. Using a compute-bound microbenchmark isolates the effects of CPU load on throttling behavior. We use 600x600 matrix multiplication (MM) written in C for the microbenchmark. MM is a compute-bound workload with predictable instruction and memory access patterns, enabling controlled characterization of voltage-induced clock frequency scaling. As such, it provides a measurement of application-delivered device performance usable by schedulers to allocate tasks under power-domain constraints caused by throttling. While real workloads may exhibit additional contention in memory (e.g. due to paging) and network subsystems, we have not observed voltage scaling triggered by such subsystems and leave exploring memory- and network-intensive workloads to future work. The *individual* stress test runs four MM microbenchmark jobs (one per core) 30 times on each device, one-at-a-time. During this test, the other functioning devices in the subcluster are powered on but idle. The second, *maximal*, stress test runs four MM jobs 30 times on all devices (running concurrently) in the same subcluster. We log DVFS throttling events as described above for both stress tests. To

capture the impact of throttling on execution performance (for scheduler use), we record the fraction of the total time each node spent in a throttled state (termed the *throttled fraction*), during the stress test. We compute the throttled fraction for each stress test as the number of seconds the throttle flag (logged with `vcgencmd`) is set during execution of four MM jobs on an individual device divided by the total time taken to finish the four MM jobs.

4.2.1 Device Behavior Under Shared Power

We use a simple taxonomy to categorize device throttling behavior under shared-power, referred to as **Pi personalities** The devices consistently fall into three categories: *throttled*, indicating power or hardware degradation sufficient to trigger frequency scaling with or without contention, *unthrottled*, indicating devices never exhibit throttling under maximal subcluster load, and *volatile* indicating devices that exhibit load-dependent throttling behavior (i.e. neither *throttled* nor *unthrottled*). Figure 4.1 depicts this taxonomy for the devices under study. For each subcluster, we specify the functioning device count and use red for throttled devices, yellow for volatile devices and green for unthrottled devices.

4.2.2 Throttling Onset Under Shared-Power Concurrency

This categorization motivates us to investigate the relation between subcluster load and the throttled fraction for each *volatile* device in a subcluster. To generate subcluster load, we choose, at random, a set of devices to induce "background" load while a *volatile* device is under study. We refer to the number of devices chosen for the background load (excluding the volatile device under test) as the *concurrency level* and measure throttled fraction for each *volatile* device as a function of increasing concurrency levels. As with the previous tests, we generate a sample of 30 throttled fractions for each combination of

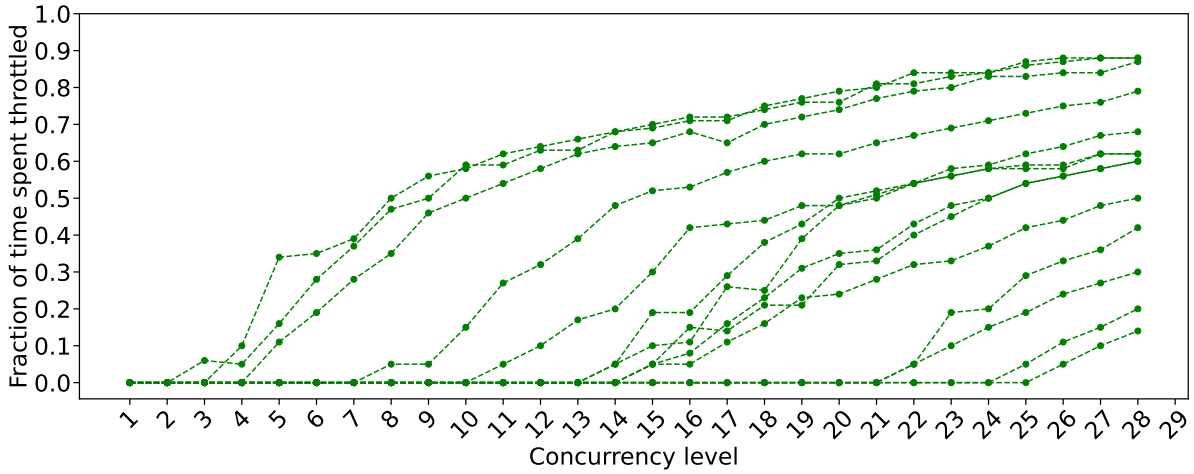

Figure 4.2: Fraction of time each node spent in throttled state (throttled fraction), as y-axis, and number of devices concurrently executing within a subcluster (concurrency level), as x-axis, for volatile devices in a representative subcluster (subcluster-8). Each line represents a device under test and each point corresponds to the number of concurrently running devices required to cause the corresponding observed median throttle fraction.

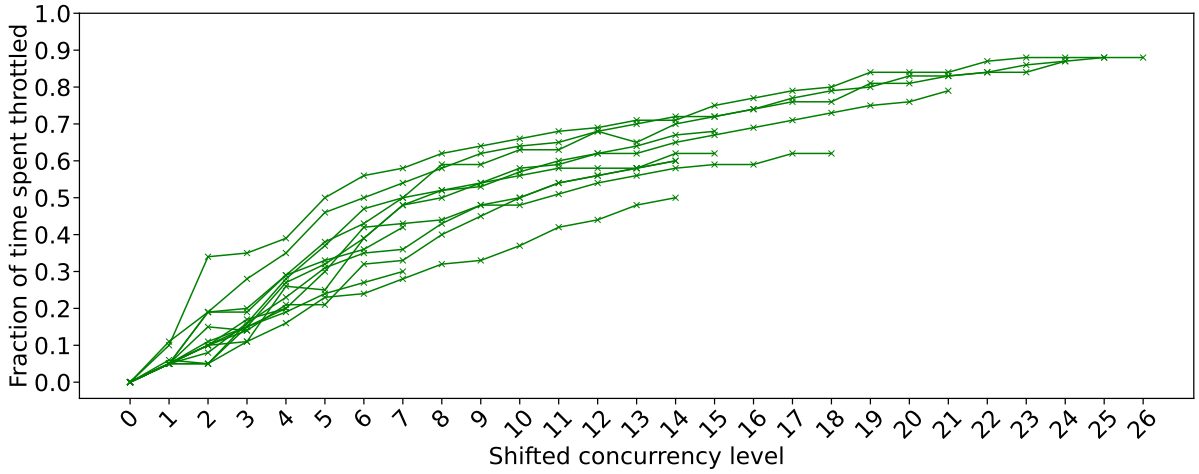

Figure 4.3: Throttled fraction relation from Figure 4.2, with zero-shifted (i.e. origin-crossing) lines. This figure illustrates that the linear patterns are regular across the different devices and could be considered to represent a common pattern of increasing throttle fraction with increasing concurrency level.

throttled device and concurrency level and summarize the relationship using the median of this sample (to filter any outliers that might occur due to operating system noise during a specific benchmark run).

The result of a background load test is a set of x-y pairs matching concurrency level (as an x value) with median observed throttled fraction (as a y value) for each *volatile* device in a subcluster. For example, Figure 4.2 plots the x-y pairs for all *volatile* devices in subcluster-8 (an arbitrarily chosen but representative subcluster). The plotted pairs, from the same device, are linked by lines, one per *volatile* device. The figure shows the minimum concurrency level at which throttling occurs differs by device, and the throttled fraction is monotonically increasing for all devices. These properties are consistent for background load tests of *volatile* devices across all subclusters.

If it is possible to conduct a background load test for every *volatile* device in every subcluster at every concurrency level, indexed by a *volatile* device identifier, a scheduler can use this information to determine the effective CPU compute rate for any combination of devices it selects (see Chapter 5). This scheduler could simply determine the Pi personality of each device it is considering, and for any assignment of devices to subclusters, use the maximal speed for *unthrottled* devices, the minimal speed for *throttled* devices, and a look-up of the throttled fraction (based on concurrency level) of the speeds for the *volatile* devices. This methodology, however, requires a full set of stress tests and background load tests for all devices in all subclusters.

To investigate improving the scalability of the approach, we explore throttled fraction trends across subclusters for any commonalities. We summarize the per-subcluster x-y pairs for the *volatile* machines into a *trajectory* for each subcluster. To form a subcluster trajectory, we first note that the plots shown in Figure 4.2 are approximately parallel, indicating the throttled fraction relationship is consistent and independent of the minimum concurrency level at which it begins to occur. Thus, to summarize all device trajectories

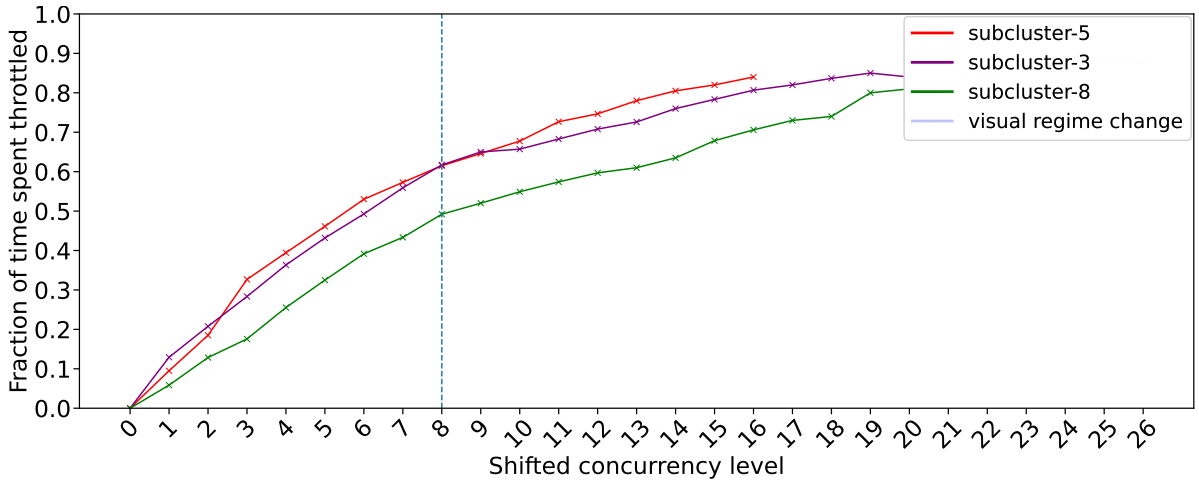

Figure 4.4: Subcluster trajectories for subcluster-3 (purple), subcluster-5 (red), and subcluster-8 (green). The vertical line represents a visual indicator for a slope shift in the line later used to indicate a regime change for regression models in Section 4.2.3.

in a subcluster into a single trajectory, we first transform (termed "zero-shift") each device trajectory to intersect the origin, where the sample x-y pairs in Figure 4.2 of the devices now roughly overlap with each other as seen in Figure 4.3, i.e. by subtracting the maximum concurrency level for which no throttling occurs from the x-axis value on a per-device basis. We then compute the median throttled fraction over these zero-shifted, overlapping points, per concurrency level to construct the subcluster's trajectory. Figure 4.4 shows this trend-comparing representation across multiple subclusters (specifically subclusters 3, 5, 8).

As examples, Figure 4.4 shows three different subcluster trajectories. Note, they differ in terms of active and functioning devices. Subcluster-5 has 25 functioning devices, subcluster-3 has 39 functioning devices, and subcluster-8 has 42 functioning devices. From the figure, subcluster trajectories also appear to depend on the number of active devices drawing power from the shared power distribution unit for the subcluster. Further (as indicated by the vertical line) the relationship appears to be different for concurrency levels < 8 compared to ≥ 8.

4.2.3 Linear Models of Subcluster Throttled Fraction

Based on this graphical analysis of subcluster trajectories, we hypothesize the overall relationship between throttled fraction, concurrency level, and active device count (for *volatile* devices) is piece-wise linear. Thus, in the remainder of this study, we explore three different linear models for describing each subcluster's throttled fractions. As a baseline, we consider ordinary least squares (OLS). We also consider a piece-wise linear model (PW) with two segments. Both models use the zero-shifted concurrency level as their only explanatory variable. Finally, we consider a multivariate linear model (MPW) with two segments that takes both concurrency level and number of active, functioning devices as explanatory variables. Note these models are fitted for a subcluster using all the profiled *volatile* device lines for that subcluster. The subcluster trajectories described in the previous subsection only motivate the use of our linear predictive models. While more sophisticated, non-linear models (with higher fitting costs and potentially greater sensitivity to measurement variability) may also be appropriate. In Section 4.3 we show the effectiveness of the linear approach and, from these results, conclude additional predictive power may not be necessary. We also explore root-cause analysis for these patterns and explain why our data-driven approach can be more widely applicable in Section 4.4.

To evaluate how well these models fit with our subcluster behavior we compare their R^2 and Bayesian Information Criterion (BIC) score. Table 4.1 shows the BIC score for the entire subcluster dataset, and the min, max, mean R^2 score (across all subclusters) when using OLS, PW, and MPW models to predict the throttled fraction across all 12 subclusters. To elaborate, each model is fitted using every subcluster's profile data and BIC uses the negative log-likelihood as a goodness-of-fit measure (smaller is better), adding a penalty term (to penalize overfitting) for the number of parameters to estimate (scaled

Full-set trained model	R^2 score			BIC score (on full subcluster set)
	min	max	mean	
OLS	0.50	0.88	0.77	-7326.59
PW	0.67	0.94	0.87	-8305.78
MPW	0.76	0.95	0.89	-8415.99

Table 4.1: Range of R^2 (higher is better) scores across all 12 subclusters and BIC (lower is better) for the full subcluster set, using predicted throttled fractions from regression models fitted using every subcluster. MPW has best R^2 and BIC entries.

by the log of sample size). From the table, the R^2 scores for each linear model increase with model complexity, indicating the additional parameters are adding explanatory power. Note, the OLS model requires two estimated parameters, the PW model requires four estimated parameters, and the MPW model requires six estimated parameters. Further, the overall BIC score for the most complex model (MPW) is lowest, indicating this additional explanatory power does not come at the expense of over fitting. Also note, the data in rows 2 and 3 of Table 4.1 is when PW, MPW use concurrency level 8 to switch between their piecewise segments (see Figure 4.4). Rather than estimate this point of regime change as an additional parameter, we compute the R^2 and BIC scores exhaustively for all possible concurrency levels (as change point) in all subclusters. We observe the highest average R^2 and overall lowest BIC score for concurrency-level 8

MPW's lead in terms of fit confirms the observations (illustrated in Figure 4.4) that the number of active devices adds profitable explanatory power. That is, MPW has the highest mean R^2, BIC and the narrowest range of R^2 scores, showing the effectiveness of the additional parameter, with better and consistent fit across all subclusters. For completeness, we present the derived model equations in Table 4.2. Based on these models

Model	Equation derived from full cluster as training data
OLS	$y = 0.112 + 0.040 * x$
PW	$y = (x < 8)[0.011 + 0.071 * x] +$ $(x \geq 8)[0.375 + 0.022 * x]$
MPW	$y = (x < 8)[0.167 - 0.004 * \text{avail\_devs} + 0.071 * x] +$ $(x \geq 8)[0.645 + 0.007 * \text{avail\_devs} - 0.023 * x]$

Table 4.2: Derived regression equations using every subcluster as training data (rounded to three decimal places). y is predicted throttled fraction, x is the concurrency level, and avail_devs is the total number of functioning devices in a subcluster.

and their reasoning we now run our experiments in Section 4.3 to determine which model is best in terms of prediction error, and investigate reducing the data (and thus profiling) required to achieve good prediction results.

4.3 Empirical Evaluation

In the previous section, we evaluated three candidate models: ordinary least squares (OLS) using concurrency as the predictor, piecewise linear (PW) with two regimes, and multivariate piecewise linear (MPW) incorporating the two regimes, concurrency level, and power-domain size. We used R^2 to quantify variance explained and BIC to select among candidate models while penalizing complexity, showing that piecewise linear models provide an accurate and parsimonious representation of concurrency-dependent throttling behavior. In this section we aim to answer the following research questions:

- RQ1: Do the regression model scores from Table 4.1 correspond to prediction accuracy for the subclusters?
- RQ2: Is there a representative set of subclusters the regression models can train on to preserve prediction accuracy while minimizing profiling overhead?

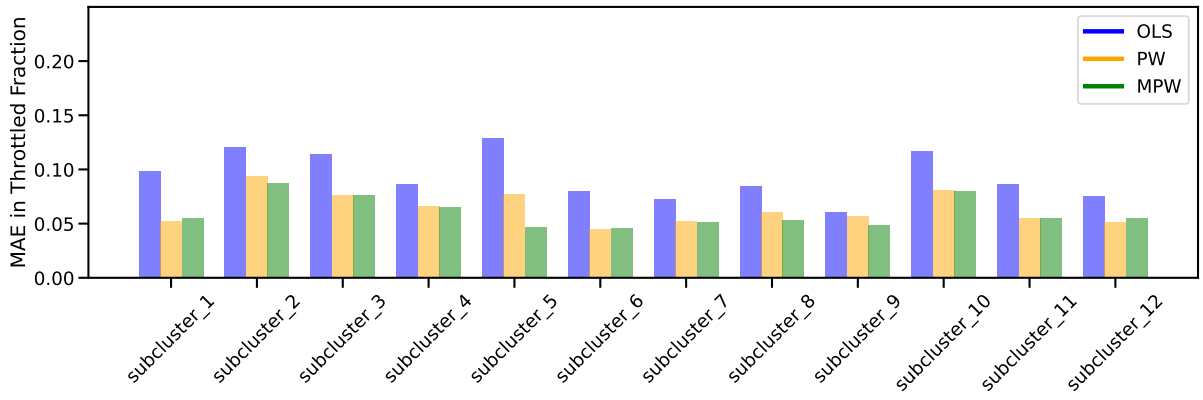

Figure 4.5: MAE for each subcluster. MAE is computed as the absolute error of the difference between the predicted and actual throttled fraction, on average across experiments. Models fitted on exhaustive profiles from all 12 subclusters. Overall MPW reaches the lowest errors across all subclusters.

Model trained on full-set	RMSE			MAE		
	Min	Max	Range	Min	Max	Range
OLS	0.074	0.150	0.076	0.060	0.129	0.069
PW	0.057	0.121	0.064	0.045	0.094	0.049
MPW	0.059	0.109	0.050	0.046	0.087	0.041

Table 4.3: Range of RMSE and MAE across subclusters. MPW has the narrowest error range (indicating stability) and lowest maximum RMSE and MAE. Models fitted on exhaustive profiles from all 12 subclusters.

We evaluate the prediction accuracy of the models using root mean squared error (RMSE) and mean absolute error (MAE), when trained on the full and exhaustive profiles of all 12 subclusters.

We specifically look at which model approach provides the lowest aggregate error and stable performance across all subclusters. Figure 4.5 compares the MAE for the exhaustive profile across all three regression methods. The x-axis represents each subcluster and the y-axis is MAE for the throttled fraction, (i.e. the fraction of time spent throttled during execution). Each bar represents the error calculated using predicted y values from each model. Blue bars are for OLS, yellow for PW, and green for MPW. These errors represent

the MAE achievable by each linear model when applied to all the data. Note, we omit graphing RMSE here because the ranking comparison is identical. Because throttled fraction errors are small, we also present the minimum and maximum RMSE and MAE in Table 4.3 to show the range of errors across the subclusters.

Over the 12 subclusters, MPW achieves the lowest aggregate error most often, and exhibits the most consistent prediction performance, while OLS performs the worst. Straight-line OLS regression assumes a single linear relationship between concurrency and throttling, and therefore cannot capture regime changes introduced by the increase in the number of volatile devices (devices which throttle dependent on background load), as explained in Section 4.2.2. PW and MPW explicitly capture these regime shifts, and MPW further incorporates power-domain capacity draw, (i.e. the number of active devices) leading to substantially lower aggregate error in most domains. PW slightly outperforms MPW on four of the twelve domains($\sim 0.001 - 0.002$ RMSE, MAE for throttled fraction). But across all domains, MPW yields a tighter error range ($max - min$) for both RMSE and MAE, indicating more robust error performance under domain-to-domain variability.

The tighter error bounds for MPW suggest incorporating power-domain size reduces cross-domain variance by capturing systematic differences in shared-power capacity, improving overall prediction power. The domains where PW marginally outperforms MPW likely reflect cases where power-domain size provides limited additional signal (e.g. cabling losses or atypical device composition) or where the dominant effect is already captured by the regime split at the change point. In such domains, the additional covariate may introduce slight prediction variance without improving fit. Nevertheless, the overall reduction in error dispersion, better R^2 and BIC scores from Table 4.1 indicates MPW generalizes more reliably across heterogeneous power domains, answering RQ1.

4.3.1 Sparse Profiling for Scalable Performance Prediction

Although exhaustive profiling provides accurate characterization, it is impractical for large-scale edge clusters due to the cost of profiling all devices and concurrency configurations. Therefore, we investigate whether sparse profiling is sufficient for accurate performance prediction. We hypothesize that profiling three representative subclusters, those with the minimum, median, and maximum active device counts, is sufficient to achieve acceptable prediction error performance across the cluster. This hypothesis is tested by fitting models to *all* combinations of three subclusters and comparing predictive accuracy against all subclusters individually. We then compare the model prediction error performance for the combination of the subcluster with the minimum, maximum, and median number of active devices to the globally best model prediction errors among all three-set combinations. We refer to this latter combination as the *Min-Med-Max holdout strategy*. Our intuition for the hypothesis of combining the minimum, maximum, and median active device subclusters being sufficient as these three accurately capture the range of active power draw to throttle fraction relationships. That is, selecting domains spanning the observed active capacity range provides fitting data for the full spectrum of subcluster sizes, enabling the model to generalize across the device count varied subclusters.

To evaluate our sparse profiling hypothesis (and the Min-Med-Max holdout strategy), we show the MAE when we use sparse profiling to predict the throttled fraction for each subcluster in Figure 4.6. The graph layout is the same as Figure 4.5. To provide a visual comparison of this strategy against the performance of other three-domain combinations, we overlay each bar in the figure with a red dashed line. The red line denotes the minimum and maximum MAE observed *across all three-domain combinations*. The red X marks the global minimum error across combinations. From the figure, the height of the MPW

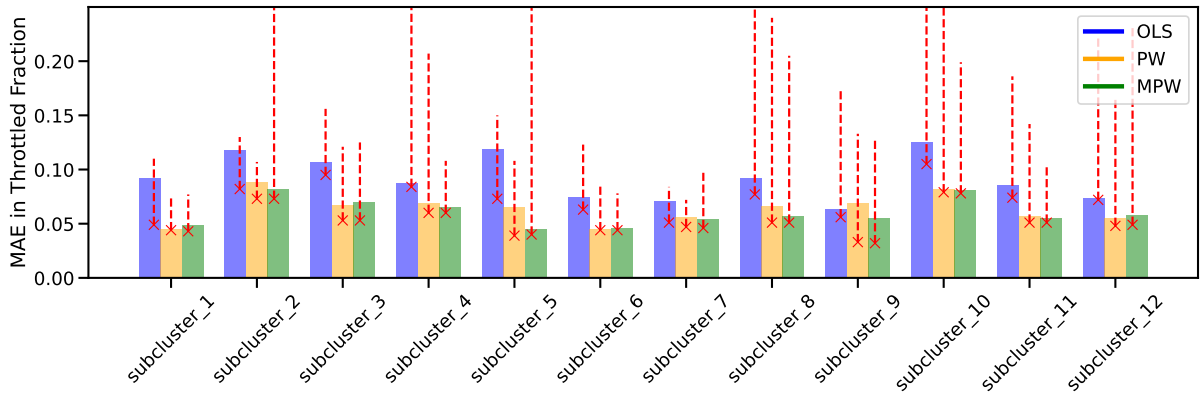

Figure 4.6: MAE for each subcluster for the Min-Med-Max strategy. The red line is the minimum, maximum error across all three-domain models. The Min-Med-Max set achieves prediction error close to global minimum (red-X) for all models. Moreover, MPW's MAE for this set is close to optimal (i.e. the MAE shown Figure 4.5 when we exhaustively profile all subclusters).

bar is equivalent to, or slightly above the global minimum (the red X) for each subcluster. Thus, the MPW model trained on Min-Med-Max to predict throttled fraction in *all other* subclusters achieves an MAE similar to the best performing MPW model out of *all three-domain training combinations*. As before, the graphs for RMSE are similar and omitted due to space constraints. Note, the sporadically large heights of the dashed lines in the figure indicates model performance is sensitive to the three subclusters used for model fit. This shows model prediction accuracy is dramatically impacted by the choice of three-subcluster set.

Table 4.4 summarizes Figure 4.6 and reports the error gap relative to the global minimum error. We show the difference (in columns 2 and 4) between the global minimum and the Min-Med-Max strategy minimum for RMSE and MAE, respectively. The data shows the Min-Med-Max strategy achieves prediction error very close to the globally best-performing three-domain training combination across all models. This result suggests selecting fitting domains that span the observed active capacity range is sufficient to capture the dominant sources of cross-domain variability and to enable accurate performance

Model	RMSE		MAE	
	Difference between min(Min-Med-Max) and min(Across all sets)	Range (Min-Med-Max)	Difference between min(Min-Med-Max) and min(Across all sets)	Range (Min-Med-Max)
OLS	0.018	0.077	0.014	0.062
PW	0.010	0.056	0.012	0.043
MPW	0.013	0.054	0.013	0.037

Table 4.4: RMSE and MAE statistics for the holdout experiments for the three domain combinations. Columns 2 and 4 shows the difference between the global minimum and the Min-Med-Max strategy minimum for RMSE and MAE. Columns 3 and 5 show the range ($max - min$) for RMSE and MAE, for the Min-Med-Max strategy. Min-Med-Max strategy achieves prediction error close to the best-performing three-domain training combinations across all models. And MPW's range indicates stability in performance under subcluster variability.

prediction with sparse profiling. Columns 3 and 5 of the table show the RMSE and MAE range ($max - min$) for the Min-Med-Max strategy. MPW has a narrow range of errors for both RMSE and MAE, indicating stability in performance under subcluster variability. Note, this range is also similar to or better than RMSE and MAE range shown in Table 4.3 – when we exhaustively profile all 12 subclusters. This shows Min-Med-Max's performance, with significantly less profiling, is very close to optimal.

For completeness, we report the regression coefficients learned from this training set in Table 4.5 and summarize the R^2 and BIC scores in Table 4.6. The R^2 and BIC scores are similar to or better than those obtained with the full training set (in Table 4.1), indicating no degradation in explanatory power or model parsimony when restricting the training data to these three representative domains. Moreover, MPW again has the best R^2 and BIC scores. PW and MPW, with their additional parameters, also outperform OLS using this strategy indicating better capture of throttled fraction across power-domains. Overall, these results show MPW performing best under both full-set and Min-Med-Max fitting strategies, indicating power-domain active capacity is a useful explanatory variable

Model	Equation derived using the Min-Med-Max profiling strategy
OLS	$y = 0.121 + 0.041x$
PW	$y = (x < 8)[0.014 + 0.075x] +$ $(x \geq 8)[0.406 + 0.020x]$
MPW	$y = (x < 8)[0.157 - 0.004\,\text{avail\_devs} + 0.075x] +$ $(x \geq 8)[0.667 - 0.008\,\text{avail\_devs} + 0.023x]$

Table 4.5: Derived regression equations for the Min-Med-Max strategy (rounded to three decimal places). y is the predicted throttled fraction, x is the concurrency level, and avail_devs is the total number of functioning devices in a subcluster.

Min-Med-Max trained model	R^2 score			BIC score (on full subcluster set)
	min	max	mean	
OLS	0.53	0.87	0.77	-7298.08
PW	0.73	0.94	0.87	-8244.28
MPW	0.79	0.95	0.89	-8383.22

Table 4.6: R^2 and BIC scores calculated using predicted throttled fraction for Min-Med-Max strategy across subclusters. Values comparable to exhaustively profiled model scores in Table 4.1.

and profiling the minimum, median, and maximum capacity domains provides sufficient coverage for accurate and scalable cluster-wide performance prediction, answering RQ2.

Finally, this study also illustrates a general methodology applicable to similar edge clusters with shared power domains. To use it, a cluster administrator runs individual and maximal stress tests (Section 4.2) to classify devices within each cluster (Figure 4.1). Second, they profile the volatile devices in subclusters with the minimum, median, and maximum number of active devices using the background load test (Section 4.2.2). Finally, they construct an MPW model from this profiled data (i.e. the zero-shifted throttled fraction data and regime change concurrency level either visually or empirically derived).

Note, a PW model is equivalent when all subclusters have the same number of active devices. The resulting model predicts the throttled fraction for any subcluster given a concurrency level and active device count. Moreover, combining this prediction with the MM job's gigaflop rate for a subcluster yields a lower bound on subcluster compute performance which can be used to inform scheduling decisions.

4.4 Discussion

Our characterization and modeling methodologies target a class of edge and IoT deployments with specific, but widely encountered, operational characteristics. In particular, we consider deployments composed of off-the-shelf hardware components, including devices, cabling, USB hubs, and power distribution units, that are used over extended periods without modification to their lifecycle. These systems are typically purpose-built for long-lived operation in technology-hostile environments, where routine maintenance and frequent component replacement are impractical due to cost, access constraints, or deployment scale.

While these assumptions may appear restrictive, they are in fact common across many real-world IoT and edge deployments, particularly in outdoor and industrial settings. Such clusters are often designed to prioritize affordability and sustainability, favoring commodity hardware potentially purchased second-hand or amortized over long operational lifetimes [53,69]. As a result, aging and heterogeneous devices are the norm rather than the exception. The cluster studied in this work reflects this increasingly prevalent deployment model.

In some environments, devices are replaced when their operational "personality" degrades beyond acceptable bounds. However, the applications we target, such as wildlife conservation, forest fire monitoring, and digital agriculture, frequently operate in remote

locations where subclusters are sealed to mitigate environmental exposure [4, 44]. In these settings, replacing individual devices becomes increasingly costly as cluster size grows, and system components must often operate in a degraded state until replacement is economically feasible. Our methodology explicitly accommodates such gradual degradation rather than assuming frequent hardware refresh.

Although device behavior could, in principle, evolve beyond the three personality classes identified in this study, our empirical evidence suggests such changes are rare in practice. We have collected over a year of intensive stress-test data and have not observed additional degradation modes beyond those captured by our taxonomy. Should devices simply shuffle between existing behaviors, our profiling methodology can be re-executed to update the system composition. We continue to monitor the cluster to track long-term behavioral trends.

Finally, we explored potential root causes of power-induced throttling, including device aging, power supply variability, and wiring effects. Our experiments with swapping cables, ports, power supplies, and devices yielded inconclusive results. Similar power-induced performance variability has been reported in other edge deployments (see Section 4.5) but operators mitigate the problem by overclocking devices or supplying elevated voltages, accelerating hardware degradation. Importantly, our approach does not depend on isolating a root cause. Instead, it captures the observable effects of power-induced variability and incorporates them into predictive models. This design choice allows our approach to scale and remain effective even when underlying causes are difficult to diagnose precisely.

Taken together, our results shows explicitly modeling power-induced variability, rather than attempting to eliminate it, provides a practical and robust foundation for performance prediction in long-lived, resource-constrained edge deployments.

4.5 Related Work

Predicting system performance has been studied in the context of datacenter scheduling, cloud resource management, and edge orchestration [8, 57, 82]. Regression models, piecewise-linear models, and machine learning-based predictors have been used to estimate execution time and resource contention effects [8, 9, 57, 70]. These approaches assume homogeneous and stable infrastructure or require exhaustive profiling of system configurations [29]. In contrast, our work develops a multivariate piecewise-linear model that captures concurrency-dependent throttling regimes in degraded shared-power domains and demonstrates accurate prediction with sparse profiling of only a small subset of power domains.

To the best of our knowledge, this work is the first to empirically characterize power-induced performance degradation in a large-scale edge cluster and to provide a scalable predictive modeling methodology for degraded shared-power domains. We next present our taxonomy and characterization methodology.

4.6 Chapter Summary

In this chapter, we present a large-scale experimental characterization of aging edge devices and quantify power-induced performance degradation in a shared-power Raspberry Pi cluster. We identify three repeatable device behavior classes, *throttled*, *unthrottled*, and *volatile*, that capture persistent and concurrency-dependent throttling behavior. We develop and validate predictive performance models, showing multivariate piecewise linear models provide the best fit across shared power domains. Furthermore, we propose and validate a sparse profiling methodology, where profiling only the minimum, median, and maximum capacity power domains yields predictive accuracy comparable to profiling all

subclusters. In the next chapter, this characterization is used to build an end-to-end scheduler that adapts to these changing power availabilities to optimize performance and energy efficiency of Machine Learning (ML) workloads.

Chapter 5

End-to-End Scheduling for Dynamic Resource Conditions at the Edge

5.1 Introduction

After understanding how performance can be significantly impacted by the evolving resource contention in remote edge environments, in this chapter we focus on utilizing these properties to design PLUMBER an adaptive scheduler for dynamic resource conditions in edge environments. Specifically we focus on running Machine Learning (ML) and Artificial Intelligence (AI) inference workloads at the edge. Advances in machine learning (ML) have enabled increasingly sophisticated inference workloads to execute directly on resource-constrained edge devices, reducing latency, communication cost, dependence on reliable high-performance wide area network connectivity, and the cloud. Consequently, edge and Internet of Things (IoT) applications increasingly perform inference locally rather than relying on cloud offloading [78]. However, implementing efficient inference in these environments remains challenging even when the edge-based inference algorithms are tuned for resource constrained execution at the edge. Specifically, because edge devices

typically are not highly engineered resources with reliable power sources and vast internal memory systems, their available resource capacity can fluctuate (often dramatically) over time.

That is, unlike static data centers, edge deployments are characterized by unpredictable network bandwidth and compute variability caused by contention for "scarce" shared resources [47, 54]. In these dynamic environments, "one-size-fits-all" deployment strategies do not maximize delivered application performance. The best pipeline decomposition (in terms of performance) often depends on current resource conditions: balancing computation is advantageous when bandwidth is plentiful, whereas minimizing communication becomes increasingly important as bandwidth becomes constrained. Because these conditions continually change, no single decomposition strategy performs well across all deployment scenarios. Adaptive scheduling is therefore required to continuously select the decomposition best suited to the current operating environment.

To address this challenge, we present PLUMBER, an adaptive scheduler for distributed deep learning (DL) inference on resource-constrained edge clusters. Rather than relying on a fixed deployment configuration, PLUMBER dynamically selects pipeline decompositions based on current network conditions, background load, workload characteristics, and request arrival patterns. Overall we develop an online scheduling framework that selects between computation- and communication-optimized pipeline decompositions using a pipeline-aware performance model, reducing profiling overhead by more than 95%. Furthermore, we introduce a "workload compaction" strategy that maximizes workload density per joule by dynamically balancing and consolidating workloads across devices, reducing both idle and active power consumption. And finally, we evaluate PLUMBER on a 117-node Raspberry Pi 3B+ cluster using EfficientNetB0 [71], ResNet18 [36], and ViT-B16 [19], representing efficient convolutional, residual convolutional, and transformer-based architectures. Across realistic Azure Functions workload traces [61] and network

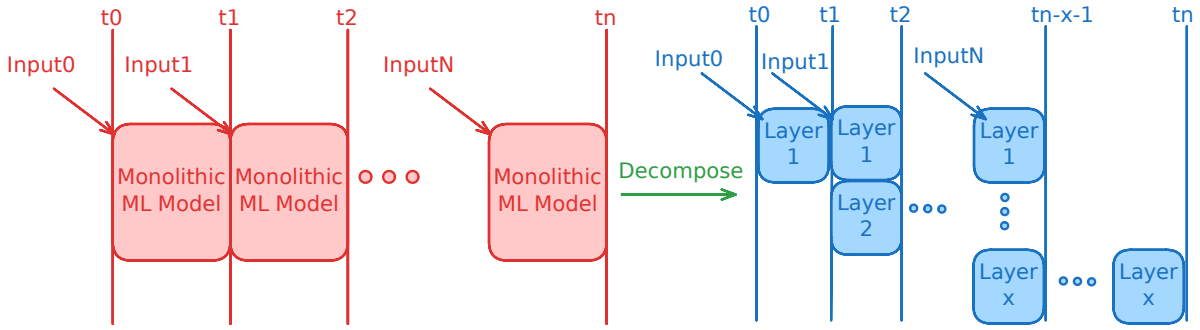

Figure 5.1: A monolithic DL model is partitioned into sequential pipeline stages that execute on separate edge devices. Different pipeline decompositions correspond to different layer partitioning serving as candidate schedules by PLUMBER.

bandwidths ranging from 2 to 240 Mbps, PLUMBER consistently improves scheduling and energy efficiency over static decomposition.

5.2 PLUMBER Architecture

PLUMBER is an online scheduler for distributed DL inference that dynamically selects pipeline decompositions based on predicted execution costs under current resource conditions. It partitions DL models into sequential stages [52, 83] to distribute inference and allow multiple inference requests to be processed simultaneously, as shown in Figure 5.1. For each arriving job, PLUMBER evaluates candidate decompositions by estimating compute throttling, network contention, and device utilization, ultimately selecting the schedule to maximize throughput while minimizing overall energy usage.

PLUMBER assumes these DL models are decomposable across multiple edge devices, and through this limits the number of edge devices used, i.e. limits the contention for shared resources. PLUMBER partitions DL models at layer boundaries, allowing it to evaluate alternative organizations online without modifying the underlying model representation. To do so, PLUMBER maintains state information for every device in the cluster (e.g., assigned job and estimated execution progress). This allows PLUMBER to

estimate background load, emulate network bandwidth sharing, and predict performance degradation caused by power-driven CPU throttling due to shared power infrastructure.

As formalized in Algorithm 1, PLUMBER evaluates each candidate decomposition θ using three metrics: local throughput (th_{local}), global cluster throughput (th_{global}), and utilization ($util$). We define $util$ as a proxy for energy consumption, based on the time devices spend at full 1.4GHz clock speed (T_{active}) versus throttled or idle states, at 0.6GHz, (T_{idle}). Given power levels of P_{active} ≈5.1W at 1.4GHz and P_{idle} ≈1.9W at 0.6GHz [26]. The energy used by a computation is then $\mathcal{E} = P_{active} * T_{active} + P_{idle} * T_{idle}$. Here T_{active} and T_{idle} are determined using *vcgencmd* performance counters.

Algorithm 1 Dynamic Pipeline Scheduler (PLUMBER)

Input: New job J (Model, N inputs, arrival time t)

Output: Optimal decomposition parameters θ^* and execution delay d^*

$valid\_delays \leftarrow [0] \cup \{\text{Active task times} > t\}$
$\Theta \leftarrow \{\text{CandidateDecompositions(J, N)}\}$
for each valid decomposition $\theta \in \Theta$ **do**
    for each valid delay $d \in valid\_delays$ **do**
        $cluster \leftarrow \text{BackgroundLoadInfo}(d)$
        $(T_{active}, T_{idle}) \leftarrow \text{Simulate}(\theta, J, cluster)$
        $util \leftarrow T_{active} + T_{idle}$
        $th_{local} \leftarrow \text{Throughput}(N, J)$
        $th_{global} \leftarrow \text{Throughput}(N, cluster)$
    end for
end for
$\theta^*, d^* \leftarrow \arg\max_{\theta,d} (\text{Score}(-util, th_{global}, th_{local}))$
return $\{\theta^*, d^*\}$

5.2.1 Estimating Throttling Overhead

To quantify instability caused by CPU throttling due to uneven power distribution from shared power supplies, we directly use our findings from chapter 4. The PW regression equation is used to predict the throttling factor tf based on the number of devices concurrently running computations (background load) on the shared power supply. We define tf as the ratio of time spent under CPU throttling ($T_{throttled}$), measured via hardware counters (`vcgencmd`), to total execution time (T_{total}): $tf = T_{throttled}/T_{total}$

The model is used to calculate the overhead added due to the conditional throttling, assuming operations across DL and matrix multiply are proportional. That is, to avoid profiling every possible DL layer combination under varying throttling states, we assume tf for a DL model's forward pass is proportional to the matrix multiplication workloads used to derive the PW model, given the majority of their computation is FLoating Point Operations (FLOPs). We define T_{best} as the "best-case" latency—the execution time when no throttling occurs. When throttling commences, CPU frequency drops from 1.4 GHz to 0.6 GHz, reducing computational throughput by approximately 57%. This implies the additional penalty period (T_{added}) is roughly half of the measured throttled time: $T_{throttled} \approx 2 \cdot T_{added}$. That is, the additional overhead T_{added} relative to T_{best} runs at approximately $\frac{1}{2}$ the clock speed so it can be estimated by dividing $T_{throttled}$ by 2. We have empirically verified this relationship during cluster profiling. Given that $T_{total} = T_{best} + T_{added}$, we can express the total runtime in terms of only T_{best} and tf:

$$T_{total} = T_{best} \left(\frac{2}{2 - tf} \right) \tag{5.1}$$

This derivation allows us to estimate the runtime for any arbitrary model split using only its baseline latency (T_{best}) and the current environmental tf (from the PW model), bypassing the need for exhaustive re-profiling.

5.2.2 DL Model Decomposition and Profiling

With the above assumptions and reduced profiling overhead, we can attempt to support online scheduling easier for candidate pipeline decompositions under changing network conditions and compute contention. To briefly quantify why exhaustively profiling every possible layer partition is impractical for realistic deep learning models we provide an example, using the ViT-B16 model. By limiting its decomposition to only three partitions, each with a minimum of 10% FLOP allocations per stage, it still yields more than 2,500 candidate decompositions for each available network bandwidth state.

To make online scheduling tractable, PLUMBER restricts candidate decompositions using three constraints. First, each pipeline stage must fit within the available device memory (we use <600MB in our study). Second, partition boundaries must preserve simple tensor interfaces to avoid complex model rewiring or developer intervention. Finally, we restrict pipelines to have between three and six stages. Fewer than three stages exceeds memory limits for large models like ViT-B16, whereas more than six stages produce partitions whose communication overhead outweighs any additional parallelism in ResNet18 and EfficientNetB0. Given these constraints, PLUMBER evaluates two complementary decomposition strategies:

- **Compute-Balanced (COMP SPLIT):** Distributes FLOPs evenly across layers to balance the computational load.
- **Communication-Optimized (N/W SPLIT):** Selects partitions to minimize the size of transferred tensors, reducing communication overhead.

All pipelines execute synchronously, eliminating intermediate buffering and providing predictable memory consumption within device limitations. Under synchronous execution, pipeline throughput is limited by the slowest stage, making highly unbalanced

decompositions undesirable. Thus, PLUMBER only considers these two decomposition strategies.

These constraints reduce the profiling space from more than 2,500 candidate decompositions to only 108 baseline measurements, i.e., 3 models $\times$ 18 decompositions $\times$ 2 strategies, a reduction of $>$95%. We collect these baseline latencies during warm-up and combine them with the analytical performance model to evaluate candidate schedules online.

5.3 Experimental Methodology and Evaluation

We evaluate PLUMBER under realistic edge deployment conditions to answer four research questions.

- RQ1: Are the performance models used by PLUMBER sufficiently accurate to support online scheduling?
- RQ2: Does PLUMBER improve end-to-end scheduling performance relative to static decomposition strategies across diverse workloads and operating conditions?
- RQ3: How do environmental conditions, including network bandwidth, background load, and model architecture, influence the scheduling decisions made by PLUMBER?
- RQ4: How robust is PLUMBER to changes in workload composition and arrival patterns?

To answer these questions, we describe our experiments, hardware platform, workloads, baselines, and evaluation metrics.

5.3.1 Experimental Setup

We evaluate PLUMBER on a 117-node Raspberry Pi 3B+ cluster under varying network bandwidth, background load, workload composition, and arrival patterns. Each device has a 4×1.4GHz ARM processor, 1GB memory, and communicates over Ethernet with a maximum available bandwidth of 240 Mbps. The devices are organized into three subclusters of 39 devices, where each subcluster shares a common power distribution unit (PDU). This shared power infrastructure results in power-induced CPU throttling for devices when executing compute-intensive workloads concurrently. To emulate a range of edge deployment environments, we limit available bandwidth to 240, 30, 10, and 2 Mbps, and equally share available bandwidth among simultaneously communicating devices.

The performance, and adaptability of PLUMBER is measured by testing it against realistic workload traces, varying compositions and varying computation, communication loads from different DL model tasks. We evaluate PLUMBER using three representative deep learning image classification models: EfficientNetB0, ResNet18, and ViT-B16. Models are exported to ONNX [17], quantized, and partitioned into inference pipelines of three to six stages according to decomposition constraints described in Section 5.2. Each pipeline processes batches of ten 1 MB images.

To capture both regular and bursty edge workloads, we evaluate two arrival processes. We implement a Uniform arrival process representing periodic sensing applications with approximately constant inter-arrival times, and we use inter-arrival times from an Azure Functions trace to represent bursty serverless workloads exhibiting realistic temporal variability. Unless otherwise noted, workload composition is uniformly distributed across the three models.

We compare PLUMBER against two static decomposition strategies. *Only Compute*, which always selects the compute-balanced decomposition (COMP SPLIT), while *Only*

Communication always selects the communication-optimized decomposition (N/W SPLIT). In contrast, PLUMBER evaluates both strategies for each arriving job and dynamically selects the decomposition predicted to maximize overall scheduling and energy efficiency given current cluster conditions. All three schedulers use the same underlying execution framework described in Section 5.2 and differ only in their strategy selection.

We evaluate scheduler performance using two complementary metrics, Scheduling Efficiency (η) measures achieved throughput relative to an ideal non-throttling schedule. Relative Energy Efficiency ($\mathcal{E}_{rel}$) estimates energy relative to the same, ideal non-throttling, schedule. As we normalize both metrics using the ideal (unthrottled) execution, they are unitless and thus, can be combined to capture a benefit-to-cost ratio (see Table 5.2). Across our experiments, we vary four independent variables: (1) available network bandwidth (2, 10, 30, and 240 Mbps), (2) background load (1% and 50%), (3) workload arrival process (Uniform and Azure Functions), and (4) workload composition. These variables represent the dominant sources of variability encountered in resource-constrained edge deployments and allow us to evaluate PLUMBER across a broad spectrum of operating conditions.

5.3.2 Validating the Performance Model

This experiment evaluates whether the predictions of our scheduler are sufficiently accurate to support online scheduling. Accurate scheduling decisions require reliable estimates of the execution time of candidate pipeline decompositions under changing resource conditions. Rather than empirically profiling every possible decomposition under every combination of network and power contention, PLUMBER predicts execution latency using the analytical performance model described in Section 5.2.1. We compare two prediction approaches. Pipeline-Aware estimates the changing background load experienced throughout pipeline execution as different stages start and end. In contrast,

Model	Avg Latency (s)	Standard Deviation (s)	Pipeline-Aware				Pipeline-Unaware			
			Worst RMSE (s)	Avg RMSE (s)	Worst MAE (s)	Avg MAE (s)	Worst RMSE (s)	Avg RMSE (s)	Worst MAE (s)	Avg MAE (s)
EfficientNetB0	3.998s	0.365s	**0.795s**	**0.572s**	**0.655s**	**0.448s**	3.264s	2.412s	3.241s	2.387s
ResNet18	2.982s	0.123s	**0.652s**	**0.466s**	**0.520s**	**0.374s**	1.879s	1.371s	1.876s	1.365s
ViT_B16	22.294s	4.124s	**5.525s**	**4.476s**	**4.668s**	**3.791s**	23.655s	20.178s	22.619s	19.125s

Table 5.1: Comparison of latency estimation error for Pipeline-Aware vs. Pipeline-Unaware estimations. Errors are reported for Worst and Avg RMSE and MAE. Avg latency and standard deviation is measured when compute is not throttled. Pipeline-Aware estimations (boldface) are significantly closer to measured latencies across background load variations, compared to Pipeline-Unaware, validating our performance model.

Pipeline-Unaware estimation assumes that background load remains constant for the duration of an inference pipeline. Both approaches use the same regression model to estimate throttling effects; they differ only in whether they account for the temporal interference effects during pipeline execution. Prediction accuracy is evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) across varying levels of background load. Each experiment is repeated 30 times.

To compare the prediction accuracy, Table 5.1 summarizes errors for EfficientNetB0, ResNet18, and ViT-B16. Across all three models, Pipeline-Aware, consistently produces substantially lower prediction error than the Pipeline-Unaware baseline. The improvement is particularly pronounced for ViT-B16, where average RMSE decreases from 20.18s to 4.48s. This demonstrates the degree to which performance interference impacts prediction. Similar improvements are observed for RMSE and MAE across all evaluated models. These results demonstrate that Pipeline-Aware estimation is sufficiently accurate for online scheduling, allowing PLUMBER to evaluate candidate decompositions analytically rather than through exhaustive runtime profiling. That is, the scheduler need only evaluate candidate decompositions, rather than execute or profile exhaustively, making online adaptive scheduling computationally practical. The increased prediction accuracy translates directly into improved scheduling decisions. Using the Pipeline-Aware estimator increases scheduling efficiency by at least 63% while reducing energy consumption by at

minimum 46% relative to the Pipeline-Unaware scheduler.

5.3.3 End-to-End Scheduling Performance

We next evaluate the scheduling efficiency of PLUMBER under realistic edge deployments. Our goal is to determine whether adaptive pipeline decomposition results in higher scheduling efficiency and lower energy consumption than static decomposition strategies. We compare PLUMBER against two baseline schedulers that select either compute-optimized (Only Compute) or communication-optimized (Only Communication) decompositions, regardless of current operating conditions. Each scheduler executes identical workloads under both Uniform and Azure Functions arrival traces for different available network bandwidths between 2 and 240 Mbps. Performance is evaluated using Scheduling Efficiency (η), which measures achieved throughput relative to the theoretical best-case schedule, and Relative Energy Efficiency ($\mathcal{E}_{rel}$), which quantifies total energy consumption relative to the same idealized execution. An ideal schedule in this case, is the theoretical schedule if devices never throttle. Note, because η and $\mathcal{E}_{rel}$ are unitless, their ratio is a benefit-to-cost ratio, comparable across all experiments (each having a potentially different timescale).

Table 5.2 summarizes the performance of the decomposition strategies, across all evaluated workloads and network conditions. PLUMBER consistently achieves the maximum normalized throughputs (largest η values). It also achieves the best (largest) benefit-to-cost ratios (shown in boldface in the table). Note, the non-adapting strategies perform well near the extremes with respect to network bandwidth, but neither is best across the full range of conditions. These results demonstrate that dynamically selecting pipeline decompositions allows PLUMBER to adapt to changing resource availability without sacrificing either throughput or energy efficiency. In contrast, PLUMBER dynamically selects

Arrival Type	Avail B/W (Mbps)	PLUMBER			Only Compute			Only Communication		
		$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$
Azure Functions Trace	240	0.42	0.94	**2.24**	0.42	0.94	**2.24**	0.79	0.53	0.67
	30	0.47	0.99	**2.11**	0.47	0.98	2.09	0.5	0.86	1.72
	10	0.47	1.0	**2.13**	0.5	0.99	1.98	0.49	0.99	2.02
	2	0.46	1.0	**2.17**	0.47	1.0	2.13	0.46	1.0	**2.17**
Uniform	240	0.4	0.96	**2.4**	0.4	0.96	**2.4**	0.94	0.62	0.66
	30	0.63	0.99	**1.57**	0.96	0.99	1.03	0.72	0.85	1.18
	10	0.59	1.0	**1.69**	1.21	1.0	0.83	0.62	0.95	1.53
	2	0.73	1.0	**1.37**	1.42	1.0	0.7	0.73	1.0	**1.37**

Table 5.2: Evaluation of energy efficiency($\mathcal{E}_{rel}$) and scheduling throughput(η) across varying bandwidth availability. The largest benefit-to-cost ratio is shown in boldface.

the decomposition strategy most appropriate for the current resource conditions. At intermediate bandwidths (10–30 Mbps), where neither compute- nor communication-optimized decompositions are universally preferable, PLUMBER consistently identifies decompositions to achieve higher throughput and lower energy consumption than either static policy. This adaptive behavior allows PLUMBER to remain close to the theoretical scheduling limit while simultaneously reducing unnecessary device utilization. Across all evaluated deployment conditions, PLUMBER remains within approximately 5% of the theoretical scheduling limit, and frequently within 1% under moderate and low bandwidth. Another interesting observation is, several measured $\mathcal{E}_{rel}$ values, are <1. This result does not imply the strategies exceed theoretical optimums. Rather, the theoretical reference assumes only the absence of power-induced throttling and does not account for workload compaction. By consolidating computation onto fewer devices when appropriate, PLUMBER reduces

both active and idle energy, yielding lower energy consumption than the reference schedule despite operating under realistic resource contention.

Overall, adaptive pipeline decomposition enables PLUMBER to consistently outperform static decomposition strategies across diverse edge deployment conditions, approaching the optimal throughput and reducing energy consumption. Static strategies perform well only when deployment conditions suit their optimization objective, whereas PLUMBER consistently delivers high scheduling efficiency and energy efficiency across diverse workloads, arrival patterns, and network environments.

5.3.4 Impact of Resource Conditions on Scheduling Decisions

We next examine why these improvements occur by analyzing how available network bandwidth and model architecture influence decomposition selection. Specifically, we investigate the resource conditions under which compute-balanced and communication-optimized decompositions become preferable. Figure 5.2 compares the execution time of the compute-balanced and communication-optimized decompositions across network bandwidths for EfficientNetB0, ResNet18, and ViT-B16. As expected, the preferred decomposition depends on the relative cost of computation and communication.

At high network bandwidth (240 Mbps), communication latency contributes only a small fraction of total execution time. Under these conditions, compute-balanced decompositions consistently outperform communication-optimized decompositions because they distribute computation more evenly across participating devices, reducing pipeline imbalance and improving utilization. The additional communication introduced, is largely hidden by the available network capacity.

The crossover point between the two decomposition strategies is not fixed but depends on the computational characteristics of the deployed model. EfficientNetB0, ResNet18,

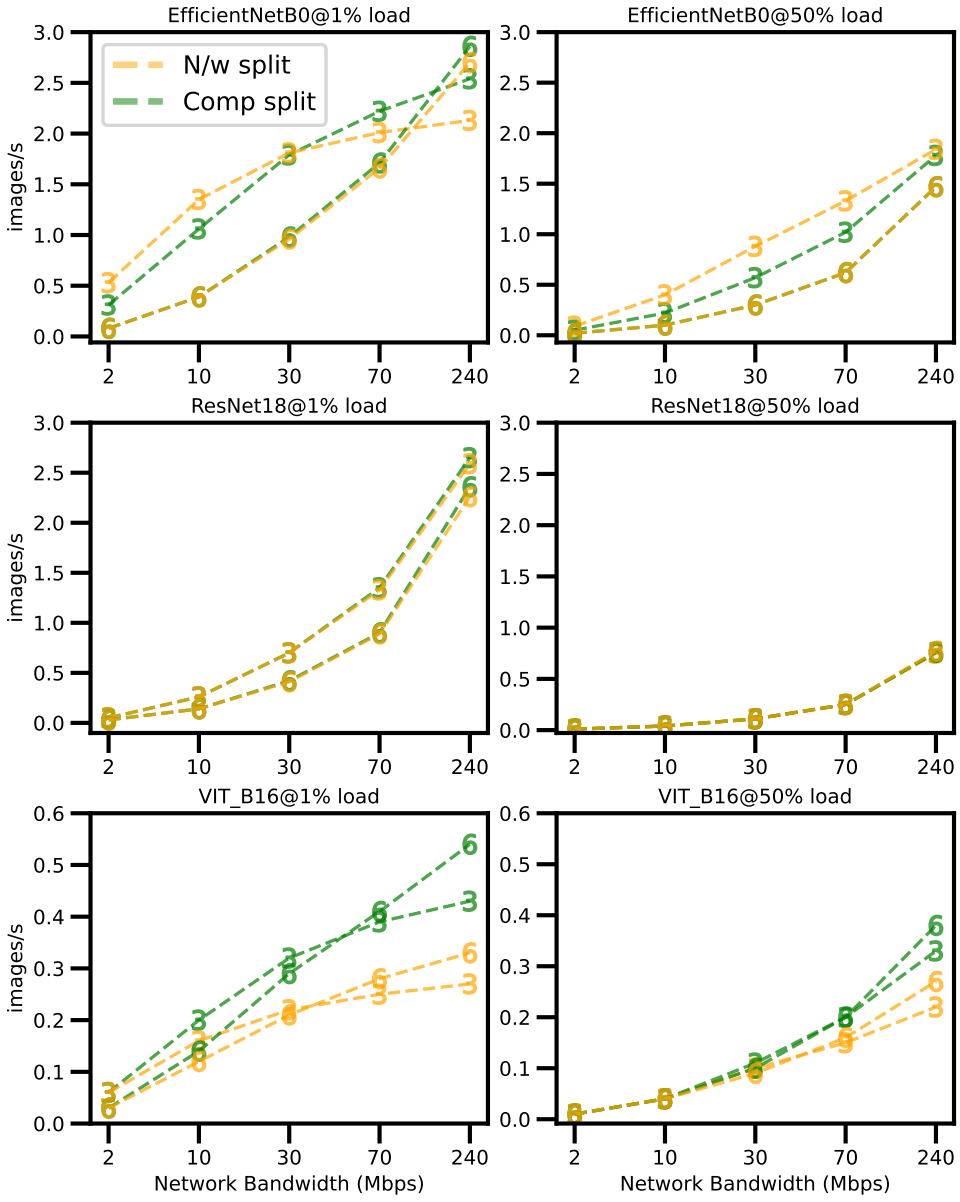

Figure 5.2: Throughputs across varying network bandwidths and background loads. Each row represents a different model (EfficientNetB0, ResNet18, ViT-B16), while columns differentiate between low (1%) and high (50%) subcluster background load. Orange lines denote n/w splits, and green lines denote comp splits. Markers indicate the number of splits (3 or 6). The points represent throughput of the most recent job and higher throughput indicates better decomposition choice.

and ViT-B16 exhibit different ratios of computation to communication, causing each architecture to transition between preferred decomposition strategies at different band-

widths. Consequently, no single decomposition strategy is optimal across all workloads or deployment environments. Thus from a scheduling perspective, pipeline decomposition represents a dynamic tradeoff between computational load balancing and communication overhead. As available resources fluctuate, the scheduler must continuously rebalance these competing objectives to minimize energy consumption and improve scheduling efficiency.

That is, no single pipeline decomposition is universally optimal. And by continuously evaluating candidate decompositions with its online performance model, PLUMBER adapts to changing resource conditions and consistently selects the decomposition predicted for best scheduling and energy efficiency without requiring manually tuned heuristics.

5.3.5 Robustness to Workload Composition

Real-world edge deployments rarely execute a single application in isolation. Instead, workloads evolve over time as new applications are introduced, request rates fluctuate, and the relative frequency of different inference tasks changes. We therefore evaluate the robustness of PLUMBER under varying workload compositions and arrival patterns to determine whether adaptive pipeline decomposition continues to provide benefits when workload characteristics differ from those used to derive the scheduling model.

To evaluate workload diversity, we vary the proportion of EfficientNetB0, ResNet18, and ViT-B16 inference requests while maintaining the same total workload volume. We evaluate scheduler efficiency again using the Uniform and Azure functions job arrival traces. And compare PLUMBER against the specialized decomposition strategies using Scheduling Efficiency (η) and Relative Energy Efficiency ($\mathcal{E}_{rel}$). Table 5.3 shows example results when ViT-B16 jobs occur with a higher probability in the job mix. Similar results for varying work loads are omitted for space considerations, but all confirm the results

Arrival Type	Avail B/W	PLUMBER			Only Compute			Only Communication		
		$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$
Azure Functions Trace	(Mbps) 2	0.67	1.0	**1.49**	1.22	0.89	0.73	0.68	0.99	1.46
Uniform	2	0.66	1.0	**1.52**	1.41	0.81	0.57	0.81	0.93	1.15

Table 5.3: Evaluation of energy efficiency($\mathcal{E}_{rel}$) and scheduling throughput(η) when ViT-B16 has a higher probability of selection than other models in the workload. Here 2 Mbps has non-trivial schedules due to increased ViT-B16 jobs.

shown in Table 5.2 do not depend on workload mix. Across all evaluated workload compositions and arrival patterns, PLUMBER consistently maintains scheduling efficiency comparable to or exceeding both static scheduling policies while preserving similar or lower relative energy consumption.

The benefits of adaptation become particularly apparent as workload composition shifts toward models with different computational and communication requirements, for example, a higher frequency of ViT-B16 inference tasks. Because the preferred decomposition varies across model architectures, changes in workload composition also change the resource demands placed on the cluster. Static strategies cannot account for these changing demands. In contrast, PLUMBER continuously evaluates candidate decompositions for each arriving inference request, allowing scheduling decisions to naturally adapt as the workload evolves. For 2 Mbps, the communication-optimized strategy fails to match the performance of PLUMBER, contrasting with Table 5.2.

These results show, PLUMBER does not rely on a fixed workload mix or arrival pattern to achieve high performance. Instead, adaptive decomposition enables the scheduler to respond to changing application demands while maintaining consistently high throughput and energy efficiency across diverse deployment scenarios. PLUMBER remains effective across changing workload compositions and arrival patterns without requiring workload-specific tuning.

In summary, these experiments demonstrate PLUMBER provides accurate online performance estimation (RQ1), translates predictions into improved scheduling performance (RQ2), adapts decomposition decisions to changing conditions (RQ3), and remains robust across diverse workload compositions and arrival patterns (RQ4). Taken together, these results show adaptive pipeline decomposition enables consistently efficient distributed DL inference on resource-constrained edge clusters.

5.4 Chapter Summary

In this chapter we present PLUMBER, an adaptive scheduler for distributed deep learning inference on resource-constrained edge clusters. By combining online performance modeling with dynamic pipeline decomposition, PLUMBER adapts to changing network conditions, compute contention, workload composition, and arrival patterns. Experimental results on a 117-node Raspberry Pi 3B+ cluster demonstrate adaptive decomposition consistently approaching the optimal solution scheduling efficiency while improving energy efficiency relative to static decomposition strategies. The results show, treating model decomposition as a runtime decision, rather than fixed choices, enables robust, efficient edge AI across dynamic environments.

Chapter 6

Conclusion, Future Directions and Broader Impact

In this dissertation we set out to demonstrate that treating power infrastructure as a first-class systems resource throughout the lifecycle of an edge deployment enables efficient planning, accurate performance characterization, and effective runtime scheduling. We first showcase the co-design of remote edge deployments for power efficiency and communication management. The work in chapter 3 elucidates the usefulness of simulation models to represent edge deployments and how compatible their representation is with constraint solvers. We show how different heuristics handle the complexity of the simulation model, the quality of solutions generated under strict deadlines and how they not only optimize resource allocation but can also inform infrastructure decisions to build edge topologies capable of meeting duty cycle requirements.

But as seen in chapter 4, performance of real, aged clusters can deviate from simulation models. Specifically we characterize these interactions of power infrastructure and performance on 451 real edge devices, into identifying patterns of unreliability. These conditional unreliable patterns are sufficiently captured by linear models and the methodology used

to derive them can be achieved with a reduced profiling set.

And finally in chapter 5 we use these findings to build an end-to-end dynamic scheduler to adaptively decompose ML/AI applications at the edge given the variable availability of compute and communication. In this work we extend the models from previous chapters to fit our real edge cluster and measure its accuracy for online performance estimation. And that using these models we can successfully develop a robust scheduler for diverse workloads and arrival patterns for remote edge ML applications. Overall our body of work showcases the impact of power as a first class constraint to manage and how to build models for such edge deployments in a profile-efficient methodology.

For the future, there is a common limitation to overcome. Specifically all works depend on the tasks to be computationally bound or approximately comparable to matrix multiply. This eliminates applications that are memory, I/O or network bound. Further works can explore implementing the generic methodology from chapter 4 to network, I/O or memory bound applications to measure their power-induced variability. This would help extend our characterization and adaptive scheduling to applications outside ML/AI on the edge.

However, this leads us to the issue of the dissertation falling into the category of "just another scheduler" where developers would have to develop their custom implementations based on our generic methodology, further complicating any of their existing deployment models. Instead, we present a more optimistic view of it's impact with *co-operation in scheduling* [15]. The work was done as part of summer research at Meta, where we extended Meta's data-center job placement frameworks for streaming applications [43] to load balance across their infrastructure. A major challenge in the work was getting multiple levels of schedulers and resource managers to agree with each other. For example, a data-center level allocator would place a job within a server rack to minimize network latencies, but under certain conditions, a rack-level allocator (for example [45]) need not

agree with the top-level decision and perform a different operation – leading to inconsistent placements, unbalanced resource utilization and often involved manual intervention to fix.

The report proposes how we can iteratively use the different hierarchies in scheduling to develop consistent schedules across the different levels of decisions. This allows the schedulers to remain scalable, be aware of only necessary information and still perform their multi-objective optimizations. The results from the work show that this iterative methodology achieves a Pareto-optimal result in terms of load balancing the multiple-objectives and scheduler effort in terms of time taken to make an allocation, compared to simply expanding the objectives within a single allocator, thereby increasing complexity and slowing down scheduling decisions.

For the ML applications our work targets in chapter 5 we already know there's a large body of work (seen in chapter 2) where different modifications are performed on the ML model/algorithms per application or device type rather than resource contention. As future work, and to allow for maximum possible impact, contention-aware schedulers like our work in PLUMBER, can become part of a lower-level allocator, where we can plug into these existing application/hardware adaptive schedulers rather than superpose or replace these existing works. This methodology would improve scalability and extend the flexibility of these existing works, thereby building a holistic ecosystem of schedulers rather than building "Just Another Scheduler" that is near-omniscient and hyper optimized for its infrastructure, being unusable for any other application or cluster design.

Bibliography

[1] H. Ahn and T. Chen, *Performance characterization of using quantization for dnn inference on edge devices*, in *7TH IEEE INTERNATIONAL CONFERENCE ON FOG AND EDGE COMPUTING*, 2023.

[2] C. Avasalcai, C. Tsigkanos, and S. Dustdar, *Decentralized resource auctioning for latency-sensitive edge computing*, in *2019 IEEE International Conference on Edge Computing (EDGE)*, pp. 72–76, 2019.

[3] C. Avasalcai, C. Tsigkanos, and S. Dustdar, *Adaptive management of volatile edge systems at runtime with satisfiability*, *ACM Trans. Internet Technol.* **22** (2021), no. 1.

[4] D. Balouek-Thomert, I. Perez, S. D. Faulstich, H. A. Holmes, I. Altintas, and M. Parashar, *Keynote talk: Leveraging the edge-cloud continuum to manage the impact of wildfires on air quality*, in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 27–31, 2023.

[5] P. J. Basford, S. J. Johnston, C. S. Perkins, T. Garnock-Jones, F. P. Tso, D. Pezaros, R. D. Mullins, E. Yoneki, J. Singer, and S. J. Cox, *Performance analysis of single board computer clusters*, *Future Generation Computer Systems* **102** (2020).

[6] N. BjØrner, A.-D. Phan, and L. Fleckenstein, *Vz - an optimizing smt solver*, in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*, p. 194–199, 2015.

[7] L. Cao, A. Merican, D. Z. Tootaghaj, F. Ahmed, P. Sharma, and V. Saxena, *Ecaas: A management framework of edge container as a service for business workload*, in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '21, (New York, NY, USA), p. 73–78, Association for Computing Machinery, 2021.

[8] L. Chen, C. Lin, S. Luo, H. Xu, and C. Xu, *Grad: Intelligent microservice scaling by harnessing resource fungibility*, in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 474–486, 2025.

[9] S. Chen, A. Jin, C. Delimitrou, and J. F. Martínez, *Retail: Opting for learning simplicity to enable qos-aware power management in the cloud*, in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 155–168, IEEE, 2022.

[10] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, *Towards efficient model compression via learned global ranking*, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1518–1528, 2020.

[11] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'brien, *Iridis-pi: a low-cost, compact demonstration cluster*, *Cluster Computing* **17** (2014), no. 2 349–358.

[12] I. I. Cplex, *V12. 1: User's manual for cplex*, *International Business Machines Corporation* **46** (2009), no. 53 157.

[13] A. N. Dangwal, R. Singh, S. Unnithan, K. Yuan, E. Zheng, C. Krintz, and R. Wolski, "Plumber: Scheduling deep learning inference pipelines on resource-constrained edge clusters." Publication pending.

[14] A. N. Dangwal, R. Singh, S. Unnithan, K. Yuan, E. Zheng, C. Krintz, and R. Wolski, *Reliable unreliability in edge deployments*, in *International Conference On Distributed Computing In Smart Systems And The Internet Of Things (DCOSS)*, June, 2026.

[15] A. Dangwal, Y. Jiang, C. Arnold, J. Fan, M. Bassem, and A. Rajagopal, *Designing co-operation in systems of hierarchical, multi-objective schedulers for stream processing*, *arXiv preprint arXiv:2512.07792* (2025).

[16] L. de Moura and N. Bjørner, *Z3: An efficient smt solver*, in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

[17] O. R. developers, "Onnx runtime." `https://onnxruntime.ai/`, 2021. V1.19.2.

[18] I. S. B. Dimitrios Papakyriakou, *High performance linpack (hpl) benchmark on raspberry pi 4b (8gb) beowulf cluster*, *International Journal of Computer Applications* **185** (Jul, 2023) 11–19.

[19] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et. al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, *arXiv preprint arXiv:2010.11929* (2020).

[20] R. Du, M. Xiao, and C. Fischione, *Optimal node deployment and energy provision for wirelessly powered sensor networks*, *IEEE Journal on Selected Areas in Communications* **37** (2019), no. 2 407–423.

[21] M. Effros, A. Goldsmith, and M. Médard, *The rise of instant wireless networks*, *Scientific American* **302** (2010), no. 4 72–77.

[22] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski, *Where's the Bear? – Automating Wildlife Image Processing Using IoT and Edge Cloud Systems*, in *Internet-of-Things Design and Implementation (IoTDI), 2017 IEEE/ACM Second International Conference on*, pp. 247–258, IEEE, 2017.

[23] S. Elsayed, C. Krintz, and R. Wolski, *Just the FACTS: Flexible and Energy Efficient Federated Access Control for the Edge*, in *IEEE International Conference on Fog and Mobile Edge Computing*, 2024.

[24] R. P. Foundation, *Raspberry pi computer hardware - power supply*, 2026. Accessed: 2026-01-01.

[25] J. Gascon-Samson, M. Rafiuzzaman, and K. Pattabiraman, *Thingsjs: Towards a flexible and self-adaptable middleware for dynamic and heterogeneous iot environments*, in *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things*, (New York, NY, USA), p. 11–16, Association for Computing Machinery, 2017.

[26] J. Geerling, "Power consumption benchmarks." `https://pidramble.com/wiki/benchmarks/power-consumption`, 2026. Accessed: 2026-07-30.

[27] P. Ghosh, J. Bunton, D. Pylorof, M. A. M. Vieira, K. Chan, R. Govindan, G. S. Sukhatme, P. Tabuada, and G. Verma, *Synthesis of large-scale instant iot networks*, *IEEE Transactions on Mobile Computing* **22** (2023), no. 3 1810–1824.

[28] M. Giordani, T. Shimizu, A. Zanella, T. Higuchi, O. Altintas, and M. Zorzi, *Path loss models for v2v mmwave communication: Performance evaluation and open challenges*, in *2019 IEEE 2nd Connected and Automated Vehicles Symposium (CAVS)*, pp. 1–5, IEEE, 2019.

[29] V. Gohil, S. Dev, G. Upasani, D. Lo, P. Ranganathan, and C. Delimitrou, *The importance of generalizability in machine learning for systems*, *IEEE Computer Architecture Letters* **23** (2024), no. 1.

[30] N. Golubovic, R. Wolski, C. Krintz, and M. Mock, *Improving the Accuracy of Outdoor Temperature Prediction by IoT Devices*, in *IEEE Conference on IoT*, 2019.

[31] P. Guan, A. Dangwal, A. Taherkordi, R. Wolski, and C. Krintz, *Energy-aware iot deployment planning*, in *Proceedings of the 21st ACM International Conference on Computing Frontiers*, pp. 61–70, 2024.

[32] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, D. Srinivasan, B. Panda, A. Baptist, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, *Fail-slow at scale: Evidence of hardware performance faults in large production systems*, *ACM Trans. Storage* **14** (Oct., 2018).

[33] J. Haj-Yahya, M. Alser, J. Kim, A. G. Yağlıkçı, N. Vijaykumar, E. Rotem, and O. Mutlu, *Sysscale: exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors*, in *International Symposium on Computer Architecture*, 2020.

[34] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, *arXiv preprint arXiv:1510.00149* (2015).

[35] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, *Thermal response to dvfs: analysis with an intel pentium m*, in *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, (New York, NY, USA), p. 219–224, Association for Computing Machinery, 2007.

[36] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[37] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, *Couper: Dnn model slicing for visual analytics containers at the edge*, in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, (New York, NY, USA), p. 179–194, Association for Computing Machinery, 2019.

[38] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. A. Beerel, S. P. Crago, and J. P. Walters, *Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices*, in *2022 25th Euromicro Conference on Digital System Design (DSD)*, pp. 298–307, 2022.

[39] K. Huang and W. Gao, *Real-time neural network inference on extremely weak devices: agile offloading with explainable ai*, in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, (New York, NY, USA), p. 200–213, Association for Computing Machinery, 2022.

[40] H. Jawad, R. Nordin, S. Gharghan, A. Jawad, and M. Ismail, *Energy-efficient wireless sensor networks for precision agriculture: A review*, *Sensors* **17** (2017), no. 1781.

[41] C. Krintz, R. Wolski, N. Golubovic, B. Lampel, V. Kulkarni, B. Sethuramasamyraja, B. Roberts, and B. Liu, *SmartFarm: Improving Agriculture Sustainability Using Modern Information Technology*, in *KDD Workshop on Data Science for Food, Energy, and Water*, Aug., 2016.

[42] Z. Krpić, I. Lukić, M. Habijan, and L. Loina, *Re-evaluating compute performance in sbc clusters: Hpl benchmarking across generations*, *Future Generation Computer Systems* **176** (2026).

[43] N. Kumar, P. M. Ruiz, V. Menon, I. Kabiljo, M. Pundir, A. Newell, D. Lee, L. Wang, and C. Tang, *Optimizing resource allocation in hyperscale datacenters: Scalability, usability, and experiences*, in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 507–528, 2024.

[44] L. Kurafeeva, A. Subedi, R. Hartung, M. Fay, A. Biswas, S. Jha, O. Kilic, C. Krintz, A. Merzky, D. Thain, *et. al.*, *xgfabric: Coupling sensor networks and hpc facilities with private 5g wireless networks for real-time digital agriculture*, in *Proceedings of the SC'25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 2317–2327, 2025.

[45] S. Lee, Z. Guo, O. Sunercan, J. Ying, T. Kooburat, S. Biswal, J. Chen, K. Huang, Y. Cheung, Y. Zhou, *et. al.*, *Shard manager: A generic shard management framework for geo-distributed applications*, in *Proceedings of the ACM SIGOPS 28th symposium on operating systems principles*, pp. 553–569, 2021.

[46] Y. Li, M. Ghasemiahmadi, and L. Cai, *Uplink cooperative transmission for machine-type communication traffic in cellular system*, in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, IEEE, 2016.

[47] Q. Liang, P. Shenoy, and D. Irwin, *Ai on the edge: Rethinking ai-based iot applications using specialized edge architectures*, *arXiv preprint arXiv:2003.12488* (2020).

[48] C. Liu, J. Wang, L. Zhou, and A. Rezaeipanah, *Solving the multi-objective problem of iot service placement in fog computing using cuckoo search algorithm*, *Neural Processing Letters* **54** (2022), no. 3 1823–1854.

[49] MaxLinear, *Five Output Universal PMIC, DataSheet*, 03, 2018. Rev 1B.

[50] N. M. Mwasaga, *Design and evaluation of micro high-performance computing as an educational tool to improve high-performance computing usability*, 2022.

[51] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. Sweeney, *Producing wrong data without doing anything obviously wrong!*, in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2009.

[52] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, *Pipedream: Generalized pipeline parallelism for dnn training*, in *Proceedings of the 27th ACM symposium on operating systems principles*, pp. 1–15, 2019.

[53] P. Ngoy, F. Dar, M. Liyanage, Z. Yin, U. Norbisrath, A. Zuniga, J. Pestana, M. Radeta, P. Nurmi, and H. Flores, *Supporting sustainable computing by repurposing e-waste smartphones as tiny data centers*, *IEEE Pervasive Computing* (2025).

[54] V. Ninkovic, D. Vukobratovic, D. Miskovic, and M. Zennaro, *Comsplit: A communication-aware split learning design for heterogeneous iot platforms*, *IEEE Internet of Things Journal* **12** (2024), no. 5 5305–5319.

[55] J. Okwuibe, J. Haavisto, I. Kovacevic, E. Harjula, I. Ahmad, J. Islam, and M. Ylianttila, *Sdn-enabled resource orchestration for industrial iot in collaborative edge-cloud networks*, *IEEE Access* **9** (2021).

[56] J. H. Park, G. Yun, M. Y. Chang, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, {*HetPipe*}*: Enabling large* {*DNN*} *training on (whimpy) heterogeneous* {*GPU*} *clusters through integration of pipelined model parallelism and data parallelism*, in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 307–321, 2020.

[57] Y. Peng, S. Chen, Y. Zhao, and Z. Yu, *UFO: The ultimate QoS-Aware core management for virtualized and oversubscribed public clouds*, in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, (Santa Clara, CA), pp. 1511–1530, USENIX Association, Apr., 2024.

[58] G. Pottie and W. Kaiser, *Wireless integrated network sensors*, *Commun. ACM* **43** (2000), no. 5.

[59] A. Prasanth and S. Jayachitra, *A novel multi-objective optimization strategy for enhancing quality of service in iot-enabled wsn applications*, *Peer-to-Peer Networking and Applications* **13** (2020) 1905–1920.

[60] T. Rault, A. Bouabdallah, and Y. Challal, *Energy efficiency in wireless sensor networks: A top-down survey*, *Computer Networks* **67** (2014).

[61] F. Romero, G. I. Chaudhry, Í. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca, C. Kozyrakis, and R. Bianchini, *Faa$t: A transparent auto-scaling cache for serverless applications*, in *Proceedings of the ACM symposium on cloud computing*, pp. 122–137, 2021.

[62] S. F. Roselli, K. Bengtsson, and K. Åkesson, *Smt solvers for flexible job-shop scheduling problems: A computational analysis*, in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019.

[63] M. Schlueter, J. Egea, and J. Banga, *Extended ant colony optimization for non-convex mixed integer nonlinear programming*, *Computers and Operations Research* **36** (2009), no. 7.

[64] M. Schlueter, J. Egea, and J. Banga, *Extended ant colony optimization for non-convex mixed integer nonlinear programming*, *Computers and Operations Research* **36** (2009), no. 7 2217–2229.

[65] G. Schwarz, *Estimating the dimension of a model*, *Annals of Statistics* **6** (1978), no. 2.

[66] B. Shi, W. Wei, Y. Wang, and W. Shu, *A novel energy efficient topology control scheme based on a coverage-preserving and sleep scheduling model for sensor networks*, *Sensors* **16** (2016), no. 10.

[67] H. Song, R. Dautov, N. Ferry, A. Solberg, and F. Fleurey, *Model-based fleet deployment of edge computing applications*, in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, (New York, NY, USA), p. 132–142, Association for Computing Machinery, 2020.

[68] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, *Memory errors in modern systems: The good, the bad, and the ugly*, *ACM SIGARCH Computer Architecture News* **43** (2015), no. 1 297–310.

[69] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto, *Junkyard computing: Repurposing discarded smartphones to minimize carbon*, in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[70] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez, *Lwhbench: A low-level hardware component benchmark and dataset for single board computers*, *Internet of Things* **22** (July, 2023).

[71] M. Tan and Q. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[72] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, *The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures*, in *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pp. 108–112, IEEE, 2013.

[73] "UCSB SmartFarm." `http://www.cs.ucsb.edu/~ckrintz/projects/index.html`. [Online; accessed 25-July-2016].

[74] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, A. Sanglas, and F. Palomares, *Reliable and efficient integration of ai into camera traps for smart wildlife monitoring based on continual learning*, *Ecological Informatics* **83** (2024) 102815.

[75] K. V. Vishwanath and N. Nagappan, *Characterizing cloud computing hardware reliability*, in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, (New York, NY, USA), p. 193–204, Association for Computing Machinery, 2010.

[76] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, *Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics*, in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 257–266, IEEE, 2020.

[77] J.-B. Wang, J. Zhang, C. Ding, H. Zhang, M. Lin, and J. Wang, *Joint optimization of transmission bandwidth allocation and data compression for mobile-edge computing systems*, *IEEE Communications Letters* **24** (2020), no. 10 2245–2249.

[78] W. Wang, K. Li, B. Ji, X. Liu, J. Yu, and Q. Wu, *A survey of ai inference technologies for on-device systems*, *IEEE Internet of Things Journal* (2025).

[79] W. Wei, R. Yang, H. Gu, W. Zhao, C. Chen, and S. Wan, *Multi-objective optimization for resource allocation in vehicular cloud computing networks*, *IEEE Transactions on Intelligent Transportation Systems* **23** (2021), no. 12 25536–25545.

[80] M. Weiser, B. Welch, A. Demers, and S. Shenker, *Scheduling for reduced CPU energy*, in *First Symposium on Operating Systems Design and Implementation (OSDI 94)*, (Monterey, CA), USENIX Association, Nov., 1994.

[81] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, *A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters*, *Future Generation Computer Systems* **37** (2014) 141–147.

[82] B. Xie and H. Cui, *Deep reinforcement learning-based dynamical task offloading for mobile edge computing*, *The Journal of Supercomputing* **81** (2025), no. 1 35.

[83] B. Yang, J. Zhang, J. Li, C. Ré, C. R. Aberger, and C. De Sa, *Pipemare: Asynchronous pipeline parallel dnn training*, *arXiv preprint arXiv:1910.05124* (2019).

[84] Y. Yu, J. Tang, J. Huang, X. Zhang, D. K. C. So, and K.-K. Wong, *Multi-objective optimization for uav-assisted wireless powered iot networks based on extended ddpg algorithm*, *IEEE Transactions on Communications* **69** (2021), no. 9 6361–6374.

[85] S. Yue and J. Ren, *Efficient federated meta-learning over multi-access wireless networks*, *Next Generation Multiple Access* (2024) 547–581.

[86] "Z3 Prover." `https://github.com/Z3Prover/z3`. [Online; accessed 12-September-2023].

[87] S. Zhang, C. Wang, Y. Jin, J. Wu, Z. Qian, M. Xiao, and S. Lu, *Adaptive configuration selection and bandwidth allocation for edge-based video analytics*, *IEEE/ACM Transactions on Networking* **30** (2021), no. 1 285–298.

[88] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, *A systematic dnn weight pruning framework using alternating direction method of multipliers*, in *Proceedings of the European Conference on Computer Vision (ECCV)*, September, 2018.

[89] Y. Zhao, H. Sharif, P. Pao-Huang, V. Shah, A. N. Sivakumar, M. Valverde Gasparino, A. Mahmoud, N. Zhao, S. Adve, G. Chowdhary, *et. al.*, *Approxcaliper: A programmable framework for application-aware neural network optimization*, *Proceedings of Machine Learning and Systems* **5** (2023) 400–413.

[90] Z. Zong, M. Guo, M. Zhai, Y. Tang, J. Li, and J. Zhai, *Training large models on heterogeneous and geo-distributed resource with constricted networks*, *Big Data Mining and Analytics* **8** (2025), no. 4 966–980.