UNIVERSITY OF CALIFORNIA
RIVERSIDE

Machine Learning-Assisted Resource Management in Edge Computing Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Zhihui Shao

June 2021

Dissertation Committee:

Dr. Shaolei Ren, Chairperson
Dr. Christian Shelton
Dr. Chengyu Song
Dr. Daniel Wong

The Dissertation of Zhihui Shao is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

During the preparation of this thesis, I have received a lot of invaluable help and tremendous supports from my supervisor, committee, lab mates, friends, and family. I would like to take this opportunity to express my heartfelt gratitude to all who have helped me.

Firstly, I would like to give my sincerest thanks to my advisor Professor Shaolei Ren for his continuous supports and assistance during my Ph.D. study at UC, Riverside. Dr. Ren provides me lots of guidance on my research topic as well as the learning methodology to be a successful researcher. Without his painstaking teaching and insightful advice, the completion of this thesis would not have been possible. Also, I would like to thank my dissertation committee: Professor Christian Shelton, Professor Chengyu Song, and Professor Daniel Wong, for their comments and suggestions in my qualify, proposal, and final defense exam.

Additionally, I would like to thank all the co-authors for their insightful guidance and assistance in my research papers, which are the main part of my dissertation. Specifically, I would like to thank Professor Mohammad Atiqul Islam for his contribution to my publications. Professor Islam teaches me a lot, including but not limited to, computing system insights, experiment designs, and manuscript written skills.

Also, I would like to thank all my labmates in the Nonlinear computing lab at UC, Riverside. Specifically, I thank Jianyi Yang, Luting Yang, Fangfang Yang, and Bingqian Lu for your great for their generous support and help during my four-year Ph.D. life.

Last but not least, I would like to extend my deep gratefulness to my family, for their persistent support and encouragement. Most importantly, a special thanks to my beloved wife Jielin Guo who is always by my side for last eight years, from China to US.

Funding Acknowledgment. I appreciate the funding support from National Science Foundation under grant number CNS-1551661 and ECCS-1610471.

Publication Acknowledgment. The text of this dissertation, in part or in full, is a reprint of the material as it appears in list of publications below. The co-author Dr. Shaolei Ren listed in that publication directed and supervised the research which forms the basis for this dissertation.

1. Zhihui Shao, Mohammad Atiqul Islam, Shaolei Ren, *"DeepPM: Efficient Power Management in Edge Data Centers using Energy Storage"*, 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), pp370-379 [122].

   Added to the dissertation as Chapter 2. Zhihui Shao is the major contributor of the project. Mohammad Atiqul Islam assisted in the evaluation and experiments.

2. Zhihui Shao, Mohammad Atiqul Islam, Shaolei Ren, *"Heat Behind the Meter: A Hidden Threat of Thermal Attacks in Edge Colocation Data Centers"*, IEEE Intl. Symp. on High Performance Computer Architecture, Seoul , 2021 [123].

   Added to the dissertation as Chapter 3. Zhihui Shao is the major contributor of the project. Other co-author, Mohammad Atiqul Islam, assisted in the evaluation and experiments. Besides, the preliminary idea of chapter 3 is proposed in workshop [121].

To my wife.

ABSTRACT OF THE DISSERTATION

Machine Learning-Assisted Resource Management in Edge Computing Systems

by

Zhihui Shao

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2021
Dr. Shaolei Ren, Chairperson

The widespread adoption of the Internet of Things and latency-critical applications has fueled the burgeoning development of edge colocation data centers (a.k.a., edge colocation) - small-scale data centers in distributed locations. For such data centers, optimal resource management is crucial for system efficiency and security. In this dissertation, we explore machine learning-assisted approaches to optimize resource management. Firstly, we propose battery-assisted power management in edge data centers considering the computing performance and thermal behavior under significant workload fluctuations. In particular, the workload fluctuations allow the battery to be frequently recharged and made available for temporary capacity boosts. But, using batteries can overload the data center cooling system which is designed with a matching capacity of the power system. We design a novel power management solution, DeepPM, that exploits the UPS battery and cold air inside the edge data center as energy storage to boost performance. DeepPM uses deep reinforcement learning (DRL) to learn the data center thermal behavior online in a model-free manner and uses it on-the-fly to determine power allocation for optimum latency performance without

overheating the data center. Next, we study the vulnerability and thermal attack opportunities from the mismatch between power load and cooling load in edge colocation data centers. We discover that the sharing of cooling systems also exposes edge colocations' potential vulnerabilities to cooling load injection attacks (called thermal attacks) by an attacker which, if left at large, may create thermal emergencies and even trigger system outages. Importantly, thermal attacks can be launched by leveraging the emerging architecture of built-in batteries integrated with servers that can conceal the attacker's actual server power (or cooling load). We consider both one-shot attacks (which aim at creating system outages) repeated attacks (which aim at causing frequent thermal emergencies). For repeated attacks, we present a foresighted attack strategy which, using reinforcement learning, learns on the fly a good timing for attacks based on the battery state and benign tenants' load. Finally, we investigate the general combinatorial optimization problems, focusing on robust solutions utilizing machine learning-assisted methods. Combinatorial optimization is commonly used in many applications , but in general very challenging to solve due to its NP-hard nature. The robust combinatorial optimization is even more difficult, which can be formulated as a minimax problem. To solve the minimax problems efficiently, we propose a novel machine learning-assistant method — RobustPN, leveraging the *actor-critic* platform and *learning to optimize* techniques. To verify the performance of RobustPN, we perform simulations on two combinatorial optimizations: a synthesis problem with three variables and a workload offloading problem in edge computing. Our result shows that the proposed RobustPN can provide robust solutions for combinatorial optimization efficiently.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The widespread adoption of the Internet of Things (IoT) and latency-critical applications has fueled the burgeoning development of edge co-location data centers (a.k.a., edge co-location) - small-scale data centers in distributed locations. Due to limited resources and demand for low latency, we conduct several explorations for resource management in edge computing systems using machine learning methods, including both the physical resources (e.g, power and battery) and the computing resources. Besides, we tack the resource management from the perspective of both data center operators and users (attack vulnerability).

In this chapter, we will first introduce the resources for an edge computing system. Then, brief background knowledge of machine learning methods is presented in section 1.2, such as deep reinforcement learning. Finally, we summarize the main contribution of this thesis in section 1.3.

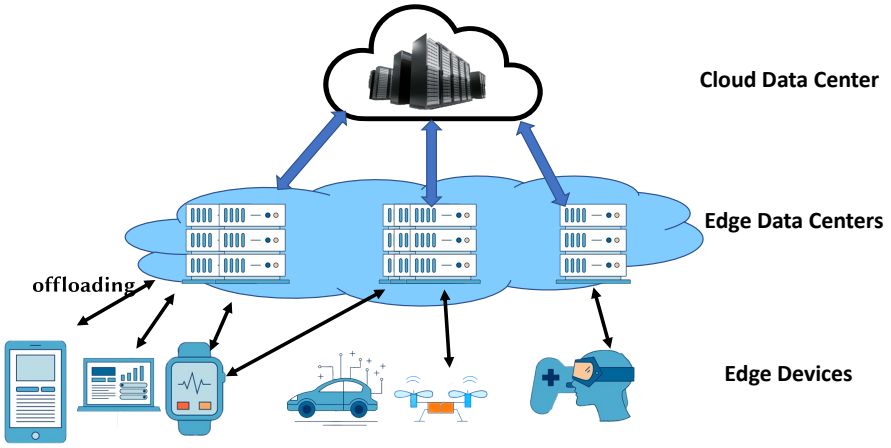


Figure 1.1: Edge computing system.

1.1 Edge Data Center Resources Management

With the recent development of faster internet speed (e.g., 5G), smart devices, and the Internet of Things, edge computing has experienced great growth as a complementary to cloud computing. According to [63], the edge computing market size accounts for $3.5 billion in 2019 2020, and the edge computing demands largely increase during the outbreak of COVID-19. With huge edge computing demands, the edge data center is becoming a new core component in a modern computing system, handling hundreds of millions of user requests and providing ultra-low latency services for latency-critical applications, such as virtual reality and autonomous driving.

Figure 1.1 shows a typical architecture for mobile edge computing system. Here, the cloud data center is usually deployed far away with huge computing capacity (e.g., Google Datacenter). The cloud data center can provide powerful computing resources but with a large servicing time. Serving as complements, the edge data centers are usually distributedly installed closer to users (near the internet edge) in the urban region. Comparing

with the cloud data center, the edge data centers can provide much lower latency, which is significantly important for current user-interactive applications and stream video services. However, constrained by the limited space, the edge data center is usually quite small, housing only several physical servers. Hence, the resources in edge data centers are both precious and vulnerable. To improve the service quality and availability, it is important to explore and manage the limited resources in a graceful manner.

In this section, we introduce the general resource allocation problems in edge computing system: including physical resources management (section 1.1.1) and computing resources allocation (section 1.1.2).

1.1.1 Physical Resources Management

An edge data center typical represents a set of servers and the respective physical control system, such as power control/supply system and cooling system. The physical servers provide ubiquitous computing services to users, whereas the power and cooling system deliver indispensable resources to servers for their normal operation. For instance, the power supply system provides the required energy to power up the servers and protect short-term utility power outages using the pre-installed battery. The detailed architecture and physical resources are shown in Figure 1.2 for a small edge data center.

As shown in Figure 1.2, an edge data center is installed with a power supply system and a cooling system. The power system drains AC energy from the power utility, which is converting to DC power by an AC-DC converter. Then, the power is delivered to servers through power distributed units (PDU), using the pre-installed battery as an energy buffer

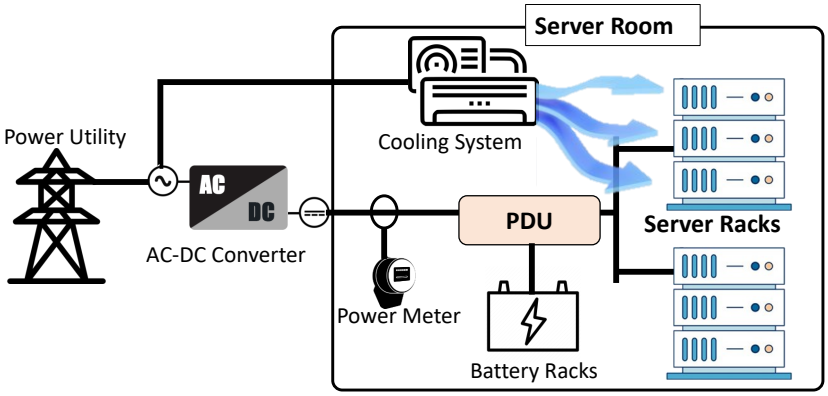


Figure 1.2: Power supply and cooling system in an edge data center.

to handle emergencies. Also, a power metered is usually installed to monitor and control the power consumption of servers. During normal operation, servers consume energy from the power supply system and most of the power energy is converted to heat. Then, the cooling system works to remove the heat to ensure a lower temperature in the server room. Typically, the cooling system includes an air-conditioner (AC) and some accessories, such as air ducts and fans. More detailed information about the cooling and power system can be found in Chapter 2 and Chapter 3.

In an edge data center, the dilemma of power management arises from the mismatch between huge arriving workloads and limited designed power capacity. To increase the utilization, power oversubscription is widely adopted in edge data centers. The power oversubscription implies that only partial workload can be processed at the edge data center and the rest will be either offloaded to the cloud or dropped. For a better service, the pre-installed battery unit is widely used to provide extra power for peak shaving [7, 25]. Then, the general power management problem is to decide the power consumption of servers or the charge/discharge of the battery unit. The detailed problem formulation can be found in Chapter 2, where we consider the power management under power oversubscription and

thermal constraints. According to our literature reviews, Chapter 2 is the first work that considers power management with data center level thermal constraints.

As an extension of the power management problem, we explore the security of an edge data center, from the perspective of thermal vulnerability. In contrast to the power management with batteries, here we consider the dangers of pre-installed batteries, which can serve as a weapon by generating thermal attacks. The detailed problem formulation and experiment verification can be found in chapter 3. In this part, the novelty arises from the thermal vulnerability of the current power and cooling system. Also, we solve repeated thermal attack problems via reinforcement learning.

1.1.2 Computing Resources Allocation and Workload Assignment

As shown in Figure 1.1, the computational workloads of edge devices are offloading edge or cloud data center and processed by respective physical servers. Specifically, one workload can be offloaded to multiple servers and one server may also provides services to multiple users. Due to the space limitations, the computation resources (e.g., CPU) are often in shortage under the ever-increasing computation requirement from edge devices. Also, the resource capacity differs and varies for edge data centers, relying on their physical location and surrounding environment. Therefore, an operator policy is required to manage the dynamic computing resources and in modern computing systems with multiple users and multiple physical servers.

In general, we consider an computing system, housing N eligible physical edge servers and providing serves for edge devices (e.g., vehicular applications). The target is to determine the N-dimensional offloading action $\mathbf{a} = (a_1, a_2, ...a_N)^T$ given the side

information $\mathbf{x} = (x_1, x_2...x_N)^T$ of each server-workload pair, in order to maximize the overall performance. The performance can be presented as $U = F(\mathbf{x}, \mathbf{a})$, where F is the utility function. Note that F may be non-convex and non-concave function. Typically, an optimal policy can be obtained by solving the respective optimization problem in (1.1), where $\mathcal{A}$ is the eligible action space. Since the offloading actions are discrete (binary) variables, the problem (1.1) can be also considered as a combinatorial optimization problem.

$$\pi(x) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} F(\mathbf{x}, \mathbf{a}) \tag{1.1}$$

Combinatorial optimization is a notorious NP-hard problem, which is difficult to solve within polynomial time. Previously, greedy algorithms are designed for the combinatorial problems, such as traveling salesman problem (TSP) [59, 161]. However, these greedy algorithms cannot guarantee a good approximation to the optimal solution. Fortunately, some recent researches attempt to solve the combinatorial optimization, utilizing deep learning techniques. For instance, [31] proposes to learn an heuristic for the TSP problem using policy gradient, and [13] approximates the optimal policy for TSP using reinforcement learning.

However, these works are usually focusing on the optimization problems with an accurate parameter x, such as the distance between each cities in TSP. In edge computing, the optimization parameter x is usually estimated with a prediction model, and then optimization is applied based on estimated x. This framework is denoted "Predict, then Optimize" [34]. For example, the total serving time or success rate of edge offloading is typically estimated with historical data. Due to estimation errors, the parameter $\mathbf{x}$ is in-

accurate, resulting in performance degradation. Considering these inaccuracies, a robust policy is required for the resource management and workload assignment. Hence, we focus on the robust combinatorial optimization and solve it leveraging machine learning-assisted method.

$$\pi(x) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \min_{\delta \in \Delta(\mathbf{x})} F(\mathbf{x} + \delta, \mathbf{a}) \tag{1.2}$$

In general, the robust combinatorial optimization can be formulated as a mini-max problem in Eqn (1.2). Here, $\Delta(\mathbf{x})$ is the error space relevant to $\mathbf{x}$, which is defined by the respective robust constrains and the error budget. For instance, under L_2 norm constrain, it can be represented as $|\delta|_2 \leq \epsilon$ and ϵ is the respect error budget. More detailed information about robust optimization is presented in chapter 4.

1.2 Machine Learning Preliminaries

With increasing computation capacity and data, machine learning is widely used in many industry areas, such as computer vision, control system, etc [50]. In this thesis, we propose to leverage machine learning for resource management problem in computing systems. This part firstly introduces the preliminary knowledge and machine learning techniques used in this dissertation. Specifically, we first present a discussion on the Markov decision process (MDP), which is the fundamental theory for reinforcement learning. Then, we introduce two popular reinforcement learning algorithms — Q learning and Deep-Q-Network (DQN). Finally, a machine learning-assisted optimization method, called *'learning to optimize'*(L2O) is presented for general resource management problems in section 1.2.4.

1.2.1 Markov Decision Process

In edge computing systems, many resources, such as battery energy, are correlated in the time domain. Specifically, the change of battery energy due to allocation decisions will have impacts on its future energy. So, it is difficult to solve the allocation problem directly. Fortunately, MDP can be utilized to model such resource allocation problems with time correlation, where the resource is modeled as the state. The general MDP formulation can be summaries as four parts:

- System state: $s \in \mathcal{S}$
- Action: $a \in \mathcal{A}(s)$
- State transition probability: $P(s, a, s')$
- Reward function: $r = R(s, a, s')$

Here, s is the observed state, and $\mathcal{S}$ is the respective domain. Then, a is the target action variable, whereas $\mathcal{A}(s)$ is the eligible action space based on the current state. Importantly, the tuple (s, a, a') means the state transition from s to a new state s' when action a is take. Respectively, P defines the transition probability of the tuple and R represents the reward from this transition. Note that P and R depend on the system environment, which is hard to model directly.

In MDP, given a fixed policy $\pi(s)$, a sequence $(s_0, a_0, r_0, s_1, a_1, r_1, s_2 \dots)$ can be recorded based on environment models, where s_0 is the initial state . The objective of MDP problem is to find a policy π^* to maximize the long-term reward $\sum_t [\gamma^t \cdot r_t]$, where $\gamma \in (0, 1)$ is the discount factor. A typical diagram for MDP is shown in Figure 1.3.

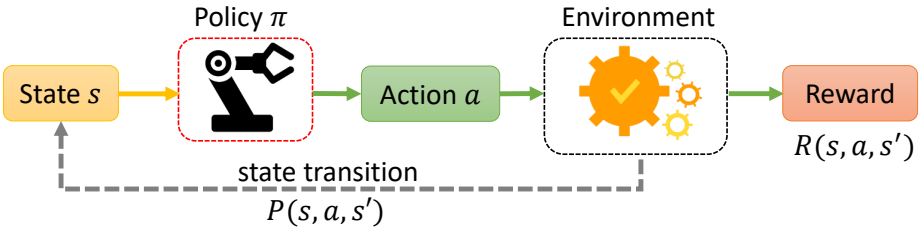


Figure 1.3: General MDP problem.

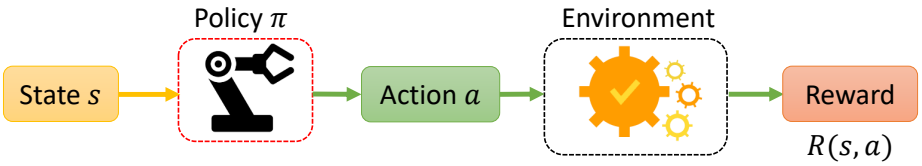


Figure 1.4: Simplified MDP without state transition.

Additionally, there is a special situation for resource allocation, where the actions have a small or nearly no impact on states. In this thesis, we denote it as a simple state MDP. Since the state is not correlated with action, we omit the state transition P and the reward is defined as $R(s, a)$. For instance, in the workload-offloading problem, the arriving workloads usually have no or small relation with allocation results. Hence, we handle the 'computing resources allocation problem' using simple state MDP.

Similarly, the objective is to find a policy π^* to maximize the total reward over the entire state distribution $\mathcal{S}$: $\pi^* = \text{argmax}_\pi \sum_{s \in \mathcal{S}} [R(s, \pi(s))]$. With an unknown reward function $R(s, a)$, it can also be treated as the multi-arm bandit problem. When the reward function $R(s, a)$ is known and defined in advance, it can be solved as optimization problems. In this disertation, we are focusing on the problem with a known reward function, leveraging machine learning to solve the optimization problem. A typical diagram for simple state MDP (optimizations) is shown in Figure Figure 1.4.

1.2.2 Reinforcement Learning

With correlations among states and actions, the general MDP problem is hard to solve directly using conventional optimization algorithms. Reinforcement learning (RL) is a category of algorithms widely used to solve MDP problem [74, 138]. There are two sub-groups of RL: model-based RL and model-free RL. Model-based RL solves MDP with known or learned environment models, whereas model-free RL does not require any models on P or R and learns the policy directly. Since the environment is unknown and hard to model in edge computing systems, this thesis will focus on model-free RL for the physical resource allocation problem. For instance, Q-learning is one popular model-free RL algorithm [58, 162]. In model-free RL, an agent learns a policy through the interaction with the environment, including four steps as following:

- Observe current system state s;
- Provide an action based on the trade-off between exploration and exploitation;
- Observe new system state s' and the instant reward r;
- Update the policy based on the bellman equation (1.3).

In model-free RL, the action strategy is very important, considering the trade-off between exploration and exploitation. For instance, ϵ-greedy is one exploration method used in this thesis. Besides, there are two methods when updating the policy: on-policy and off-policy. For instance, Q-learning is a popular off-policy model-free RL algorithm, while 'SARSA' is an on-policy model-free RL [140]. Here, on policy means that the Q

value is updated using the most-updated policy rather than the current action strategy with explorations.

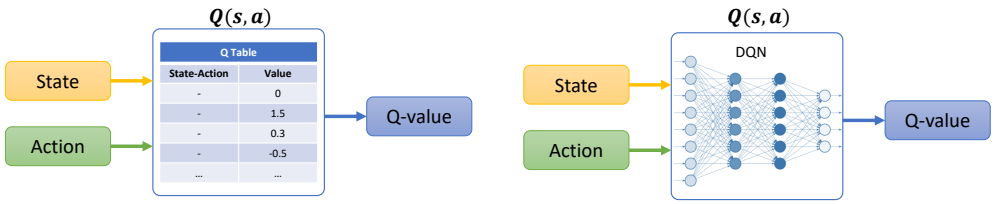
The representation and update rules of learned policy depend on the specific RL method. In the context of Q learning, the policy is represented as a discrete table — Q table. It stores the values for all available state-action pairs and the policy is extracted as the action with the highest Q value. As for the policy update, the bellman equation and off-policy rules are used to update Q values as shown in Eqn (1.3). In summary, we focus on a model-free off-policy reinforcement learning algorithm — Q learning — for physical resource management in this thesis.

$$Q(s,a) = Q(s,a) + \alpha[r + \max_{a} Q(s',a)] \tag{1.3}$$

1.2.3 Deep Reinforcement Learning

Q learning is well-developed for traditional model-free MDP problems with small discrete state-action space. Since the policy is stored in a finite Q table, Q learning cannot directly handle the scenario when state-action space is large or continuous. For instance, the power management problem in Chapter 2 contains continuous action and state space, resulting in an infinity Q table. According to [91], we can utilize deep reinforcement learning (DRL) techniques to address the continuous state-action space issues.

In DRL, a feed-forward neural network is used to approximate an infinity Q table, providing a non-linear mapping from state-action pair to the respective Q value. This neural network is called Deep-Q-network(DQN), which combines traditional Q learning



(a) (b)

Figure 1.5: (a) Policy illustration for Q learning. (b) Policy illustration for Deep-Q-Network.

with powerful deep neural networks [102]. Figure 1.5 illustrates the policy diagrams for RL and DRL. Similar to Q learning, the DQN can be presented as $Q(s, a|\theta)$, where θ is the learnable weights. The policy can be calculated as $\pi(s) = \max_a Q(s, a|\theta)$. The weight parameters θ can be trained using gradient descent optimizing by minimizing the loss function in Eqn (1.4), where $\mathcal{D}$ is the training dataset collected and updated in the online process. The more detailed training process and other techniques in DRL (e.g., replay buffer) are discussed in Chapter 2.

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,a,s') \in \mathcal{D}} (y - Q(s, a|\theta))^2 \tag{1.4}$$

$$y = R(s, a, s') + \gamma \cdot \max_{a \in \mathcal{A}} Q(s', a|\theta)$$

Besides DQN, there are many other DRL algorithms for continuous state-action space, including policy gradient, deterministic policy gradient (DPG), Deep Deterministic Policy Gradients(DDPG), DQN with double q learning, etc [92,110,129,149]. In this thesis, we focus on designing an effective DQN algorithm and other algorithms are left as a future exploration for resources allocation problems in computing systems.

1.2.4 Learning to Optimize (L2O) and Policy Gradient

Besides RL algorithms for MDP, there are other algorithms designed for simple-state MDP, which target at maximizing the total rewards $\sum_{s\in\mathcal{S}}[R(s,a)]$. Typically, these problems can also be considered as general optimization problems. Some optimization problems can be easily solved with commercial solvers (e.g., *scipy.optimize* package) when reward function R is convex and action a is continuous variable. However, it is non-tractable to solve the combinatorial optimization problems, where action a is discrete and reward function R is non-convex. For instance, the workload offloading in vehicular edge computing belongs to the combinatorial optimization problems.

'Learning to Optimize' (L2O) represents a set of algorithms, focusing on learning a solver for the combinatorial optimizations. In L2O, the solver is implemented with a deep neural network (DNN), which provides a direct mapping from optimization parameter (e.g., state s) to the respective solution (e.g., action a). Figure 1.6 shows the general learning architecture for L2O, where the estimated utility(reward) is directly utilized to train the DNN (optimization) model. Note that the optimization can be represented as $a = M(s|\theta)$, where θ is the learnable weights. In L2O, θ is learned with gradient descent by maximizing the utility or reward over the training dataset as shown in Eqn (1.5). In the inference phase, an optimal solution can be directly provided by the optimization model.

$$L(\theta) = \sum_{s\in\mathcal{S}}[R(s, M(s|\theta))] \tag{1.5}$$

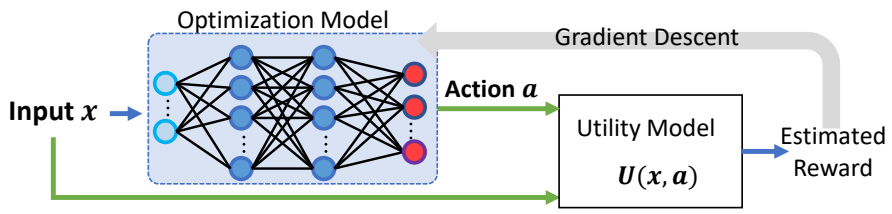


Figure 1.6: General learning architecture for Learning to optimize.

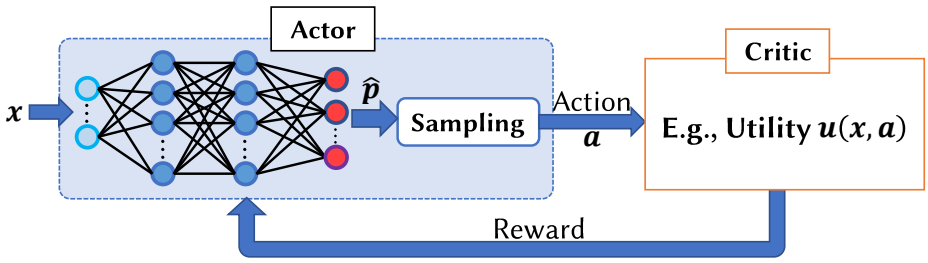


Figure 1.7: Actor-critic architecture with randomly sampling for discrete action space.

Typically, previous L2O architecture is focusing on the continuous optimization problem because the gradient descent requires a differentiable function for back-propagation. However, an innovative method should be considered for discrete action space. In this thesis, we turn to the **policy gradient** method using an *actor-critic* framework. Figure 1.7 illustrates the typical architecture for actor-critic framework, including a policy model and a critic module. Actor-critic is firstly developed for reinforcement learning problem, where the actor and critic model are trained alternately due to state correlation [80]. When convergent, the actor servers a policy model and the critic can be used as a value model. More information about actor-critic for RL problems are described in [80, 116].

In our problem, the critic model can be directly replaced with the utility function without training. Then, the actor model can be trained with policy gradient. According to [139], the training gradient of the actor model can be calculated as Eqn (1.6), where $J(\theta)$ is the loss function and θ is the trainable weights for actor model π_θ. For a faster

convergence, we utilize the **reinforced policy gradient**, by subtracting a baseline value $b(x)$ from $V(x, a)$ [120]. More detailed implementations can be found in Chapter 4.

$$\nabla_\theta J(\theta) = \mathbb{E}_x[(V(x, a) - b(x)) \cdot \nabla_\theta \ln \pi_\theta(a|x)] \tag{1.6}$$

However, the solution from L2O may be not optimal or robust, especially when the parameter x is estimated or measured with errors. Hence, we need a robust solution as discussed before. In Chapter 4, we propose a innovative robust learning for optimization algorithm, called — RobustPN. According to our literature reviews, our work is the first one that solves robust optimization problems with machine learning-assisted methods. Briefly speaking, the critic module is replaced with a robust-critic neural networks, providing worst-case rewards under a specific uncertainty budget. In RobustPN, we first pre-trained the critic module using the traditional L2O method. Then, the actor model is trained using the policy gradient algorithm with feedback from the critic model. Finally, the actor model and the critic model can be combined to provide more robust solutions (polices). More explanations and results can be found in Chapter 4.

1.3 Thesis Contributions

In this dissertation, we focus on resource management in edge computing systems, including physical resource management using reinforcement learning and robust computing resources allocation using *'learning to optimize'* method.

In **Chapter 2**, we target power management in an edge-computing system using reinforcement learning. Specifically, we propose battery-assisted power management

in edge co-location data centers, considering the computing performance and thermal behavior. Due to significant workload fluctuations, edge data centers built in distributed locations suffer from resource underutilization and require capacity underprovisioning to avoid wasting capital investment. The workload fluctuations, however, also make edge data centers more suitable for battery-assisted power management to counter the performance impact due to underprovisioning. In particular, the workload fluctuations allow the battery to be frequently recharged and made available for temporary capacity boosts. But, using batteries can overload the data center cooling system which is designed with a matching capacity of the power system. In this chapter, we design a novel power management solution, DeepPM, that exploits the UPS battery and cold air inside the edge data center as energy storage to boost performance. DeepPM uses deep reinforcement learning (DRL) to learn the data center thermal behavior online in a model-free manner and uses it on-the-fly to determine power allocation for optimum latency performance without overheating the data center. Our evaluation shows that DeepPM can improve latency performance by more than 50% compared to a power capping baseline while the server inlet temperature remains within safe operating limits (e.g., 32℃).

Complementary to power management, we study the vulnerability and thermal attack opportunities in **Chapter 3**. Firstly, we explore the thermal vulnerability from the mismatch between power load and cooling load in edge data centers. Specifically, we consider edge co-location data centers (a.k.a., edge co-locations), proving services for latency-critical applications. In an edge co-location, multiple entities/tenants house their physical servers together, sharing the power and cooling infrastructures for cost efficiency

and scalability. In this part, we discover that the sharing of cooling systems also exposes edge co-locations' potential vulnerabilities to cooling load injection attacks (called thermal attacks) by an attacker which, if left at large, may create thermal emergencies and even trigger system outages. Importantly, thermal attacks can be launched by leveraging the emerging architecture of built-in batteries integrated with servers that can conceal the attacker's actual server power (or cooling load). Then, we consider both one-shot attacks (which aim at creating system outages) repeated attacks (which aim at causing frequent thermal emergencies). For repeated attacks, we present a foresighted attack strategy which, using reinforcement learning, learns on the fly a good timing for attacks based on the battery state and benign tenants' load. We also combine prototype experiments with simulations to validate our attacks and show that, for a small 8kW edge co-location, an attacker can potentially cause significant losses. Finally, we suggest effective countermeasures to the potential threat of thermal attacks.

Besides physical resources (e.g, battery and power), we further investigate the computing resources management problem in **Chapter 4**. In edge computing systems, most resource allocation problems (e.g., workloads assignment) can be formulated as combinatorial optimization problems with discrete actions and predicted optimization parameters, which is NP-hard in nature. Due to the prediction error, a robust solution is required for combinatorial optimization. Hence, we target solving robust combinatorial optimization utilizing machine learning. Firstly, we formulate the robust CO problem as a general minimax optimization problem. To solve the minimax problems efficiently, we propose a novel machine learning-assistant method — RobustPN, leveraging the *actor-critic* platform

and *learning to optimize* technique. To evaluate the performance of RobustPN, we perform simulations on two robust CO problems: a synthesis toy problem and a workload offloading problem with different error budgets for robustness. Our results show that a robust policy (solution) can be retrieved fast and accurately from well-trained policy and critic DNN models, verifying the efficiency and effectiveness of the proposed RobustPN.

To summarize, in this chapter, we briefly introduce the edge computing system and the respect resource management problems, including physical resources (e.g., power and battery) and computing resources. These problems can be formulated as either a Markov decision process (MDP) problem or a combinatorial optimization (CO) problem. Then, Reinforcement learning algorithms are presented for the MDP problems while *'learning to optimize'* methods are discussed for CO and robust CO problems. Finally, the contribution of each chapter is summarized in the last section.

Chapter 2

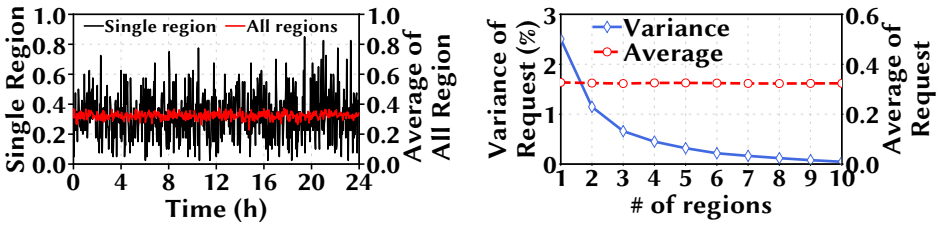
DeepPM: Efficient Power Management in Edge Data Centers using Energy Storage

2.1 Introduction

With the emerging Internet of Things (IoT), 5G network and embedded artificial intelligence, computation workloads are gradually moving from the cloud toward Internet edge [23]. Edge data centers can provide computing services with ultra-low latencies, offering great opportunities for latency-critical applications, such as smart cities, augmented reality and intelligent video acceleration [61, 126]. According to Cisco's report [24], approximately 30% of internet workloads will be processed in edge data centers by 2022.

To achieve user proximity and provide low latency computing services, edge data centers are geographically spread to many locations. However, due to the loss of multiplexing at the aggregate (i.e., random fluctuations canceling out each other), it also results in more workload fluctuations in the edge data center as compared to that of a large centralized one. As an example of typical edge data center workload, we look at Uber's rideshare requests in ten different regions of the Boston area [146]. The rideshare requests in each region in the Uber data set can be seen as the typical workload pattern of an edge data center dedicatedly serving the regional users. Fig. 2.1(a) shows the rideshare request for a single region (Haymarket Square) as well as the average for all the regions normalized to the single region peak. Here we see a staggering variation in regional user requests (i.e., workload) compared to the aggregated of the entire area. Fig. 2.1(b) further shows that the variation in the workload decreases as we combine more regions together.

These rapid workload fluctuations can have a detrimental effect on efficient data center management. It leads to resource underutilization and wasted power and cooling capacities when the edge data center is sized to meet the peak demand [70]. Consequently, resource underprovisioning (i.e., allocating less capacity than the peak demand) has been widely adopted in modern data centers to improve efficiency [12, 43]. However, underprovisioning requires power capping to avoid infrastructure overloads when the demand exceeds the capacity. But, since power capping may also adversely affect the latency performance, we need a graceful dynamic power management to facilitate underprovisioning without severely affecting the response latency [12, 43].



(a) (b)

Figure 2.1: (a) Normalized Uber rideshare requests in a single region vs average of all regions in the Boston area [146]. (b) Change in variance and average number of requests with region aggregation.

In this work, we identify that the rapid fluctuations in edge data center workloads are particularly suitable for employing capacity-constrained energy storage devices such as batteries to temporarily increase the infrastructure capacity during overloads and counter the performance impact due to power capping. More specifically, in edge data centers, the workload spikes are short-lived (Fig. 2.1(a)) which, if done judiciously, allows the battery to be frequently recharged between overload events and be made available for supplementing the data center capacity. The same, however, is not applicable for larger data centers where slowly changing workload results in extended capacity overloads (e.g., tens of minutes) that the battery cannot sustain (i.e., running out of energy due to the lack of recharging opportunity).

In particular, we aim at using the batteries in data center's uninterruptible power supply (UPS) unit which provides backup power during utility power outages [53, 54]. The main idea here is to store energy (i.e., recharge) in the UPS battery when the power demand is low and use (i.e., discharge) it later at a suitable time, for example when the demand exceeds the capacity. While prior works have also studied using UPS battery for data center power management, a key unwanted pitfall of cooling system overload has been

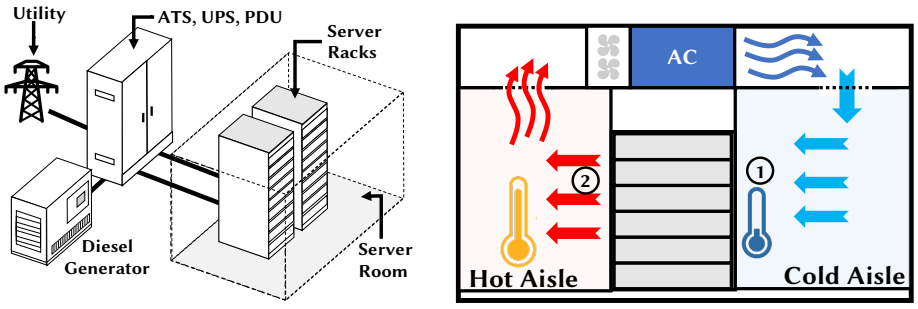
mostly overlooked [53, 54]. Data center cooling system is typically designed with capacity matching that of the power system. Hence, using the extra power from the UPS battery also drives the corresponding server heat generation beyond the cooling system's capacity. When overloaded, the cooling system cannot remove all the heat generated, leading to rapid temperature build-up inside the data center due to heat accumulation. According to [95], the server inlet temperature may increase by more than 10°C if the cooling system is overloaded for 10 minutes. Such cooling system overloads and the ensuing temperature increase can lead to automatic server shutdown to avoid permanent damage and fire hazard. For instance, Dell EMC server will perform a protective shutdown once the server inlet temperature exceeds the threshold of 32°C [37].

Our contributions. In this chapter, we develop a novel power management solution for edge data centers to use UPS batteries to boost capacity while also keeping the data center cool. While we exploit the rapid fluctuations and short-lived spikes of edge data center workloads to utilize UPS batteries, we tackle the cooling system overload by exploiting the *cold air* inside the data center as a heat buffer that absorbs transient heat spikes. Concretely, we use the cold supply air (e.g., at 27°C) within the data center as an energy storage by exploiting its temperature difference from the server safe operating limit (e.g., 32°C) to temporarily hold the extra heat generated due to the battery usage.

However, developing a power management solution that effectively exploits both the data center cold air and UPS battery is challenging. First, both the energy stored in the batteries and the temperature of the cold air have *memories*. That is, using the battery now to supplement the power capacity diminishes the available battery for future

use. Similarly, an increase in cold air temperature due to absorbing heat spike will leave less room for temperature increase to handle future heat spikes without overheating (e.g., temperature exceeding 32°C). Hence, our power management incorporating the battery and cold air needs to consider future demand in its decisions. Second, to allow safe heat spikes due to battery usage, we also need to accurately predict the impact of heat spikes on cold air temperature by extracting the data center's thermal dynamics. However, thermal modeling for every edge data centers individually is impractical due to their large number and diverse physical locations and operating environments. Not to mention, such thermal models need to be updated every time there is a change in the data center environment (e.g., data center/server layout).

To solve the aforementioned challenges, we propose a deep reinforcement learning (DRL) based algorithm — DeepPM. We are motivated to use a DRL based solution since it can autonomously learn and incorporate both the future requirements (e.g., capacity boosts from the battery) and dynamics of the data center environment (e.g., temperature change) in its decisions. Moreover, using DRL's model-free approach we can capture greater details of our problem and overcome simplifying assumptions made in existing model-based data center management approaches [143]. In DeepPM, we formulate the power management problem as a Markov decision process (MDP) where the power demand for incoming workload, battery energy, and cold air temperature at the server inlet constitute different MDP states while the action space is the total power allocation. We formalize a parameterized reward function that penalizes for the increase in latency, cold air temperature, and battery energy usage. We use deep Q-learning with long short-term memory (LSTM) network to



(a) (b)

Figure 2.2: (a) Edge data center power delivery system. (b) Edge data center cooling system with cold and hot aisle containment. ①: server inlet temperature T_{in}, ②: server outlet temperature T_{out}.

learn the optimum action at each MDP state. Our MDP formulation together with the LSTM network allows DeepPM to make a decision based on only the current state (i.e., power demand, battery energy level, and cold air temperature), while the deep Q-learning allows us to learn and utilize on-the-fly the data center thermal dynamics in a model-free manner.

To evaluate DeepPM, we compare it with three other baseline algorithms. Our results show that DeepPM can effectively exploit the data center cold air and the UPS battery to provide more than 50% improvement in latency performance while keeping server inlet temperature within 32°C. We also conduct a sensitivity study to see how different settings affect DeepPM's performance.

2.2 Preliminaries

Power infrastructure. A typical edge data center's capacity ranges from a few kilowatts to a few tens of kilowatts. The data center connects to the power utility and

typically also has a backup generator that serves as a secondary power source. An automatic transfer switch (ATS) reroutes the data center power connection from the utility to the generator during a power outage. However, it may take few tens of seconds to a few minutes to bring the generator online [53]. Hence, to support uninterrupted operation during the power switchover, the data center is equipped with a UPS with battery backup. The server racks get power from a power distribution unit (PDU). For small edge data centers, the PDU can be collocated with the UPS and ATS. Finally, the power the distributed to each server form their respective rack PDUs. We show a generic power infrastructure hierarchy for an edge data center in Fig. 2.2.

Cooling infrastructure. Data centers need dedicated cooling systems to remove the heat generated by servers. Since almost the entire power consumption of the servers converts into heat, data center cooling systems are provisioned with capacity matching the power infrastructure capacity. Due to smaller size, edge data center usually uses a computer room air conditioner (CRAC) as the cooling system. Fig. 3.3 illustrates a typical CRAC cooling system in edge data centers. Here, the CRAC supplies cold air at temperature T_{in} to the server inlet and collects hot exhausting air at temperature T_{out} from the server outlet. To improve CRAC's efficiency, the hot and cold aisle containment can be installed to avoid heat pollution (i.e., hot air mixing with the cold air) [106]. The hot air from the hot aisle is recirculated through the CRAC unit which removes the heat and cool it down to supply cold air to the cold aisle at temperature T_{in}. For improved cooling efficiency, the server inlet temperature is typically conditioned at 27°C, as recommended by ASHRAE [134].

2.3 Power Management using DeepPM

2.3.1 Problem Definition

In this work, we focus on an edge data center hosting multiple server racks, a UPS with battery backup, and a CRAC cooling system. We consider the data center has a total power capacity of C_0. The cooling system capacity is provisioned for the designed power capacity and can supply cold air at temperature T_0 (e.g., 27°C) when the data center power consumption does not exceed C_0. The UPS battery has a maximum recharge rate of R_{max} which is imposed to safeguard against damaging the battery cells. We consider the data center capacity C_0 can be supplemented by discharging the UPS battery using techniques similar to prior work [53, 54].

We use a discrete-time model with a time step Δt (e.g., 10 seconds) where the power management decisions are updated at the beginning of each time step. The server power allocation decision is made based on the power demand from the incoming workloads/requests and the available energy in the battery. Whenever the power allocation is lower than the power demand, the data center utilizes power capping to curb the server power consumption. The decision also takes into account the server inlet temperature to avoid overheating the data center. At time step t, we denote the power demand as p_D^t, the power allocation as p_a^t, battery energy level as b^t, and server inlet temperature as T_{in}^t.

Objective. The target of the data center operator is to dynamically allocate power to improve the data center's overall performance (e.g., latency/response time) without overheating the data center. We formalize the power management as the following

optimization problem **OPA** (**O**ptimum **P**ower **A**llocation).

$$\textbf{OPA}: \underset{p_a^t}{\text{minimize}} \quad \sum_t L(p_D^t, p_a^t) \tag{2.1}$$

$$\text{subject to} \quad T_{in}^{t+1}(p_a^t, T_{in}^t) \leq T_{th} \tag{2.2}$$

$$p_a^t - C_0 \leq b^t \tag{2.3}$$

Here, $L(p_D^t, p_a^t)$ is the total latency increase due to a power allocation p_a^t against a demand of p_D^t and T_{th} is the overheating threshold for server inlet temperature. Constraint (2.2) restricts data center from overheating and constraint (2.3) limits using the battery beyond its current energy. In addition, whenever the power allocation is less than the capacity, the battery is recharged at the rate of $\min(R_{max}, C_0 - p_a^t)$.

Challenges. Solving **OPA** is challenging because the battery energy and the cold air temperature both have *memories*. Specifically, the change in battery energy and/or cold air temperature due to the power allocation decision made in the current time slot affect the available battery and/or heat absorption capacity of cold air in future time slots. Further, constraint (2.2) requires that we estimate the server inlet temperature T_{in}^{t+1} in the next time slot based on power allocation (p_a^t) and server inlet temperature (T_{in}^t) of the current time slot. However, modeling the edge data center's thermal behavior using techniques like computational fluid dynamics (CFD) is exhaustive since it requires CFD analysis of a large number of edge data centers operating in diverse environments. Not to mention the update required every time the data center's environment (e.g., server layout) changes.

In what follows, we formulate our problem using a MDP followed by a deep Q-learning based algorithm to solve **OPA** in a model-free manner.

2.3.2 MDP Formulation

As the foundation of reinforcement learning-based solution, we first model our problem using a discrete-time MDP where the entire time horizon is divided into time slots $t = 0, 1, \cdots, \infty$. Our system state s^t at time t includes the power demand p_D^t, battery state b^t, and the server inlet temperature T_{in}^t while the action is defined by the power allocation p_a^t. The MDP formulation of our problem can be summarized as follows:

- System state: $s^t = (p_D^t, b^t, T^t) \in \mathcal{S}$
- Action: $p_a^t \in \mathcal{A}(s^t)$
- State transition probability: $P(s^t, p_a^t, s^{t+1})$
- Reward function: $r^t = R(s^t, p_a^t, s^{t+1})$
- Discount factor: $\gamma \in (0, 1)$

Here, the action space $\mathcal{A}$ is determined by the current state s^t. The tuple (s^t, p_a^t, s^{t+1}) means that the system transitions from state s^t to s^{t+1} when action p_a^t is taken. State transition probability function $P(s^t, p_a^t, s^{t+1})$ describes the probability that the system state moves from s^t to s^{t+1} given action p_a^t is taken at s^t. The reward function $R(s^t, p_a^t, s^{t+1})$ defines the immediate reward under state-action tuple (s^t, p_a^t, s^{t+1}).

Action Space. In our MDP formulation, we consider a continuous action space for power allocation with two distinct cases. First, when the power demand is below the data center power capacity, we allocate power for the entire demand, i.e., $p_a^t = p_D^t$. Second, when the demand exceeds capacity, we may supplement the power allocation with battery

up to its maximum available capacity, i.e., $p_a^t \leq C_0 + \min(p_D^t - C_0, b^t)$. The eligible action space $\mathcal{A}$ for the current system state s^t can be defined as

$$\mathcal{A} = \begin{cases} p_a^t = p_D^t, & \text{when } p_D^t \leq C_0 \\ p_a^t \leq C_0 + \min(p_D^t - C_0, b^t), & \text{when } p_D^t > C_0 \end{cases} \tag{2.4}$$

State transitions and Markovian assumptions. For our problem, the battery level b^t and server inlet temperature T_{in}^t both evolve based on the power allocation action taken. The battery level b^t perfectly follows the Markovian process since it changes only when the power allocation action p_a^t exceeds the capacity C_0 and requires battery supplement for catering the demand. However, since the temperature change is a slow process, the time granularity Δt in our problem formulation needs to be sufficiently large for temperature changes to take place to satisfy the MDP assumption. Nonetheless, such restriction on Δt can be lifted at the expense of a larger state-action space by using an augmented multi-level MDP where the next state depends on both the current and recently visited states and actions [62].

The changes in the power demand p_D^t, on the other hand, mainly depends on user behavior and may not correlate with the current state and action, thereby violating the MDP assumption. To facilitate the MDP formulation, we consider that the power demand of next time slot p_D^{t+1} is known at time t through workload estimation [33]. This enables our solution to determine the next state based on current state and action. We add an LSTM network in our design to automate the power demand estimation process.

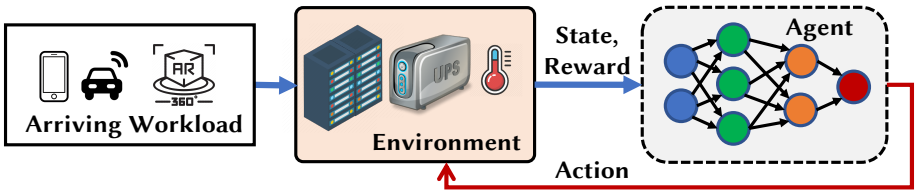


Figure 2.3: Reinforcement learning system: the agent observes system state, makes action, and receives reward from the environment.

Reward function. To attain the optimization goals of **OPA**, we devise our reward function as follows:

$$\begin{aligned} r^t = R(s^t, p_a^t, s^{t+1}) = &- L(p_D^t, p_a^t) - \beta_1 (T_{in}^{t+1} - T_{th})^+ \\ &- \beta_2 (b^t - b^{t+1}) \end{aligned} \tag{2.5}$$

where $(T_{in}^{t+1} - T_{th})^+ = \max(T_{in}^{t+1} - T_{th}, 0)$ is the temperature violation, $b^t - b^{t+1}$ is the battery usage in time slot t, and β_1 and β_2 are wight parameters.

In (2.5) a decrease in latency is positively rewarded as in **OPA**'s optimization objective. Also, since the battery energy and cold air temperature both have memories, we penalize for battery usage and temperature violation to incorporate their impacts on future decisions. In addition to acting as unit conversion coefficients[1], values of β_1 and β_2 can be tuned to change the emphasis of the optimization goal. For instance, larger values of β_1 will be more restrictive of the temperature violation while increasing β_2 will result in more conservative use of the battery. Note that, the power allocation constraints (2.3) is satisfied by our action space $\mathcal{A}(s^t)$. The objective of the MDP problem is to find the optimal action policy $\mathcal{A}^*$ for maximizing the long term reward $\sum_t \gamma \cdot r^t$ with a discount factor $\gamma \in (0, 1)$. The discount factor γ is introduced here to have a tractable problem.

[1]Units of battery level and temperature violation are converted to the unit of latency by multiplying with β_1 and β_2, respectively.

2.3.3 Reinforcement Learning and DeepPM

Reinforcement learning is a widely used approach to solve MDP problems. Q-learning is one of the reinforcement learning algorithms for solving problems with an unknown environment [144,163]. Fig. 2.3 shows the building blocks for reinforcement learning in the context of data center power management. Here, the reinforcement learning agent takes input from the environment to determine its current state which evolves based on the action taken and arriving workload. Next, we briefly discuss Q-learning and then introduce the DRL based power allocation algorithm DeepPM which is implemented with a deep neural network [103].

Q-learning. It is an off-policy reinforcement learning algorithm that can solve model-free MDP problems. In other words, Q-learning can effectively learn the optimal strategy without any prior knowledge of the environment. The learned strategy is represented as a discrete Q value table, which stores Q values for all possible state-action pairs. Then the Q policy π^Q can be extracted as choosing an action with the highest Q value in Eqn. (2.6).

$$\pi^Q(s^t) = \underset{p_a^t \in \mathcal{A}(s^t)}{\operatorname{argmax}} Q(s^t, p_a^t) \tag{2.6}$$

The critical task for Q-learning focuses on the estimation of Q values from the environment response. Typically, the Q values can be trained offline with a fixed-point iteration of Bellman equation (Eqn. (2.7)) for MDP with a known environment. Then, the conventional Q-learning with a learning rate α can be presented as

$$Q(s^t, p_a^t) = Q(s^t, p_a^t) + \alpha[r^t + \gamma Q(s_{t+1}, \pi^Q(s_{t+1}))] \tag{2.7}$$

Other methods (e.g., batched Q-learning [167]) have also been investigated to solve specific model-free MDP problems and accelerate the convergence of Q-learning. Usually, Q-learning works well with a small state-action space. It becomes intractable when the state-action space is large or continuous because of either ultra-large Q table or rarely visited state-action pairs. In our power allocation problem, both the states (p_D^t, b^t, T_{in}^t) and action (p_a^t) occupy a continuous space, resulting in an infinitely large Q table. Since the purpose of the Q table is to provide the Q values for a given action, the Q table can be replaced by a deep neural network that acts as an estimator for the Q values.

DeepPM. The basic idea of DeepPM is shown in Fig. 2.4 where a feed-forward neural network is used to approximate the Q table. The neural network takes the three state parameters (power demand p_D^t, battery state b^t, and server inlet temperature T_{in}^t) and the action (power allocation p_a^t) as the input. Now, as discussed in Section 2.3.2, the power demand p_D^t depends on the incoming user request and may well evolve independently from the state and action. To circumvent the non-MDP nature of the power demand changes, we add an LSTM layer in the feed-forward neural network with a vector h^t that encodes the history states [33]. We initialize h^0 to an all-zero vector. The LSTM layer acts as a predictor for future power demand using history h^{t-1} and allows DeepPM to navigate the state transitions for its actions. In the implementation of DeepPM, we have one fully-connected layer with 500 hidden neurons, one LSTM layer with 50 hidden neurons, one fully-connected layer with 50 hidden neurons and one fully-connected layer with Q value as the output.

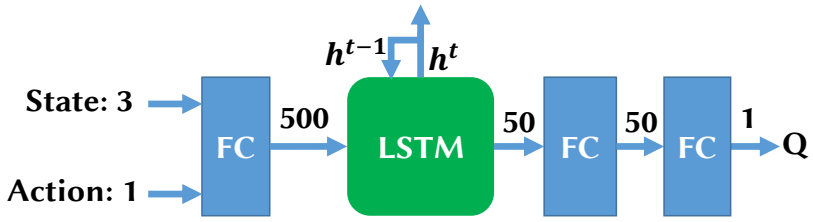


Figure 2.4: DeepPM using deep neural network. Here, FC stands for fully-connected layer, LSTM stands for Long Short-Term Memory cell, and h_t is the history states of LSTM. It takes the state and action as input and provides the corresponding Q value.

Algorithm 1 DQN Training Algorithm

Input: Greedy exploration parameter ϵ, and mini-batch size B, M, and N.

Randomly initialize DQN network $Q(s, p_a|\theta)$ with weights θ.

Initialize an empty replay buffer $\mathcal{R}$.

For Epoch = 1 to M:

Randomly initialize the battery state b^1 and server inlet temperature T^1.

For t=1 to N:

1. Select one power allocation p_a^t based on current system state s^t according to ϵ-greedy policy

   a) randomly select $p_a^t \in \mathcal{A}(s^t)$ with probability ϵ.

   b) otherwise, select p_a^t such that $p_a^t = \text{argmax}_{p_a \in \mathcal{A}(s^t)} Q(s^t, p_a|\theta)$

2. Update battery state b^{t+1} and server inlet temperature T^{t+1} from sensors.

3. Calcuate reward r^t based on Eqn. 2.5.

4. Store the simulation experience $(s^t, p_a^t, r^t, s^{t+1})$ into replay buffer $\mathcal{R}$.

5. Randomly sample B experience from replay buffer $\mathcal{R}$ as the training dataset $\mathcal{D}$.

6. Update the DQN network θ via minimizing total loss $L(\theta)$ in Eqn. 2.8.

Return DQN network with learned weights θ.

The deep neural network is also called Deep-Q-Network (DQN), which can estimate the Q value as $Q(s, p_a|\theta)$, where θ is the weight parameters. Owing to the significant recent developments in deep learning, we can learn the weight parameters θ with high precision using a well-developed gradient descent optimizer (e.g., Adam optimizer [78]). The input of DQN includes the state and action where the output of DQN provides the corresponding Q value. According to (2.7), the loss functions of DQN are defined in (2.8) and (2.9), on training data set $\mathcal{D}$.

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s^t, p_a^t, s^{t+1}) \in \mathcal{D}} \left(y^t - Q(s^t, p_a^t|\theta)\right)^2 \tag{2.8}$$

$$y^t = r^t + \gamma \max_{p_a^t \in \mathcal{A}(s^t)} Q(s^{t+1}, p_a^t|\theta) \tag{2.9}$$

To calculate the network target y^t for each gradient descent training iteration, we need to find the optimal action p_a^t (i.e., power allocation) for new state s^{t+1}. We can calculate the optimal action p_a^t for the discrete action space using, for example, exhaustive comparison. However, continuous action space is challenging to deal with because of its complexity in finding the action to maximize Q values. One straightforward approach is to discretize the continuous action space and apply standard DQN. An alternative approach, called Deep Deterministic Policy Gradients(DDPG), is to use a critic-actor policy [93] with two neural networks – a DQN and a deterministic policy gradient (DPG) network. DQN approximates the Q function $Q(s, p_a)$, with state-action as the input and Q value as the output. Whereas DPG approximates the policy function $p_a = \mu(s)$, with states as the input and actions as the output. DDPG is typically applied with high dimensional continuous action space. Since we have only one continuous action variable p_a, we use the discretiza-

tion approach and split the power allocation into 200 discrete values with reasonably high precision for our problem.

On the other hand, to ensure fast convergence during the DQN training we use experience replay buffer, mini-batch gradient descent, and ϵ-state-action exploration. The experience replay buffer stores all history state-action pairs and can be used in DQN training. Instead of using the entire replay buffer, a smaller subset of transitions is used by the mini-batch gradient descent to minimize the loss in Eqn (2.8). This is done to avoid using strongly correlated transitions which make the training process unstable [33]. In addition, since the LSTM model is being updated continuously, we cannot directly use saved history h^{i-1} for each picked transition i. Instead we use all the historical states leading up to transition i as input to our LSTM model to get the correct h^{i-1} [33, 133]. For action selection, ϵ-greedy exploration policy is utilized to balance the dilemma between exploration and exploitation. Specifically, with a probability of ϵ, a random power allocation $p_a^t \in \mathcal{A}(s^t)$ is chosen. Otherwise, the action is chosen following (2.6).

In our implementation, we initialize an empty replay buffer $\mathcal{R}$ at the beginning of training. Then the agent explores a preset number of state-action pairs with a greedy parameter ϵ at each time slot t, observes the environment responses, and then stores the experiences $(s^t, p_a^t, r^t, s^{t+1})$ into the replay buffer $\mathcal{R}$. In contrast to traditional Q-learning, the agent of DRL updates weight parameter θ with a mini-batch sampling from the replay buffer $\mathcal{R}$. The detail of the DQN learning algorithm is presented in algorithm 1.

2.4 Evaluation Methodology

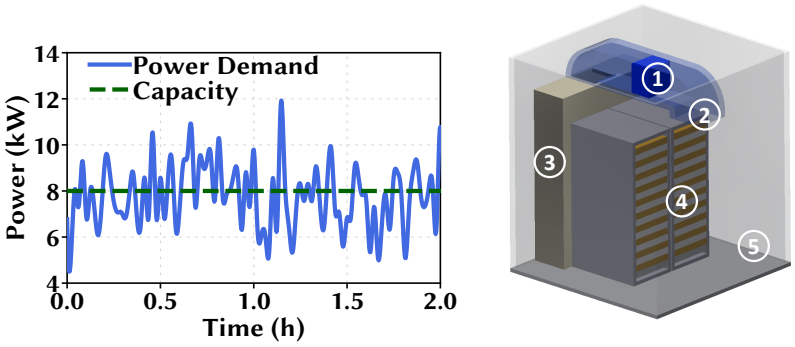
In this section, we present our default edge data center settings, thermal dynamics, battery energy model, and performance model. We then describe the implementation and parameters for DeepPM. We finally introduce three benchmark algorithms for the evaluation of DeepPM.

2.4.1 Settings

Due to limited access to a real edge data center, we resort to a simulation-based evaluation for DeepPM where the power management decisions are updated every 10 seconds.

Data center infrastructure. We consider an edge data center with two server racks each with 20 servers and a designed capacity (C_0) of 8kW. The data center has a UPS backup with battery capacity (C_B) of 0.2kWh which can provide 1.5 minutes of backup power at its maximum discharge rate of 8kW. The maximum recharging rate of the battery (R_{max}) is set to 0.5kW. The data center is cooled using a CRAC (computer room air conditioning) system with a matching designed capacity of 8kW. The CRAC can supply cold air at $T_0 = 27°$C when the data center power consumption remains within its capacity of 8kW. We set the threshold temperature (T_{th}) to 32°C. We also consider the data center employs hot aisle containment to improve cooling efficiency and avoid heat recirculation.

Workload trace. We use real-world traces to perform training and evaluation of DeepPM. We use the number of requests from two popular rideshare applications, Uber and Lyft, in the Boston, Massachusetts area as our workload [146]. Because of their large geographic service areas, customer requests from such rideshare applications are good examples of edge data center workloads. We convert the number of requests to power demand



(a) (b)

Figure 2.5: (a) 2-hour snapshot of workload/power demand trace. (b) CFD simulation model ① Air conditioner (AC). ② Supply air duct. ③ Heat containment. ④ Server racks. ⑤ Server room.

using the server power consumption model of [148] and scale the power trace to have a peak demand of 12kW. Fig. 2.5(a) shows a 2-hour snapshot of the workload trace used in our simulation.

2.4.2 Environment Models

For training and evaluation of DeepPM agent, we simulate the edge data center which is accessed by the DRL for learning the state transitions for its state-action pairs.

Thermal model. We use CFD which is one of the most widely used approaches to analyze the thermal behaviors of data centers (e.g., Google [51]). However, transient CFD simulation is slow and computationally exhaustive. Hence, as outlined in [143], we adopt a short-term CFD approach using Autodesk CFD where the thermal environment is modeled using impulse response by creating power spikes. The 3D model used in Autodesk CFD is shown in Fig. 2.5(b). As an illustration, in Fig. 2.6(a), we show the temperature

change for our data center with a 1.5kW cooling overload for 10 minutes using our thermal model.

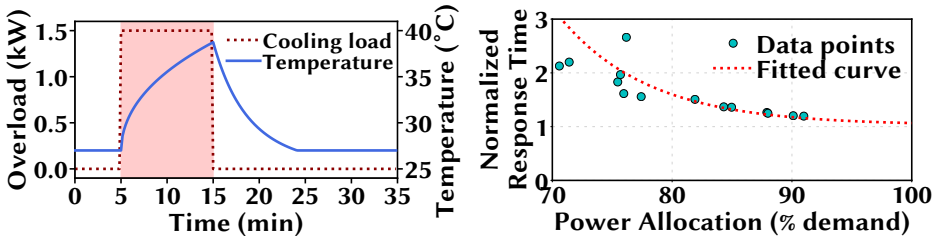
Battery Energy Model We consider a liner battery charging/discharging model where the battery energy state b^t is adjusted by subtracting discharge power and adding recharge power as follows:

$$b^{t+1} = \begin{cases} b^t + \min(C_0 - p_a^t, R_{max}), & \text{for } p_a^t < C_0 \\ b^t - (p_a^t - C_0), & \text{otherwise} \end{cases} \tag{2.10}$$

Note that, a nonlinear battery model incorporating charging/discharging loss and leakage can also be considered in the state transition without loss of generality.

2.4.3 Latency Model

We determine the latency increase due to the power capping based on web search application experiments in [70]. Specifically, we run the web-search benchmark from CloudSuite [45] for different workload levels and CPU speeds (using DVFS) and collect the 99-th percentile response times.Then, considering the power consumption at full CPU speed as the power demand for a given workload, we model the performance impact (change in response time/latency) for change in the power allocation as a percentage of the power demand. We show the performance for different power allocations in Fig. 2.6(b) where the response time is normalized to that of without any power capping. Note that, here we use web-search as an example of performance-power trade-off while many other applications also have similar performance-power relationship [70]. Hence, our proposed approach can be used for a wide range of applications.



(a) (b)

Figure 2.6: (a) Thermal model illustration with 10-minute cooling overload. (b) Latency model for partially satisfied power allocation.

2.4.4 DRL Parameters

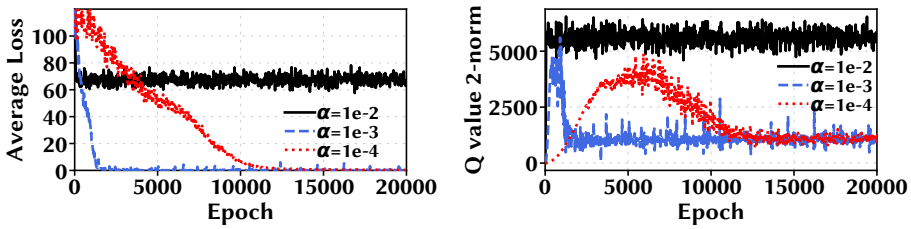
For DRL, we implement the DQN training algorithm using *"Tensorflow"* [39]. The DQN ($Q(s, a|\theta)$) is implemented with a four-layer fully-connected feed-forward neural network, which includes 500 nodes in the first hidden layer, and 50 nodes in the second and third hidden layer. The *"ReLU"* is utilized as the activation function for the three hidden layers to achieve nonlinearity.

For the training process of DQN, we use Adam optimizer with a learning rate of $\alpha = 0.001$, and the mini-batch size is set at $B = 1024$. The discount factor is set at $\gamma = 0.9$. The weight factors in reward Eqn. (2.5) are set as $\beta_1 = 100$ and $\beta_2 = 0.1$ for the penalty of overheating and battery usage, respectively. We perform 6000 training epochs for DQN.

2.4.5 Benchmark Policies

We evaluate DeepPM against three benchmark policies – PowerCap, Greedy, and GreedyT, described as follows.

PowerCap. It does not utilize the UPS battery. When demand exceeds the capacity, it caps the power at capacity (i.e., $p_a^t = C_0 < p_D^t$). Otherwise, it allocates power for



(a) (b)

Figure 2.7: Convergence curve for DQN training: (a)Average training loss. (b) 2-norm of Q values.

the full demand (i.e., $p_a^t = p_D^t$). In our evaluation, PowerCap is the baseline policy since it does not employ any performance improvement strategy such as utilizing the UPS battery.

Greedy. It uses the battery greedily without considering its impact on temperature. When demand exceeds the capacity, it uses the battery to cover the deficit amount up to the available energy in the battery, i.e., $p_a^t - C_0 \leq b^t$.

GreedyT. It has the same power allocation strategy as Greedy, except, it only uses the battery if he server inlet temperature is less than 31.5°C.

When the power demand is lower than the capacity, both Greedy and GreedyT satisfy the demand. They also recharge the battery with the unused capacity, $C_0 - p_D^t$.

2.5 Evaluation Results

In this section, we present results from DQN training and compare DeepPM with the three benchmark algorithms. Then we conduct sensitivity studies on DeepPM.

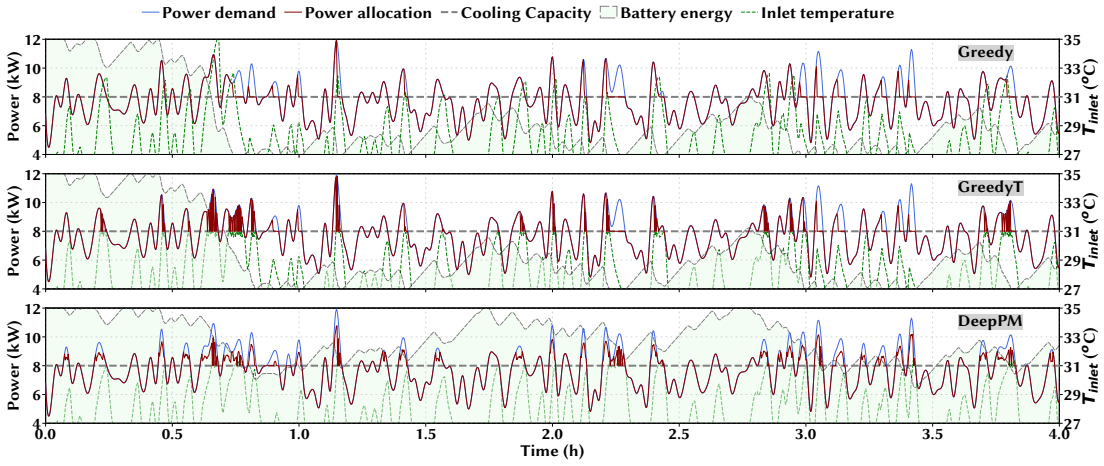
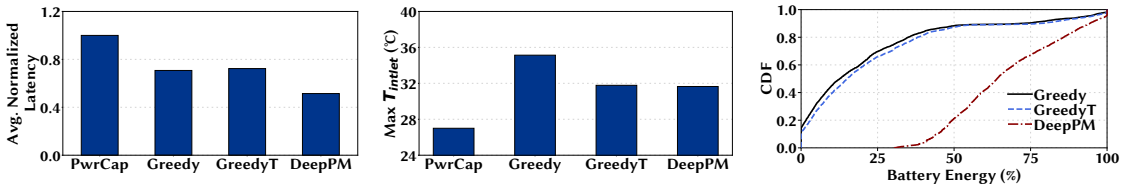


Figure 2.8: Policy illustration: 4-hour snapshot of workload allocation and environment response.



(a) (b) (c)

Figure 2.9: Performance evaluation with different algorithms.

2.5.1 DQN Training

We use the Uber and Lyft workload traces and utilize the thermal model to train DeepPM for 20,000 epochs ($\sim$ 55 hours or little over 2-days with 10 seconds time slots). We use three different learning rates: $\alpha = 0.01$, $\alpha = 0.001$, and $\alpha = 0.0001$ with a mini-batch approach. We sample the average loss for every ten epochs and show the convergence curves of in Fig. 2.7(a). We also show the evolution of 2-norm of the Q values in Fig. 2.7(b). We can see that while training does not converge for $\alpha = 0.01$, it converges after $\sim$2000 epochs

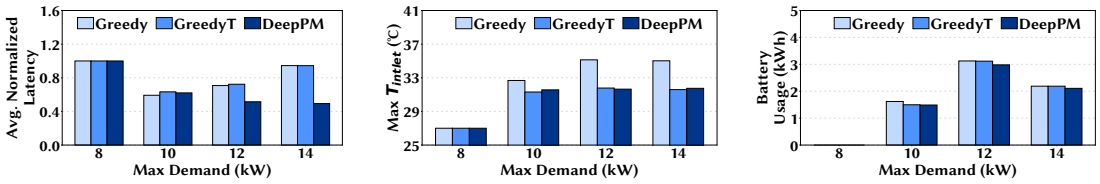
($\sim$ 6 hours) for $\alpha = 0.001$ and $\sim$13,000 epochs ($\sim$ 36 hours) for $\alpha = 0.001$. We chose $\alpha = 0.001$ as our default learning rate since it results in a earlier convergence.

2.5.2 Illustration of Power Allocation

We show a 4-hour snapshot of the power allocation under the three different policies (Greedy,GreedyT, and DeepPM) in Fig. 2.8. We include the power demand due to the incoming workloads, the power allocation, battery energy level, and server inlet temperature variation. The snapshot starts with a fully charged battery with 100% energy and a server inlet temperature of 27°C. Whenever the power allocation is above the capacity of 8kW, the data center uses its battery power and the server inlet temperature goes up due to the extra heat generated beyond the cooling capacity. A gap between demand (blue line) and allocation (red lines) indicates that the power allocation does not meet the power demand, resulting in processing latency.

First, we look at Greedy policy which allocates power from battery whenever the demand is higher than the capacity. It disregards the temperature increase and ends up with server inlet temperature going over 32°C many times (e.g., multiple times near 0.5 hours). For DeepPM, on the other hand, we see that it does not allocate power for the full demand when the server inlet temperature approaches the threshold of 32°C, for example near 0.5 hours. Restricting itself within the temperature limit is the cause of a slight increase in latency performance for DeepPM.

We also see in Fig. 2.8 that the battery energy is frequently replenished due to the rapid fluctuations in power demand. Also, DeepPM uses the batteries more conservatively



(a) (b) (c)

Figure 2.10: Impact of peak power demand on DeepPM.

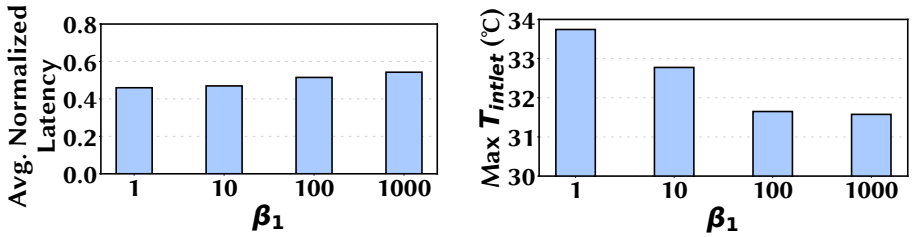
and can maintain a high battery level most of the time while Greedy nearly depletes it around 0.5 hours.

2.5.3 Performance Evaluation

Here we evaluate DeepPM for a period of 24-hours with a trained DQN discussed in Section 2.5.1.

Average Latency. We calculate the latency following our latency model in Section 2.4.3 and normalize to that of PowerCap. We take the average latency performance for the overload time slots to emphasize the true impact of DeepPM. We show the average latencies for the four policies in Fig. 2.9(a). We see that DeepPM, as compared to the baseline PowerCap, achieves more than 50% lower latency. Greedy and GreedyT also enjoy over 30% performance improvements because they use the battery. Nonetheless, both have a significantly less improvement since they greedily consumes the limited battery capacity, as opposed to DeepPM which takes the future needs into account.

Server inlet temperature. Next, we look at the maximum server inlet temperature resulting from the different policies in Fig. 2.9(b). We see that both DeepPM and GreedyT manage to maintain a server inlet temperature below the threshold limit of 32°C



(a) (b)

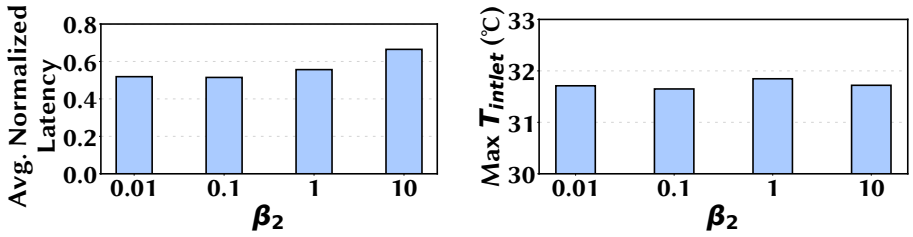
Figure 2.11: Effects of weight parameters β_1.

as opposed to Greedy which results in a maximum inlet temperature of more than 35°C. This is because Greedy does not take the temperature into account during power allocation. PowerCap, on the other hand, has the lowest inlet temperature as it never uses the battery.

Battery usage. Next, we look at how the three battery using policies utilize the battery. We show the CDF of battery energy levels in Fig. 2.9(c). We see that nearly 20% of the time both Greedy and GreedyT run without any available battery to supplement the power allocation. DeepPM, on the other hand, barely drops below a 40% energy level and therefore retains the battery energy to use when the power demand exceeds the data center capacity

2.5.4 Sensitivity Analysis

Here we examine the impacts of peak power demand and weight parameters in the reward function on DeepPM.



(a) (b)

Figure 2.12: Effects of weight parameters β_2.

Impact of power demand

We vary the peak power demand from 8kW to 14kW by scaling the data center dynamic power. We keep the data center power and cooling capacity at 8kW and the battery capacity at 0.2kWh. The results are shown in Fig. 2.10 where we omit PowerCap which does not use the battery.We see in Fig. 2.10(a) that at power demand of 8kW, the latency of all three benchmark algorithms are the same as PowerCap since there is no need for a capacity boost. However, as the demand increases, using the battery becomes useful in improving the latency performance. In Fig. 2.10(b) we see that the maximum server inlet temperature increases as the peak demand increases while both DeepPM and GreedyT maintain a temperature below 32°C. Finally, Fig. 2.10(c) shows that at 8kW peak demand (i.e., no underprovisioning) no battery is used while the battery usage increases with peak demand. Interestingly, the battery usage goes down for 14kW peak demand. This is because of the limited recharge opportunity due to higher power demand.

Impact of weight parameters β_1 and β_2

The weight parameters in DeepPM's reward function (2.5) play important roles since they determine how much DeepPM emphasizes on using the battery and violating the

temperature constraint as opposed to allowing an increase in latency due to power capping during overloads.

First, we vary β_1 from 1 to 1000 while β_2 is kept constant at its default value 0.1. Since β_1 determines the relative weight of the data center temperature violation, a lower value allows DeepPM to violate the temperature constraint more and vice versa. Consequently, we see in Fig. 2.11(b) that the the maximum inlet temperature increases with decreasing β_1. Likewise with the temperature violation constraint relaxed at lower values of β_1, DeepPM uses more battery and can result in lower latency (Fig. 2.11(a)).

Next, we vary β_2 from 0.01 to 10 while β_1 is kept constant at its default value of 100. Since, β_2 is the weight for battery usage, increasing β_2 results in a decrease in the battery usage and vice versa. We see in Fig. 2.12(a) that the latency performance β_2 has marginal impact of latency performance. On the other hand, as shown in Fig. 2.12(b), the decrease in battery usage with the increase in β_2 leads to reduction in server maximum inlet temperature.

The take away from our evaluation is that, *edge data center operation can be supplemented with UPS batteries for significantly improving performance by exploiting the rapid fluctuations in workloads and cold air as a thermal buffer.*

2.6 Related Work

Data center management. Managing the data center infrastructure with limited resources has received significant attention from the research community in the past decade. Various techniques have been proposed to aid data center management such

as improving the energy proportionality [94, 101], jointly managing servers and non-IT support infrastructure (e.g., power/cooling) [94, 101], and exploiting geographical diversity to minimize data center operation cost [113]. Likewise, infrastructure oversubscription is exploited in other works to improve the data center utilization [165]. [97, 176] use energy storage devices to temporarily increase the data center power capacity. Such performance-boosting techniques allow the power consumption to temporarily exceed the data center capacity to offer a performance lift. Constrained by the thermal design power (TDP), [42] proposes temporary power/performance boosts at the microprocessor level by utilizing phase change material and heat absorption of thermal packages. As opposed to prior works, we focus on emerging edge data centers and, instead of exploiting microprocessor-level thermal inertia, utilize the data center level thermal mass in coordination with the UPS battery and propose a DRL-based solution.

Reinforcement learning for resource management. The autonomous learning and online decision capability make reinforcement learning a prime choice for solving data center resource management problems. Due to the recent advances in deep neural network-based learning, recent works focus on DRL based approaches. For instance, [172] proposes a DRL based algorithm to schedule compute-intensive workloads for energy minimization, while [22] designs a task scheduler and resource provisioning system for large cloud service providers. In [98], DRL is used for the job and virtual machine allocation in the cloud data center and [33] uses DRL to capture user behavior for profit maximization of a cloud service provider. Our key novelty is that we focus on edge data centers with rapidly

fluctuating workloads, for which energy storage – both thermal and battery – is exploited for performance maximization subject to capacity constraints.

2.7 Conclusion

In this work, we developed a novel power management algorithm, DeepPM, for edge data centers that exploits the data center cold air and workload fluctuations to use UPS batteries for capacity boosting without overheating the data center. We used DRL to estimate the data center thermal behavior in a model-free manner and utilized it for deciding power allocation to improve latency performance while keeping server inlet temperature within safe operating limits. We showed that DeepPM can achieve a performance improvement of more than 50% compared to the power capping baseline.

Chapter 3

Heat Behind the Meter: A Hidden Threat of Thermal Attacks in Edge Colocation Data Centers

3.1 Introduction

In the wake of the Internet of Things and ubiquitous computing demand, edge computing has recently emerged as a game-changing paradigm that brings computation to the Internet edge, thereby enabling ultra-low latencies for many critical applications such as augmented reality and assisted driving [127]. Consequently, the rise of edge computing spurs the burgeoning development of multi-tenant edge colocation data centers (a.k.a., edge colocation). An edge colocation is a small-scale shared colocation data center built at numerous distributed locations for hosting latency-ultrasensitive workloads such as assisted

driving [28]. In such a colocation, the operator provides power and cooling resources to multiple entities (i.e., tenants) for housing their own physical servers. Thus, this fundamentally differs from a multi-tenant cloud platform where users/tenants share the cloud resources without *owning* the physical servers.

Edge colocations have become the preferred choice for edge service providers. For example, Vapor IO, an edge colocation operator, is rolling out thousands of edge colocations in partnership with wireless tower companies [150]. Moreover, a recent Uptime Institute survey [147] shows that more than 75% of the respondents will use edge colocations to house their physical servers and deploy edge applications.

The criticality of hosted applications, such as assisted driving [150], clearly mandates a high level of security for edge colocations. While securing servers and networks from cyber attacks remains a key issue, recent research has also identified critical vulnerabilities in data center physical infrastructures. More concretely, the practice of infrastructure oversubscription exposes data centers to well-timed power load attacks that aim at overloading the power capacity and compromising the data center availability [68, 69, 71, 169]. Likewise, data center cooling system removes server heat to avoid overheating and hence is also crucial for service uptime. If not properly managed, malicious workloads can create more hot spots that expose servers to an adverse thermal environment and thus more thermal emergencies [48]. Importantly, cooling system has emerged as a leading root cause for downtime incidents in state-of-the-art data centers (e.g., Microsoft's) [73, 111].

To meet the power capacity constraints and avoid outages [67, 158, 165], the colocation operator has power meters to continuously monitor tenants' server power usage.

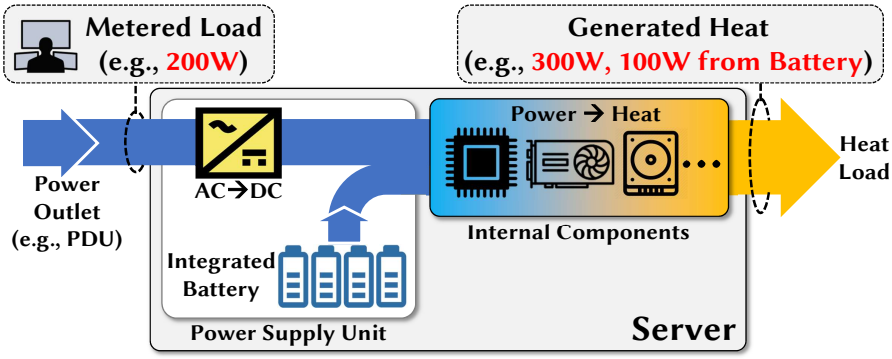


Figure 3.1: An attacker uses its built-in batteries to stealthily inject additional heat to overload the cooling system.

Meanwhile, power meters are also used as a proxy to measure servers' cooling loads,[1] ensuring that the designed cooling capacity is not violated. The reason is that nearly 100% server power is eventually converted into heat or cooling load [69,100,130]. Therefore, with proper heat dissipation, meeting the power capacity constraints also implicitly means meeting the cooling capacity constraints [100].

Contributions. In this part, we study an under-explored threat of thermal attacks — injecting additional cooling loads to overload the cooling system — in an edge colocation. While edge colocations have been generally considered as secure due to tenants' full control of their own servers, we discover that the way tenants' cooling loads are measured (i.e., using power meters as proxies) is potentially vulnerable to thermal attacks. More concretely, as illustrated in Fig. 3.1, an attacker can tap into the emerging architecture of built-in battery units and generate additional cooling loads (i.e., heat), yet without violating the power capacity enforced by the colocation operator. If left neglected, successful thermal attacks may create significant damages: (1) service outage for benign tenants due

[1]Heat generated by servers is "cooling load" for the cooling system.

to overheating (which we call *one-shot* attack); or (2) more frequent thermal emergencies that result in tenants' performance degradation (which we call *repeated* attacks). While various defenses (e.g., measuring servers' outlet temperatures and air flows) are readily available, they have yet to be included in standard practices for many data centers. As such, despite non-trivial efforts needed by thermal attacks, our study serves as a precaution for strengthening cooling system management in edge colocations.

A common practice in today's colocations is to tightly monitor tenants' power usage as well as their server inlet/outlet temperature. Nonetheless, if other effective defense mechanisms (in Section 3.7) are not properly implemented, built-in batteries integrated with servers' power supply units can assist an attacker with launching thermal attacks that are difficult to trace. To provide better energy efficiency and reliability [6, 81, 137], vendors have begun to integrate built-in batteries with servers' power supply units (e.g., Supermicro BBP [137]). Such built-in batteries can conceal the attacker's actual cooling load from the operator's power meters — by discharging built-in batteries to supply additional power, the attacker's servers can consume more actual power and hence generate more heat than the operator measures using power meters. Moreover, this additional cooling load may not be promptly pinpointed by only monitoring the servers' inlet and outlet temperatures. Consequently, indiscernible additional cooling loads can be injected by an attacker to exceed the shared cooling capacity, thus triggering thermal emergencies. While we focus on edge colocation data center, such thermal attacks may be mounted against larger colocation data centers as well, albeit the attacker needs to commit more resources.

Meanwhile, before automatic system shutdown [67, 77], handling thermal emergencies require tenants' power/cooling load reduction through clock rate throttling and/or workload re-routing to other unaffected data centers, which can adversely affect tenants' performance in terms of application response time.

While successful thermal attacks can create an adverse environment for hosting servers in edge colocations, they need non-trivial efforts. As a prerequisite for a good timing, the attacker needs to estimate benign tenants' power/cooling loads based on a voltage side channel [68]. For a one-shot attack with the goal of shutting down an entire edge data center, the attacker can install a large built-in battery and inject sufficient cooling loads continuously, resulting in overheating and triggering automatic system shutdown. For repeated attacks that aim at benign tenants' performance degradation, the attacker needs to repeatedly trigger thermal emergencies by charging and discharging its battery at appropriate times. We propose a foresighted policy based on batch Q-learning that learns on the fly a good timing for repeated attacks based on the battery state and benign tenants' load: thermal attacks are launched only when both the benign tenants' loads are sufficiently high and the remaining battery energy is more than a threshold.

We run prototype experiments to validate the potential threat of thermal attacks. To evaluate the effectiveness of our proposed repeated attack strategies, we run year-long simulations based on computational fluid dynamics (CFD) analysis. Our results demonstrate that for an 8kW edge colocation, an attacker subscribing 10% of the capacity can cause thermal emergencies for more than 3% of the year, degrading benign tenants' performance. Finally, while the existing practices may render edge colocations vulnerable,

battery-assisted thermal attacks can be fairly easily detected and nullified using a reasonable amount of efforts. We discuss such defense strategies in Section 3.7.

In conclusion, while batteries have been exploited (such as for smoothing power demand [157]), our study makes a novel contribution by leveraging servers' built-in batteries for an under-explored malicious purpose — thermal attacks that can potentially result in service outage or performance degradation in edge colocations — and serves as a precaution despite its futility when proper defensive measures are enforced.

3.2 Preliminaries on Edge Colocations

Colocations represent a critical segment and account for nearly 40% of the total energy consumption by data centers [67], serving almost all industry sectors. To complement their own megascale data centers that are typically built in rural areas, even top-brand companies like Google and Microsoft rely on third-party colocations for better performances due to close proximity to end users [141]. Importantly, in the context of edge computing, colocations play an even more crucial role, as it is not economical for individual companies to fully manage small-scale data centers in numerous locations [29].

The data center capacity includes both power and cooling capacities. Power capacity is quantified by the amount of UPS-protected power (a.k.a. critical power) that is delivered to the servers, excluding other power consumption such as UPS power losses and cooling system power. As nearly 100% server power consumption (except for fan power) is converted into heat or *cooling load*, the cooling system capacity is often sized based on the colocation's power capacity and usually also measured in kilowatt [100, 130, 131]. The data

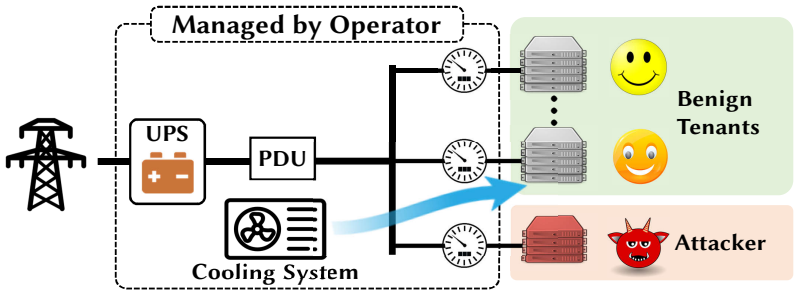


Figure 3.2: An edge colocation data center with an attacker.

center design may also leave some "headroom" in the cooling capacity to handle, if any, irregular heat generation and/or hot spots due to certain servers generating more heat than expected. In such cases, the cooling system utilization may sometimes still be high because of the increasingly common practice of power oversubscription for capital cost saving in modern data centers (e.g., Facebook aggressively oversubscribes its power capacity by 47% on average) [85, 99, 119, 165].

The colocation operator provides non-IT infrastructure support (i.e., power and cooling systems), while each tenant brings and controls its own physical servers.[2] The non-IT infrastructure is expensive and/or time-consuming to construct, taking nearly 60% of the total cost of ownership over a 10-year lifespan for a colocation operator [67, 130, 157]. Thus, like network bandwidth, the operator's power and cooling infrastructure capacity is a limited resource carefully sized based on the tenants' demand.

3.2.1 Power Infrastructure

[2] A tenant can share fraction of a rack space with other tenants.

As illustrated in Fig. 3.2, typically, an edge colocation data center uses a tree-type power hierarchy with total capacity in the range of a few kilowatts to a few tens of kilowatts shared by multiple tenants. Utility power first enters the data center through an uninterruptible power supply (UPS). Then, the UPS-protected power goes into a power distribution unit (PDU), which distributes the power to its downstream servers.

3.2.2 Cooling Infrastructure

While various cooling methods (e.g., computer room air conditioner, chiller, and "free" outside air cooling) are available [40], an edge colocation usually uses a computer room air conditioner to remove servers' heat due to its small size and often rugged deployment (e.g., outdoor with a wireless tower). Fig. 3.3 illustrates a typical cooling system in an edge colocation. For the best cooling efficiency, today's edge colocations also implement hot/cold aisle containment to prevent the hot air from mixing with the cold air [29, 52].

There are four different notions of temperature in a data center: supply air temperature T_{sup}, server inlet temperature T_{inlet} (i.e., temperature of cold air entering a server), server internal temperature $T_{internal}$ (e.g., CPU temperature), and server outlet temperature T_{outlet} (i.e., temperature of hot air exiting a server). With heat containment installed, all the servers' inlet temperature is nearly identical to the supply air temperature. Thus, supply air temperature and server inlet temperature are the lowest and baseline, whose increase will lead to increases in server internal and outlet temperatures. Server outlet temperature is typically elevated by 10+°C compared to the inlet temperature while server internal temperature is the highest and regulated by servers' internal fans. Hence, with

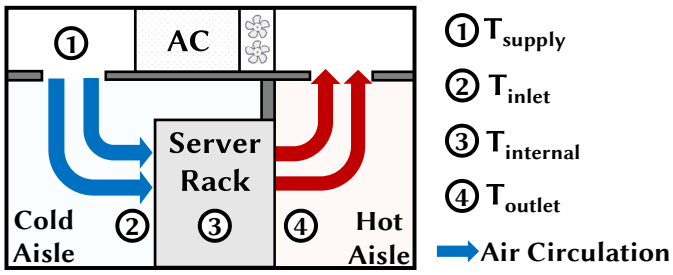


Figure 3.3: Overview of a cooling system in an edge colocation.

heat containment, we have the following [104, 109, 142]:

$$T_{inlet} \approx T_{sup} < T_{outlet} < T_{internal}. \tag{3.1}$$

In a data center, *server inlet temperature is the most important thermal metric* [134, 142], because servers' internal temperature control uses the inlet temperature as a reference [30]. For example, in modern data centers, server inlet temperature is conditioned at 27°C for cooling efficiency, as recommended by ASHRAE [5, 134]. Also note that, while server heat is responsible for increase in internal and outlet temperatures, neither $T_{internal}$ nor T_{outlet} is a reliable indicator for a server's cooling load since they depend on the server's internal heat management (e.g., fan speed) and air flow rate.

3.3 Thermal Attack

The main focus of this section is to present the potential threat of battery-assisted thermal attacks (when concealed cooling loads are *behind the meter* and not promptly detected) and help strengthen edge colocations. As a precursor, we first introduce our threat model that outlines the scenario considered for thermal attacks. We then present the

potential impacts on edge colocations. Finally, we introduce two possible attack strategies followed by discussions on their feasibility.

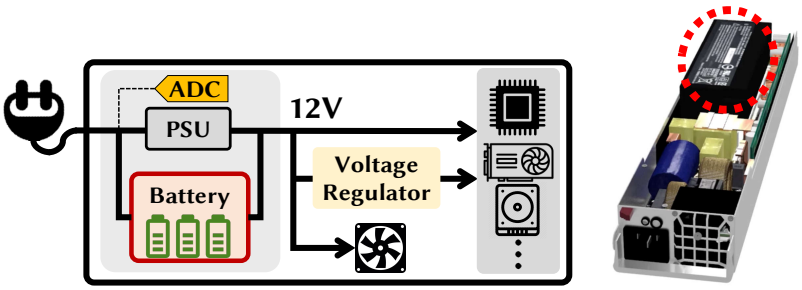
3.3.1 Threat Model

We consider an edge colocation data center with a total power/cooling capacity of C, housing a few racks of servers owned by multiple tenants. There exists a malicious tenant (i.e., attacker) that runs artificial workloads without real values and has bad intentions.

What the attacker can do. The attacker houses its own physical servers in the edge colocation, sharing the power and cooling infrastructures with benign tenants. As illustrated in Figs. 3.1 and 3.4(a), the attacker's server power supply units has built-in battery units, which can conceal the attacker's actual server power/cooling load from the operator's power meters. Fig. 3.4(a) shows an overview of the attacker's server.

The attacker subscribes a data center capacity of c_a from the colocation operator and keeps its power drawn from the operator's PDU below c_a at all times (even during an attack), in order to meet the operator's requirement.

When launching a thermal attack, the attacker runs power-hungry applications (e.g., intensive computation) to increase its actual server power consumption to $p_a > c_a$ where c_a amount comes from the operator's PDU and the rest from its built-in battery. In practice, when running at the peak load, a single server equipped with multiple CPUs and/or GPUs can easily consume several hundred watts, even more than 1kW [17]. Thus, the attacker can inject an additional cooling load of $p_b = p_a - c_a$ beyond its subscribed capacity by discharging built-in battery units (which can be achieved by a dual-source



(a) (b)

Figure 3.4: (a) Attacker's server with built-in batteries. (b) Supermicro's power supply [137]. The built-in battery module is highlighted in a red circle.

power supply that can simultaneously draw power from the PDU and the battery units [55, 75, 97, 156, 157].)

The attacker uses a voltage side channel, as proposed in [68], to estimate benign tenants' real-time total server load with a high accuracy (Fig. 3.5(b)).

What the attacker cannot do. We do not consider naive attacks, such as self-explosion and tampering with the physical infrastructures, which are beyond the scope of our work. Moreover, other attacks, such as network DDoS attacks, are also orthogonal to our focus.

3.3.2 Impact of Thermal Attacks

Although non-trivial efforts are needed in the threat model, a successful thermal attack can overload a data center's cooling system and possibly increase the server inlet temperature to a dangerous level, triggering frequent performance degradation and even system outages [30, 96].

Performance degradation

Before system shutdown, a preventative mechanism is to temporarily cap the data center-wide cooling load (i.e., server power) below the cooling capacity [64, 77]. Specifically, when the server inlet temperature exceeds a threshold (e.g., 32°C) for a certain amount of time [77], it is considered that a data center exception, called *thermal emergency*, has occurred and servers are forcibly put in a low power state. The wait-time between inlet temperature violation and thermal emergency declaration depends on operator's risk management policy. The temperature threshold for a thermal emergency is set lower than the server's automatic shutdown temperature to proactively handle an emergency. For example, in a Google-type data center, disk speeds and/or CPUs are throttled to lower the server power load (i.e., cooling load) in the event of a thermal emergency [77, 114]. Similar mechanisms also exist in multi-tenant colocations to handle a thermal emergency. Concretely, without controlling tenants' servers, the operator sends signals to tenants' own server management systems such that tenants can cap power loads below a certain level (a.k.a. power capping). The actual amount and duration of power capping can be either pre-determined based on SLA terms [5] or decided at runtime through a dynamic coordination mechanism [67, 174].

Nonetheless, handling a thermal emergency by capping tenants' server power (through, e.g., CPU throttling) inevitably results in performance degradation, which can in turn cause user dissatisfaction, revenue loss, and/or SLA violation [6, 67, 155, 165]. Some workloads may be re-routed to other unaffected data centers for service continuity, but this

comes at a higher latency since otherwise those workloads would have been processed in the preferred site to achieve the best performance without being re-routed.

System outage

In order to prevent permanent hardware damage, if the server inlet temperature continues rising despite cooling load capping, automatic system shutdown may occur, leading to a system outage (e.g., the shared PDU can power off when the inlet temperature reaches 45°C) and service interruptions [134]. Such system outages can cause loss of working data sets, and also suffer from long restart waiting time. Financially, a system outage can cost thousands of dollars every minute [111]. For latency-critical applications, an outage event may cause even more catastrophic consequences such as decreased safety in edge-assisted driving [10].

We also run a prototype experiment to demonstrate the potential impact of thermal attacks on benign tenants, and the results are in Appendix 3.3.5.

3.3.3 Attack Strategies

We introduce two possible strategies for battery-assisted thermal attacks with different goals.

One-shot attack. It aims at creating a system outage by increasing the server inlet temperature beyond the safety limit (e.g., 45°C [134]). It can also be coordinated across multiple edge colocations for a wide-area service interruption. Even successfully launched only once, the caused damage may be significant, especially for safety-critical applications (e.g., edge-assisted driving) [10].

Repeated attacks. Instead of aggressively overheating and shutting down the entire edge colocation, repeated attacks aim at frequently degrading performance of benign tenants' latency-sensitive applications over a long period (e.g., one year) by triggering thermal emergencies and cooling load capping. Thus, repeated attacks compromise the long-term cooling system availability in edge colocations.

In general, one-shot attack requires a higher battery capacity to support more intense attack loads (which may still be feasible as shown in Section 3.6). On the other hand, repeated attacks require relatively less (still a considerable amount of) resource, but they require more sophisticated timing of the attacks and can be easy to detect.

3.3.4 Feasibility of Thermal Attacks

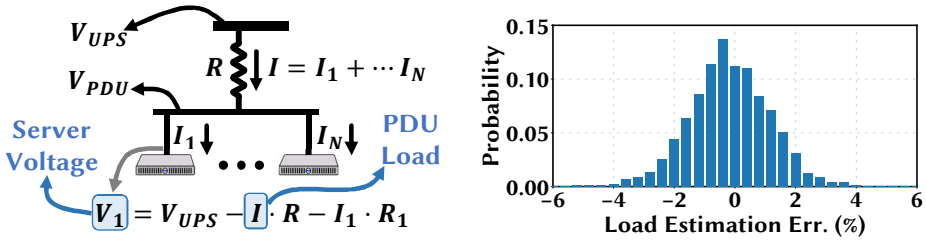
Motivation for thermal attacks. One-shot attack is as motivating as traditional DDoS attacks, as it can potentially create service outages. Likewise, repeated attacks can result in frequent performance degradation for latency-sensitive applications, which in turn causes user dissatisfaction, revenue loss, and/or SLA violation. Thus, although the cost barrier is non-trivial, battery-assisted thermal attack might still be inviting for potential attackers, such as the target colocation's ill-intentioned competitor or state-sponsored attackers.

Attacker's malicious cooling load. In recent years, vendors have integrated built-in batteries into servers' power supply units as an emerging backup power solution (e.g., Supermicro BBP [137] shown in Fig. 3.4(b)). Thus, an attacker can discharge built-in batteries to supply additional power to its servers, generating malicious cooling loads without being monitored by the colocation operator's power meters. Moreover, without air

flow meters, temperature sensors that only monitor server inlet/outlet temperature cannot reliably locate the malicious cooling load. Consequently, if left neglected, thermal attacks can be launched *behind* the meter. This is also illustrated in Fig. 3.1: an attacker generates 300W cooling load, but the colocation operator only measures 200W from the power meter and the additional 100W load is supported by the attacker's internal batteries.

Availability of off-the-shelf hardware. Servers with built-in batteries are commercially available (e.g., Supermicro [137]). The current battery energy density is enough to fit into servers and supply sufficient additional power to mask the attacker's malicious cooling loads [4], even for an one-shot attack that requires more attack loads than repeated attacks. Moreover, servers with large peak-to-average ratios are also available for generating a large amount of heat during an attack. For example, Dell manufactured PowerEdge R740/R740xd servers can be equipped with up to three Nvidia Tesla GPUs each with 225W peak and 20W idle power [3, 18].

Voltage side channel to time thermal attacks. Due to time-varying loads, the attacker needs to find a good timing for successful attacks (especially for repeated attacks) when benign tenants' aggregate power load (or cooling load) is high. The attacker can utilize a side channel — voltage side channel in our study — to estimate benign tenants' power draw from the shared PDU. The voltage side channel is robust against changes in the environment and provides high accuracy due to its *wired* signal [68]. Utilizing the voltage side channel requires one analog-to-digital converter (ADC) that can fit on a server's power supply unit (as demonstrated in an orthogonal study for USB-powered IoT devices [86]). As



(a) (b)

Figure 3.5: (a) Server voltage carries the servers' load information. (b) Load estimation error of the voltage side channel.

shown in Fig. 3.4(a), the ADC taps into the server's input voltage to sample the PDU-level voltage.

For the readers' understanding, we show in Fig. 3.5(a) the fundamental principle behind the voltage side channel as recently proposed in [68]. The key idea is that because of the voltage drop along the shared power cable, the total load information (proportional to current) is contained in the voltage signal, e.g., V_1, entering any servers connected to the PDU. Meanwhile, all today's servers have power factor correction (PFC) circuits that generate high-frequency voltage ripples, whose amplitude is strongly correlated with the server load. Thus, the attacker can sense the incoming voltage signal, extract the voltage ripples, and estimate the total load at runtime.

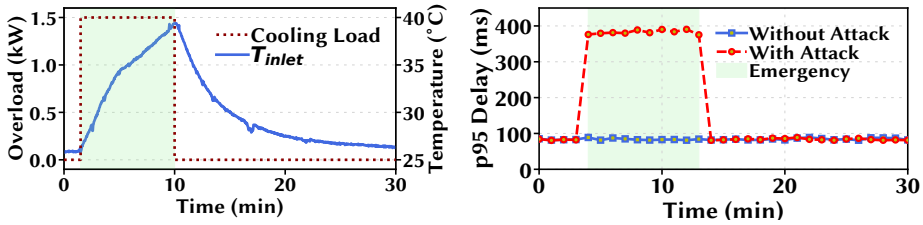
We run a 24-hour real-world workload trace in our prototype and collect the voltage signal using a NI digital data acquisition (DAQ) as an ADC proxy to extract the servers' total power load. We plot in Fig. 3.5(b) the probability distribution of load estimation errors, confirming that the voltage side channel can be leveraged for precisely timing thermal attacks.

Possibility of being detected. Detection of battery-assisted thermal attacks is not difficult, but contingent upon the edge colocation operator's practice of environment monitoring. Specifically, if the operator solely relies on power meters for monitoring tenants' loads and temperature sensors for conditioning the thermal environment, thermal attacks may possibly remain undetected until they cause damages. A service outage (due to one-shot attack) or more frequent thermal emergencies (due to repeated attacks) can trigger a thorough inspection, thus exposing the attacker. In order to proactively prevent such damages in advance, as discussed in Section 3.7, the operator can install additional monitoring apparatus such as server outlet air flow meters, which are not widely used in many data centers. Thus, although thermal attacks do not have a high degree of stealthiness, there is a need of attention to potential thermal attacks.

Relationship to power attacks. Power attacks exploit oversubscribed power capacity and can be launched *without* the need of battery [47, 68, 69, 71, 169]. On the other hand, our proposed thermal attacks are launched with the help of built-in battery for concealment of malicious cooling loads. Moreover, for repeated attacks, thermal attacks are *stateful* due to battery charging/discharging that results in temporal correlation of battery states, whereas power attacks are *stateless* and can be launched at any time without being constrained by the available battery energy. Thus, our thermal attacks are complementary to power attacks and present a potential threat by leveraging servers' built-in battery for a malicious purpose.

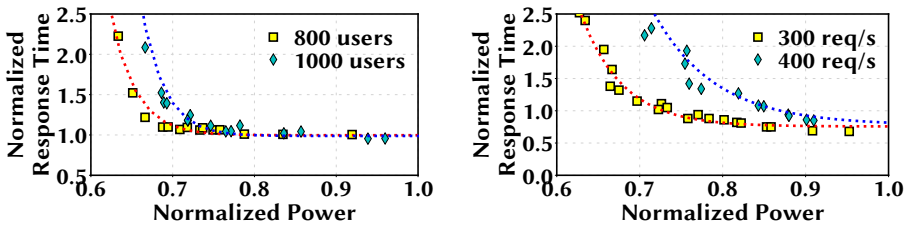
3.3.5 Prototype Demonstration of Thermal Attacks

To see the impact of thermal attacks, we run experiments on a rack of 14 Dell PowerEdge servers in a scaled environment with hot-cold aisles to mimic an edge colocation. The cooling system can support up to a cooling load of 3kW. We inject an additional 1.5kW load to overload the cooling system and measure the server inlet temperature. As shown in Fig. 3.6(a), the inlet temperature rises to nearly 40° C within minutes. Our experiment, albeit on a small scale, demonstrates the rapid increase of server inlet temperature due to a overloaded cooling system. This is also corroborated by other studies that demonstrate rapid temperature rises in data centers due to cooling malfunction [96]. We follow the ASHRAE safety limit and do not further overload our system [134].



(a) Temperature (b) Performance

Figure 3.6: Experiment in our server rack. (a) Server inlet temperature increases due to a cooling capacity overload by 1.5kW. (b) Latency performance is compromised due to server power capping for handling an emergency.



(a) Web Service (b) Web Search

Figure 3.7: Performance degradation due to power capping.

We implement the ClouSuite Web Service benchmark [2] in a set of 4 servers with a workload of 600 requests/s and show the impact of power capping on the 95-percentile response time, which is the key performance metric [57]. An x-percentile response time means that x% of the requests have a latency less than this response time. For illustration, we throttle the CPU speed to cap the total server power to 60% of the peak power. We see from Fig. 3.6(b) that during the emergency, the response time jumps nearly four times to 400ms. We also extend our experiments to Web Search implementation from CloudSuite [2]. We show the 95-th percentile response time normalized to the service level agreement (100ms) for two different numbers of users for Web Service in Fig. 3.7(a) and two different request rates for Web Search in Fig. 3.7(b), respectively. The server power consumption is normalized to the peak. We see that when the server power consumption decreases, the response times for both applications increase for any given workload level. This reveals the degree of performance degradation faced by tenants when they reduce their power consumption while the workload remains unchanged.

3.4 Learning an Attack Policy

An one-shot attack is a special case of repeated attacks if the attacker sets a sufficiently high threshold on benign tenant's load (above which an attack is launched) and greedily use up its large built-in battery energy. Thus, we now study a general repeated attack policy, Foresighted, by formulating it as a discrete-time Markov decision process (MDP) and using reinforcement learning. The repeated attack policy has a structural

property: *attack when both the benign tenants' server load and the battery energy level are sufficiently high.*

3.4.1 MDP formulation

We divide the entire time horizon into time slots (e.g., 1 minute each) indexed by $k = 0, 1, 2, \cdots, \infty$, and present our MDP formulation below.

- System state: $s = (b, u) \in \mathcal{S}$
- Action: $a(s) \in \mathcal{A}(s)$
- State transition probabilities: $P(s, a, s')$
- Reward function: $R(s, a, s')$
- Discount factor: $\gamma \in (0, 1)$

The tuple (s, a, s') means that, given an action a, the system state evolves from s to s'. In our problem, the system state includes two sub-states: battery state (the amount of remaining energy b in the batter units) and the attacker's estimated benign tenants' load state u (using a voltage side channel in Section 3.3.4 [68]). Note that we consider the estimated load as part of the system state, because the true value of servers' total load is not available to the attacker. We consider three actions: **(1)** charging the battery units; **(2)** launching a thermal attack by running the servers at peak power and discharging batteries; and **(3)** standby, i.e., running dummy workloads without charging or discharging batteries. The battery's charging rate is fixed at the vendor recommended value, while the effective discharging rate (i.e., power actually delivered to servers, excluding battery losses) is set to

p_b which, if combined with the attacker's subscribed capacity c_a, can support the attacker's total server power consumption p_a for thermal attacks. The state transitions are governed by benign tenants' load that is exogenous to the attacker and the battery energy evolution which is controlled by the attacker's charging/discharging decision.

We define the attacker's reward function as follows:

$$R(s, a, s') = w \cdot [T(s, a) - T_0]^+ - \beta(a), \tag{3.2}$$

where $T(s, a)$ is the resulting server inlet temperature, T_0 is the server inlet temperature conditioned by the operator without attacks, $\beta(a)$ is a cost term, and the operator $[\cdot]^+$ means $\max(\cdot, 0)$. Note that the attacker can easily sense the resulting inlet temperature $T(s, a)$, because today's servers have built-in temperature sensors to monitor the server inlet temperature for safety reasons (i.e., if the server inlet temperature is too high, the server may shut down by itself [30]). Clearly, after discharging batteries, the attacker needs to recharge them, which hence draws more energy from the operator's PDU than otherwise. To account for this, we add a normalized *cost* term: $\beta(a) = 1$ during an attack and $\beta(a) = 0$ otherwise. The cost is normalized to 1, because the attacker discharges a fixed amount of energy for each attack. The weight $w \geq 0$ governs the tradeoff between server inlet temperature increase and total battery usage (or attack time): the larger w, the more importance of server inlet temperature increase and hence more attacks.

In a standard MDP, the goal is to find an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ (i.e., deciding an optimal action given each system state) which maximizes the total discounted reward $\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k, s_{k+1})$. The discount factor $\gamma \in (0, 1)$ is imposed to ensure the convergence of summation and implies in practice that future rewards are relatively less important than

immediate rewards [145]. Nonetheless, the resulting server inlet temperature $T(s, a)$ is an involved function that also depends on external factors such as the edge colocation layout, and the dynamics of benign tenants' power usage is unknown to the attacker. Thus, we need an online *learning* approach to identify the optimal policy π^* on the fly.

3.4.2 Batch Q-learning

Reinforcement learning can effectively assist an agent with finding optimal actions in an unknown environment. The cooling load state is essentially uncontrollable and exogenous to the attacker. On the other hand, the battery state is fully controllable and, with simplification, can be approximated as $b_{k+1} = \min(b_k + e_k, \bar{B})$, where e_k is the charged energy during one time slot (a negative value means battery discharging for attacks) and $\bar{B}$ is the total battery capacity. Thus, we adopt batch Q-learning [168], by extending the widely-used standard Q-learning [145, 168]. Concretely, by introducing an intermediate state (also called *post state* $\tilde{s}_k$), we have two state transition processes: from s_k to $\tilde{s}_k$, we only update the battery state whose transition, according to the attacker's action, is fully determined; then, from $\tilde{s}_k$ to s_{k+1}, we will update the cooling demand state based on observations. More specifically, for each time slot k, our proposed batch Q-learning works as follows:

$$a_k \leftarrow \arg\max_{a \in \mathcal{A}(s_k)} [Q(s_k, a) + \theta V(\tilde{s}_k(s_k, a))] \tag{3.3}$$

$$\tilde{s}_k(s_k, a_k) \leftarrow f(s_k, a_k) \tag{3.4}$$

$$Q(s_k, a_k) \leftarrow (1 - \delta) Q(s_k, a_k) + \delta R(s_k, a_k, s_{k+1}) \tag{3.5}$$

$$C(s_k) = \max_a [Q(s_k, a) + \gamma V(\tilde{s}_k)] \tag{3.6}$$

$$V(\tilde{s}_k) = (1 - \delta) V(\tilde{s}_k) + \delta C(s_{k+1}) \tag{3.7}$$

where $\delta \in (0, 1)$ is the learning rate, and only the battery state is updated based on the attacker's charging/discharging action when setting the post state $\tilde{s}_k(s_k, a)$ in Eqn. 3.4.

Unlike standard Q-learning, three different *value matrixes* are used for batch learning: **state-action value** $Q(s_k, a_k)$, **post-state value** $V(\tilde{s}_k)$, and **normal state value** $C(S_k)$. First, after observing the system state s_k, the attacker makes an action a based on $Q(s_k, a)$ and post-state value $V(\tilde{s}_k(s_k, a))$ according to Eqn. 3.3. Then, post state s_k can be obtained based on attacker's action. Next, the reward R_k is obtained based on attacker's observed server inlet temperature and its reward function in Eqn. (3.2). Meanwhile, the next state s_{k+1} is obtained by estimating the cooling state through a voltage side channel as discussed in Section 3.3.4. Thus, the three value matrixes can be updated recursively according to Eqns. (3.5), (3.6) and (3.7), respectively, making the learning process converge more quickly.

3.5 Evaluation Methodology

In this section, we first present the default simulation settings and evaluation metrics, and then validate our simulation model.

3.5.1 Settings

It is practically challenging, if possible at all, to evaluate different thermal attack strategies over a timescale of years. Thus, we resort to a simulation-based approach based on the well-established computational fluid dynamics (CFD) analysis [51, 69, 109, 142] to simulate thermal dynamics. This is also the state-of-the-art methodology in data center-

Table 3.1: List of parameters with the default values.

Parameter	**Value**
Data Center Capacity	8 kW
Number of Tenants	4
Number of Servers	40
Number of Server Racks	2
Attacker's Capacity (c_a)	0.8 kW
Attacker's Total Battery Capacity ($\bar{B}$)	0.2 kWh
Attack Thermal Load from Battery	1 kW
Charging Rate of the Battery	0.2 kW
Temperature Threshold for Emergency (T_{th})	32°C
Q-learning Discount Factor (γ)	0.99
Q-learning Learning Rate ($\delta(t)$)	$1/t^{0.85}$

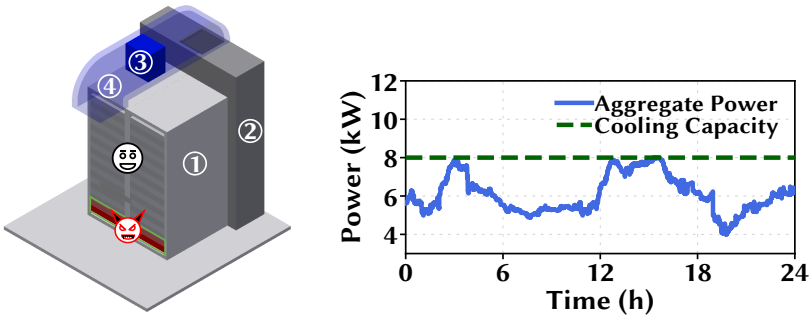
scale research [69, 142, 158, 165]. Prior to simulations, we will also validate our simulation model with real experiments on our scaled-down prototype of 14 servers. We list the default simulation parameters in Table 3.1.

Edge colocation infrastructure. We consider a containerized modular data center design, which is particularly suitable for edge colocations due to its self-contained design. We follow the specification of the Vertiv SmartMod container data center with two server racks, each holding 20 servers [151]. We consider there are four tenants (including the attacker) with a total subscribed power (i.e., the power capacity) of 8kW, where each server's maximum power consumption is 200W. The attacker has 4 servers with a total subscribed capacity of 0.8 kW while the other three benign tenants each subscribe to 2.4kW. The attacker's servers are shown in red shades in Fig. 3.8(a). Note that, while we place the attacker's servers at the bottom of the rack, their location within the rack does not play any significant role in the attack since the cooling load is determined by server power. The data center employs heat containment for the hot exhaust air returning to the AC. The AC

supplies cold air at 27°C, with a cooling capacity of 8kW. Fig. 3.8(a) shows the layout used in our experiment.

Thermal environment. The CFD analysis provides the most detailed thermal dynamics of a data center (e.g., even Google uses CFD analysis to predict thermal distributions [51]). However, it is computationally exhaustive to run transient CFD analysis for long experiments (e.g., a year) [142]. Therefore, following the literature [69, 142], we model the data center's heat flow using a heat distribution matrix, for which we only need to obtain the matrix parameters using shorter CFD analysis. Specifically, to extract the heat distribution matrix, we test the data center with a heat spike from each server and measure the resulting temperature impact for 10 minutes. We repeat the process for all servers to completely build the matrix. We use the 10-minute window to allow the heat convection through the air and capture the gradual temperature build-up from sustained server heat generation. We limit the CFD analysis of each heat spike to 10 minutes since we find no measurable impact beyond this time horizon. The accuracy of CFD analysis and the heat distribution model has been extensively verified with real systems [142, 159], and will also be validated against our prototype in Section 3.5.2. Because of the well-insulated environment, we do not incorporate the impact of outside temperature. Even with low outside temperature, if overloaded, data center's cooling system cannot remove all server heat.

Attacker. The attacker has built-in batteries integrated with the servers' power supply units. While the capacity of each server subscribed from the operator is 200W, each of the attacker's servers can run at a peak power of 450W by discharging built-in batteries



(a) (b)

Figure 3.8: (a) Data center layout. ① Server racks. ② Heat containment. ③ Air conditioner. ④ Supply air duct. (b) 24-hour snapshot of the power trace.

to supply the additional 250W. Thus, the attacker can inject up to 1kW cooling load for repeated attacks. When recharging, the built-in batteries have a total charging rate of 0.2 kW. We use battery specification of [4] with a suitable size for placing inside a server and set the attacker's default total battery capacity to 0.2kWh with 0.05kWh (i.e., 200W for 15 minutes) per server. If the attacker aims at an one-shot attack, each of its four servers has a peak power of 950W, resulting in a total attack load of 3KW. This can be achieved by using multiple power hungry GPUs (e.g., Nvidia RTX 3080, each with a full power of 320W [107]) in each server. The current battery energy density [4] is enough to support the additional load for an one-shot attack, as each attack only lasts a few minutes.

Thermal emergency and system outage. A thermal emergency is considered to arise when the server inlet temperature exceeds 32°C for at least 2 minutes. We consider 32°C as the threshold temperature, because it is the maximum allowed temperature based on the ASHRAE guideline for data centers with enterprise-grade servers and storage [134]. To handle a thermal emergency, each server (including attacker's servers) is required to cap its power below 120W (60% of capacity) to prevent more serious impacts. As a precaution, load

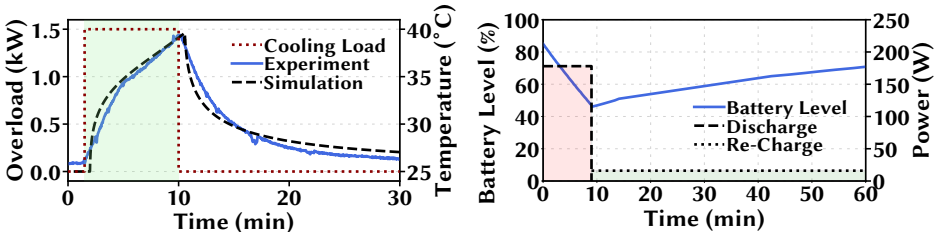
capping lasts for 5 minutes for each thermal emergency. If the inlet temperature continues rising to reach 45°C, automatic shutdown occurs (e.g., the shared PDU can power off), creating a system outage and service interruptions [134].

Power trace. For the three benign tenants, we use workload traces from Facebook and Baidu [158, 165], and generate a year-long synthetic power trace from request-level log using server power models validated in real systems [43, 44, 108]. The total power usage is scaled to have a 75% average utilization in our 8kW data center. We show a 24-hour snapshot of the power trace in Fig 3.8(b). To demonstrate its robustness across different load patterns, we also run an alternate power trace and show in Fig 3.15 in Section 3.6.6.

Application performance. For delay-sensitive workloads, high-percentile latency is the most critical metric [57]. Here, we consider 95-percentile response time as the performance metric and model the tenants' performance based on experiments on our small cluster (Fig 3.7 in Appendix 3.3.5).

Q-learning parameters. Following the literature [41], we set the default discount factor $\gamma = 0.99$ and a dynamic learning rate that is updated everyday using $\delta(t) = 1/t^{0.85}$, where t is the number of days elapsed. We use one minute as each time slot, and show the other parameters when presenting the results in Section 3.6. To initialize the table of Q values, we use random power traces offline based on an initial attack policy. Our results show that during the online learning stage, the action policy can converge quickly (often within 1-4 weeks).

Evaluation metrics. For the adverse thermal environment, we consider the average server inlet temperature increase, the probability distribution of the temperature,



(a) (b)

Figure 3.9: Experimental validation of our simulation model.

and the total emergency hours due to repeated thermal attacks. For benign tenants, we examine their performance degradation. We also study the average response time during the emergency periods normalized to that of without any emergencies.

3.5.2 Experimental Validation of Our Simulation Model

While simulation-based evaluation is widely used in data center research [48, 69, 157], we validate our simulation model using real experiments on our prototype consisting of 14 servers and a 600VA CyberPower UPS battery. We look into the two important aspects of our simulation model — thermal dynamics and battery charging/discharging model.

Temperature dynamics. We place our server rack in a sealed environment with a comparable dimension to an edge data center. The rack is cooled by the building's central cooling system and has air vents on the top. We create an additional 1.5kW thermal overload beyond the limit that can be handled by the top air vents. We obtain the heat distribution model based on CFD analysis. In Fig. 3.9(a), we show the monitored server inlet temperature change along with our temperature change simulated using our model. We see

that both the heat distribution model and temperature sensor readings exhibit very similar dynamics. This is expected since we adopt well-established CFD-based simulation [69,142].

Battery energy dynamics. In our Q-learning and the simulation, we need to validate that the linear battery model $b_{k+1} = \min(b_k + e_k, \bar{B})$, where b_k is the battery level at time k, is accurate to model the battery energy changes with respect to the charging/discharging decisions. For this, we connect two Dell desktops with a total load of ∼175W to our UPS battery. We connect a power meter between the UPS and the AC power outlet to measure the total power consumption of the battery and the desktops. We connect another power meter between the UPS and the desktops to record the total power of the two desktops. Subtracting the later from the former gives the total power consumption of the UPS. To demonstrate the battery dynamics, we first run the UPS on the battery discharging mode by unplugging it from the AC outlet. After 10 minutes, we reconnect the UPS to the AC outlet, which puts it in the battery charging mode. We show the battery energy levels in Fig. 3.9(b). In our experiment, the charging rate is lower than the discharging rate, because of the additional UPS loss to power the running desktops. This experiment conforms to our choice of a linear battery energy model. While even more complicated and detailed battery models (e.g., impact of ambient temperature) may be adopted [60], it does not offer much additional insight for our purpose and our observations still hold.

To sum up, our simulation methodology (i.e., using CFD-based analysis for modeling temperature dynamics and using a linear charging/discharing model for battery energy dynamics) matches well with the real-world observations and hence can be used to evaluate thermal attacks with a good confidence.

3.6 Evaluation Results

We first show an example of one-shot attack. Then, for repeated attacks, we compare Foresighted with another attack policy, Myopic, that launches thermal attacks in a greedy manner whenever there is enough energy in the battery and the benign tenants' aggregate load is sufficiently high. Besides Myopic and Foresighted, we also consider Random as a benchmark, where the attacker randomly launches thermal attacks whenever it has enough battery energy without considering benign tenants' power loads.

3.6.1 Thermal Attack Demonstration

One-shot attack

We consider a 30-minute snapshot and demonstrate an one-shot attack in Fig. 3.10 where the attacker injects 3kW of intense attack load at around the 18th minute, causing the server inlet temperature to rise quickly. At around the 21st minute, a thermal emergency is triggered and power capping is applied, limiting the total metered load below 5KW. Nonetheless, the attack load remains to keep the server inlet temperature high enough beyond the safety threshold of 45°C [134], successfully resulting in a system outage. This is also consistent with other orthogonal studies that demonstrate a very quick rise of inlet temperature in case of a cooling system malfunction [96]. If the one-shot attack is coordinated across multiple colocations, a service interruption may occur and create significant damages.

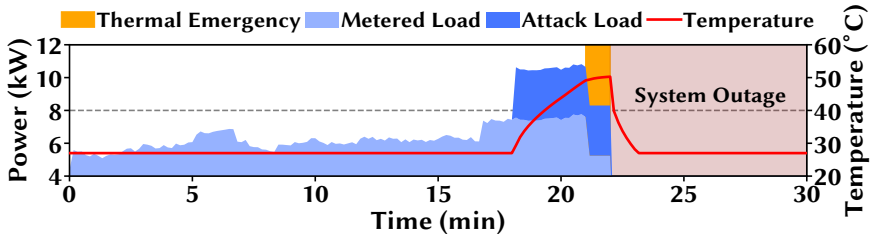


Figure 3.10: Demonstration of a one-shot attack.

Repeated attacks

We illustrate how repeated attacks create emergencies under different attack policies in Fig. 3.11 by considering a four-hour snapshot when the total power/cooling load is relatively higher. In our illustration, Random launches attacks for 8% of the times, Myopic sets the attack threshold at 7.4kW, while Foresighted uses a weight $w = 14$. These settings are chosen to yield similar attack times (i.e., 8% of the time) across different attack policies. The total power drawn from the operator's PDU is shown as "Metered Power", while the actual server power consumption also includes the contribution from the attacker's batteries ("Attack Load") and hence is larger than the metered power during the attacks. On the other hand, the actual server power is smaller than the metered power during battery charging. The discrepancy between the metered power and actual server power highlights the attacker's "behind-the-meter" cooling loads that are not monitored by the operator.

We see in Fig. 3.11 that thermal attacks using Random, which remains oblivious of the high cooling load, fail to create any thermal emergencies. Note that, Random's attacks look sparser in Fig. 3.11 since they are more spread over time while Myopic and Foresighted's attacks are concentrated in the high power/cooling load periods. Myopic exploits the voltage side channel [68] to detect benign tenants' high power loads and launches thermal attacks between hours 0 and 1. Since the power/cooling load remains at a high level, attacks con-

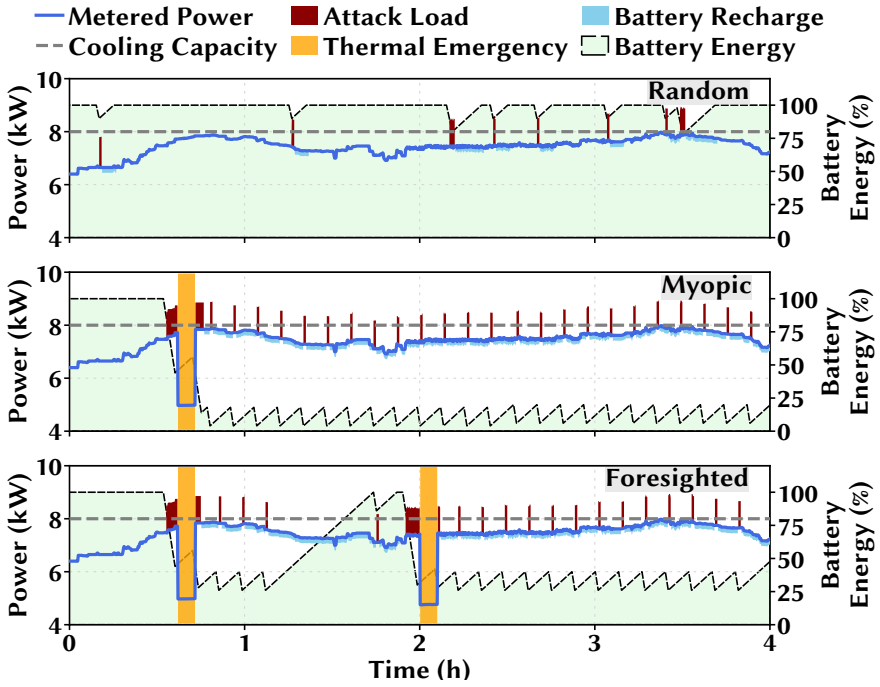
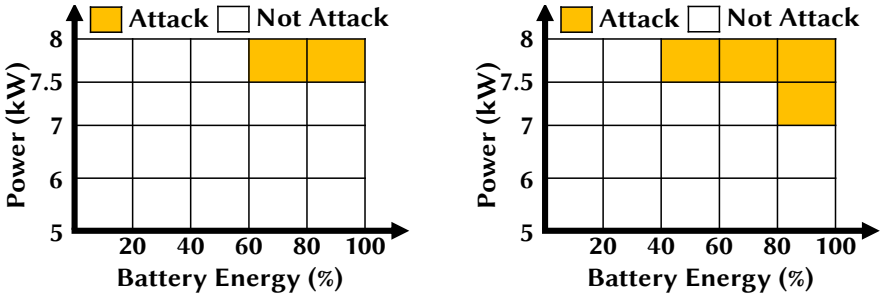


Figure 3.11: 4-hour snapshot of thermal attacks.

tinue until the operator announces a thermal emergency. At that point, attacks are stopped and the power consumption is capped to oblige to the operator's emergency handling protocol. The power returns to a normal level after being capped for 5 minutes to handle the thermal emergency.

While it also launches thermal attacks between hours 0 and 1, Foresighted does not launch a series of unsuccessful short-duration attacks like Myopic. Instead, it waits to regain the battery energy and launches a sustained thermal attack to trigger a second thermal emergency near hour 2. This shows the benefits of reinforcement learning which considers the impact of its actions on the future for maximizing the long-term benefits. Note that, even if Myopic only launches long-duration attacks with fully charged batteries,



(a) Weight $w = 9$ (b) Weight $w = 14$

Figure 3.12: Attack policy learnt by Foresighted.

unlike Foresighted, these attacks will more likely occur at the wrong times due to the lack of learning and accounting for battery level dynamics.

3.6.2 Attack Policy Learnt by Foresighted

We show in Fig. 3.12 the structural property of our repeated attack policy learnt by Foresighted: *attack when both the benign tenants' server load and the battery energy level are sufficiently high.* For illustration, we consider two different values of w (the larger w, the more weight on creating temperature increases and hence more attacks). For $w = 9$ in Fig. 3.12(a), attacks are launched only when the estimated power load (including the attacker's subscribed power capacity) is above 7.5kW and more than 60% of battery energy is left. For $w = 14$, we see that attacks are launched even for 40% remaining battery energy when the power is above 7.5kW. Meanwhile, Foresighted launches attacks at a lower power of 7kW when it has more than 80% battery energy.

3.6.3 Cost Estimate

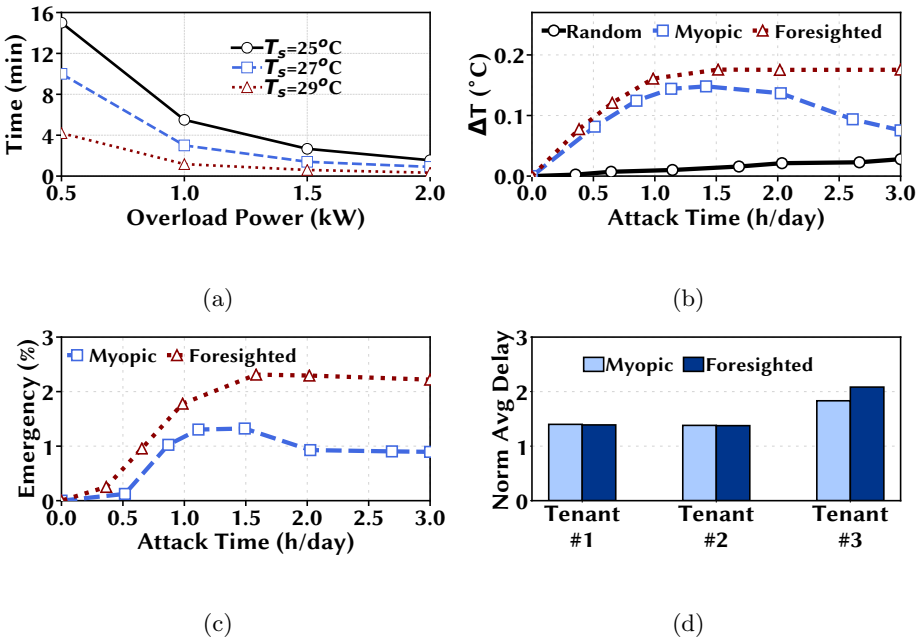


Figure 3.13: (a) Overload time required to exceed the temperature limit of 32°C. (b) Average temperature increase vs. attack time. (c) Total attack-induced emergency vs. attack time. (d) Tenants' performance during emergencies.

Benign tenants' cost. With an one-shot attack, benign tenants can suffer from service outages, which may be costly or even indirectly cause fatal damages (e.g., decreased safety for assisted driving [10]); with repeated attacks, tenants can potentially experience more frequent performance degradation. The monetary impact of thermal attacks is generally difficult to estimate. To offer an approximate point of reference, we provide a *ballpark* estimate for repeated attacks following prior studies [46, 67, 111] that calculate the cost impact resulting from the increased 95-percentile latency. Under our setting, Foresighted causes a total performance cost of roughly \$60+K/year to benign tenants in our 8kW edge colocation (roughly 80% of benign tenants' total rental costs plus amortized server costs), noting that the actual cost highly depends on the affected tenants' applications and can include additional indirect cost such as business reputation.

Attacker's cost. The attacker's cost involves the power capacity subscription cost, electricity cost, and server purchase cost: 150$/kW/month power subscription cost, 0.1$/kWh energy cost, and $4500 for each server [67]. It is on a par with the cost for other related attacks [48,68,69,71,169], and can be affordable for institutional or state-sponsored attackers.

3.6.4 Impact of Thermal Attacks

For repeated attacks, we first show in Fig. 3.13(a) how long it takes for the server inlet temperature to exceed the 32°C threshold. Naturally, the temperature exceeds the threshold sooner with increased cooling overload. Similarly, when the data center is already running hotter (i.e., higher supply temperature T_s), its temperature reaches the limit faster. We see that it takes less than four minutes to increase the data center temperature from 27°C to 32°C with one kW of additional cooling load, demonstrating the potential danger of thermal attacks.

We then vary the total attack energy injected into the edge colocation (i.e., total attack time), while keeping the attack load from the battery fixed at 1kW. We vary the attack probability for Random from 0% to 15%, the load threshold (including the attacker's own power subscription) for launching an attack under Myopic from 6.5kW to 8.0kW, and the weight parameter for Foresighted from $w = 0$ to $w = 30$. Figs. 3.13(b) and 3.13(c) show the average server inlet temperature increase (ΔT) beyond 27°C and the amount of attack-induced emergencies (measured in % of the total time) given different average daily

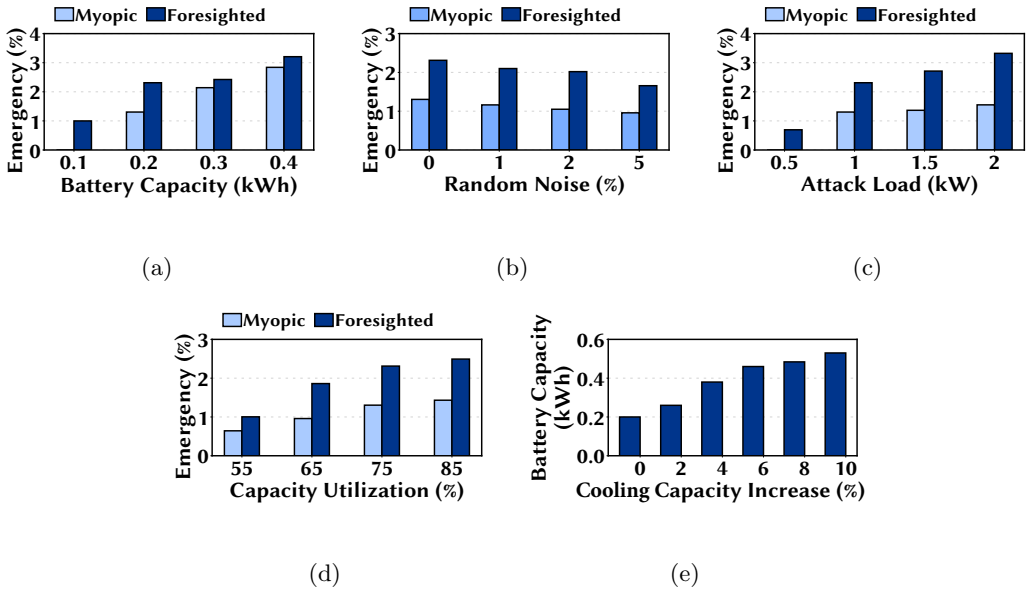


Figure 3.14: Sensitivity of Foresighted. (a) Battery capacity. (b) Load estimation due to random noise in side channel. (c) Attack load. (d) Average utilization of data center capacity. (e) Required battery capacity for extra cooling capacity.

attack times, respectively. In Fig. 3.13(c), we exclude Random because it fails to create any thermal emergency.

Temperature increase. We see in Fig. 3.13(b) that with more attacks, the temperature increase caused by Random also rises. For Myopic and Foresighted, the temperature increase rises very fast initially, when attacks are conservatively launched. However, as more attacks are launched, the temperature increase for Myopic peaks at around attack time of 1.1 hours per day and then starts to decrease. This is because Myopic launches premature attacks which deplete the battery energy and hence miss future attack opportunities. We see a similar impact on the annual thermal emergency time in Fig. 3.13(c) where Myopic's performance starts to deteriorate around attack time of 1.5 hours per day.

Foresighted takes the future into account and hence retains both the average temperature increase and annual emergency time increase with more thermal attacks. However, beyond an attack time of 1.5 hours per day, Foresighted cannot create further higher temperature increases nor more thermal emergencies. This is mainly because the total available attack opportunities are limited (i.e., benign tenants do not always have high power loads) and recharging batteries takes time. Nonetheless, given any amount of thermal attacks, Foresighted can create higher server inlet temperature increases and more thermal emergencies than Myopic.

Attack-induced thermal emergencies. In Fig. 3.13(c), we see that the attack-induced thermal emergencies for both Myopic and Foresighted are close to zero at low attack time. This is because the operator declares a thermal emergency when the data center temperature exceeds 32°C and stays there for at least two minutes. Hence, at low attack time which also corresponds to low average temperature increases in Fig. 3.13(b), there are almost no thermal emergencies due to attacks.

Performance impacts. We normalize the tenants' 95-percentile response time to that of without any emergencies. We take the average of the normalized response time during the emergency periods and show the result in Fig. 3.13(d). We see that Myopic has a slightly higher average performance impact than Foresighted. This is because Myopic mainly captures the most prominent attack opportunities while Foresighted intelligently picks up even the subtle opportunities with relatively lower impact, resulting in a lower *average* performance impact. Nonetheless, since Foresighted seizes both the prominent and subtle

attack opportunities, it results in more frequent thermal emergencies, thus resulting in a greater cost impact.

3.6.5 Sensitivity Study

We now study how the battery capacity, side channel accuracy, attack load, and data center average utilization affect the resulting thermal attacks. We also study the impact of additional cooling capacity on attacker's battery capacity requirement. We exclude Random from our study here since it fails to create any thermal emergency.

Battery capacity. Considering repeated attacks, we vary the battery capacity from 0.1 kWh to 0.4 kWh, and show the annual duration of thermal emergencies due to the attacks in Fig. 3.14(a). Naturally, a larger battery provides greater flexibility in launching thermal attacks. Hence, we see the annual thermal emergency time increases with battery capacity. We also see the difference between Myopic and Foresighted decreases with a larger battery as the battery is more likely to be available whenever Myopic needs it, like in Foresighted.

Load estimation accuracy. To test robustness against voltage side channel errors, we add varying degrees of random errors to the estimated loads of benign tenants and show our results in Fig. 3.14(b). As expected, the thermal emergency time decreases for both Myopic and Foresighted when there is more noise in the side channel. Nonetheless, Foresighted can still create a significant amount of thermal emergency, even using a noisy voltage side channel.

Attack load. The attack load determines how much additional cooling load is injected during each attack. We show the results in Fig. 3.14(c) where we keep the attacker's

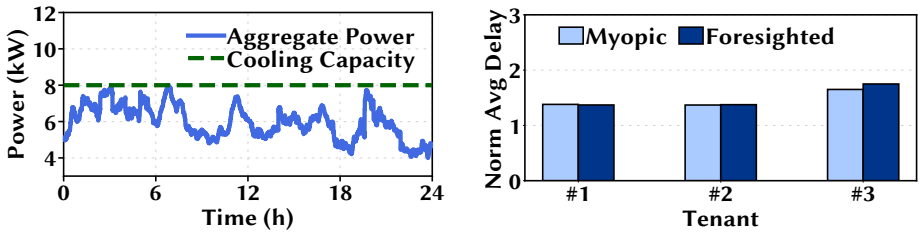
subscribed capacity at 0.8kW and scale the thermal attack load from 0.5kW to 2kW. We see that the annual emergency time greatly increase with a higher attack load and that Foresighted consistently outperforms Myopic by a great margin.

Capacity utilization. We study the impact of average data center utilization on the thermal attack by scaling the power trace of all the servers while maintaining the peak power at 8kW. Fig. 3.14(d) shows that the total thermal emergency time increases with increased capacity utilization. This is intuitive since an increased utilization means the data center more frequently operates close to its capacity, thus leading to more thermal attack opportunities.

Extra cooling capacity. We study the impact of the operator's extra cooling capacity on Foresighted's battery requirement to maintain similar impact (i.e., 2.3% emergency). In Fig. 3.14(e), we see that the extra cooling capacity mandates higher battery capacity. Specifically, the increase in battery capacity for 10% extra cooling capacity is about $\sim$0.3kWh, which can still be feasible given today's battery energy density. Note, however, that upgrading an existing data center cooling system to add extra cooling capacity is non-trivial due to constraints such as space limitation, data center uptime, etc. Thus, as discussed in Section 3.7, other defenses are more effective and cost-efficient, especially for an existing data center that has limited cooling capacity.

3.6.6 Results with an Alternate Power Trace

We conduct our year long evaluation with an alternate power trace to demonstrate that Foresighted is effective regardless of the benign tenants' load patterns. We use the Google cluster trace from [97] as the *alternate* total power trace. We show a 24-hour



(a) (b)

Figure 3.15: Results with an alternate power trace. (a) A 24-hour snapshot of the alternate power trace. (b) Tenants' performance during emergencies.

snapshot of the alternate power trace in Fig. 3.15(a). Like in the default setting, we scale the power trace to have a 75% average utilization in our 8kW edge colocation. We keep the same default settings as in Section 3.5 for Myopic and Foresighted. Fig. 3.15(b) shows that, with the alternate power trace, benign tenants suffer from similar performance degradation as in our earlier results. While we omit detailed discussion for space limitation, these findings are consistent with our earlier results.

3.7 Defense Mechanism

Tenants generally expect reliable power and cooling supplies (subject to contractual terms) from the colocation operator which manages non-IT systems. Thus, we offer possible defenses from the operator's perspective. We first discuss defenses that aim at preventing potential thermal attacks, followed by defenses that detect thermal attacks.

3.7.1 Prevention

The following defense strategies are proactive measures to inhibit potential thermal attacks.

Infrastructure resilience. A straightforward defense against thermal attacks is to reinforce an edge colocation's physical infrastructure for handling thermal overloads. For this, the operator can deploy a cooling system with additional redundancies. This approach, however, can increase the capital cost [49, 131] and be particularly challenging for existing systems. Alternatively, the operator can lower its server inlet temperature set point (to 20°C instead of the recommended 27°C) to have more margins for triggering thermal emergencies. The drawback is the increased cooling energy cost [130, 142]. Thus, while oversubscribing data center cooling capacity [100, 130, 131] and increasing temperature set point [142] have been suggested for cost efficiency, they should be carefully exercised, balancing the benefit versus risk to potential thermal attacks.

Rigorous move-in inspection. The colocation operator can employ a more rigorous background check and move-in inspection process for all tenants' servers to detect and remove integrated batteries. Note that without built-in batteries, the attacker cannot have additional power sources to support thermal attacks behind the meter or overload the shared cooling capacity, unless the data center cooling capacity is oversubscribed as suggested by recent studies [100, 130, 131]. Besides, the operator can also enforce on-site power load tests to ensure that the server power is consistent with the tenant's data center capacity subscription. The operator should be particularly careful about the servers' peak power.

Degrading physical side channels. The colocation operator may increase the attacker's uncertainties about timing attacks by degrading/eliminating the physical side channel. For example, it can add jamming noise signals into the colocation power networks

and/or use power line noise filters. Additionally, the operator may also prohibit unusual sensors (e.g., microphones) on tenants' servers in order to prevent an attacker from exploiting other possible but unknown side channels.

3.7.2 Detection

Detection strategies can be implemented to catch an attacker that may circumvent prevention approaches.

Detecting behind-the-meter cooling loads. The same power reading can result in different cooling loads and server inlet/outlet temperature, depending on whether malicious thermal attacks are launched or not. Thus, by using anomaly detection algorithms (e.g., cross-checking readings by temperature sensors and power meters), the operator can detect an irregular thermal environment possibly due to thermal attacks.

Identifying attacks from impacts. One-shot attacks can be easily identified through a thorough inspection if a system outage occurs. By contrast, repeated-attacks that inject milder loads to trigger more frequent thermal emergencies can require more efforts. Since precise temperature management is difficult with open airflow cooling, there can be occasional thermal emergencies in colocations even without thermal attacks; colocation operators often offer a long-term temperature SLA (e.g., the inlet temperature is conditioned below 27°C for 99% or more of the time) [38,65]. This may potentially allow an attacker to hide behind the statistics for a longer time. Thus, advanced algorithms can be implemented to monitor SLA metrics to early detect the presence of thermal attacks.

Improved data center monitoring. While the aforementioned approaches can detect thermal attacks, pin-pointing the attacker's servers — the source of the injected

cooling load — is still needed to hold the attacker accountable. Thus, to monitor the servers' actual cooling loads, the operator can measure each server's outlet temperature as well as the hot air flows. Alternatively, thermal cameras may be employed to identify the servers that are running extra hot. Likewise, microphone arrays can be used along with the thermal camera to pinpoint servers with fans spinning at a high speed (needed by servers that have higher cooling loads) [71]. While these monitoring apparatuses are not used in all data centers, they are readily available and can be easily installed by data centers to identify malicious cooling loads.

To sum up, there exist readily-available defenses, such as move-in inspection to disallow built-in batteries, advanced anomaly detection, and installation of monitoring apparatuses to locate the attacker. Given the potential threat of thermal attacks that are currently neglected, the edge colocation operator can implement one or more of the suggested defenses to safeguard its thermal environment for tenants.

3.8 Related Works

Power and thermal management. The common practice of aggressive capacity oversubscription can create occasional capacity overloads when the demand peaks [6, 67, 81, 157, 158, 165]. To safely ride through power emergencies, numerous graceful power capping techniques have been proposed, such as throttling CPU frequencies [165], migrating/deferring workloads [155, 158], and discharging batteries to boost power supply [6, 81, 157]. Likewise, managing server loads to handle thermal emergencies are equally crucial [77, 100, 114]. These studies, however, are not applicable for colocations whose oper-

ators have no control over tenants' servers. Moreover, they do not consider an adversarial setting. More recent works [66,67] propose market approaches to coordinate tenants' power demand in colocations, but they assume that tenants are all benign without any malicious intentions.

Data center security and thermal fault attacks. Securing data centers against cyber attacks, such as network DDoS [173] and data/privacy breach [175], has been extensively investigated. Prior studies have also considered malicious thermal load attacks on a single device [132]. More recently, data center power and cooling system security has been emerging as a crucial concern [47,48,68,69,71,87,169]. However, these works focus on overloading the power infrastructure (i.e., power attack) of large data centers with multi-level redundancy or creating hot-spots (i.e., thermal attack) in Amazon-type cloud with frequent VM shuffling. In contrast, we focus on novel battery-assisted thermal attacks in a shared edge colocation. Moreover, our repeated battery-assisted thermal attacks are *stateful* whereas prior attacks are stateless as the current attack does not depend on any past/future attacks.

Battery management and others. The prior studies have exploited batteries for various purposes, such as better energy capacity [60], concealing a household's electricity usage information from the utility for better privacy [171], smoothing data center power demand [6,81,157], among many others. To our knowledge, however, our study is the first to leverage batteries for a malicious purpose — one-shot or repeated thermal attacks in edge colocations — which highlights the need of attention to the potential threat.

3.9 Conclusion

In this chapter, we discovered that the sharing of cooling systems may expose edge colocations' potential vulnerabilities to both one-shot and repeated thermal attacks assisted with built-in batteries. For repeated attacks, we presented a foresighted attack policy which, using reinforcement learning, learns on the fly a good timing for thermal attacks. We also ran simulations to validate our attacks and showed that, for an 8kW edge colocation, an attacker can cause performance degradation for affected tenants. Finally, we suggested effective countermeasures against potential thermal attacks that are currently neglected in many data centers.

Chapter 4

Learning for Robust Combinatorial Problems: Algorithms and Applications in Computing Systems

In this chapter, we explore the robust solutions for combinatorial optimization problem leveraging machine learning-assisted methods. Specifically, we are focusing on the dynamical resource allocation problem in edge computing. A novel machine learning-assistant method —RobustPN, is proposed for the general robust optimization problem. Then, simulations are performed on a toy example and the workload assignment problems in vehicular edge computing, verifying the effectiveness and robustness of proposed method.

4.1 Introduction

With the development of the internet of things (IoT), the last decades have witnessed a huge growth of computations in edge users, such as mobile or vehicular applications [128]. To improve the service quality and user experience, the computation is gradually moving toward computing edge — edge computing. Meanwhile, most modern devices are installed with powerful local computing resources. Then, the surplus computing resources on each edge device can be considered as "data center serves". As a complementary to traditional edge computing, some researchers contribute their efforts to *device-as-a-resource* [36, 170]. For instance, vehicles with extra computation resources can provide distributed computing to other edge devices — called vehicular edge computing (VEC) system.

Due to the continuous-changing environment (e.g, device's locations or resource status), the computing resources and respective service quality typically exhibit significant fluctuations, especially for VEC system [20]. Hence, an operator needs to dynamically and efficiently allocate the computing resources and assign workloads to respect servers. Normally, the resource allocation problems can be formulated as combinatorial optimization, which is very challenging to solve due to its **NP-hard nature**.

Without any doubt, an optimal solution can be achieved with the 'brute-force' method. But it often required factorial time, which is intolerable for the instant resource allocation settings. For faster solutions, most previous researches are focusing on greedy algorithms, which can handle the combinatorial problem in polynomial time but with sub-optimal solutions [59, 161]. Recently, machine learning-assistant approaches are proposed

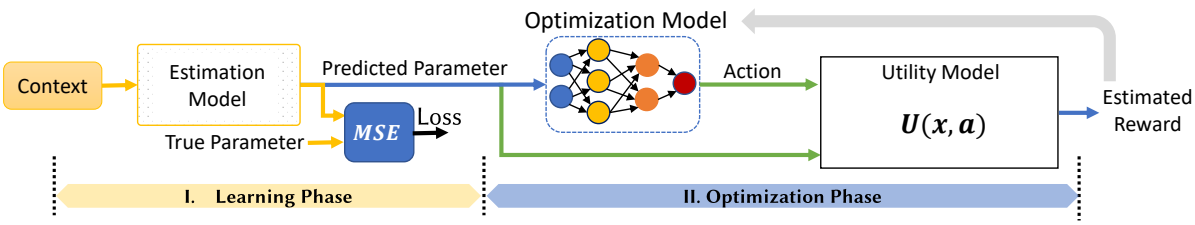


Figure 4.1: General architecture for "*Predict, then optimize*".

to directly learn the optimization solution with data or prior knowledge, such as 'neural reinforcement learning' for TSP problem or 'learning to optimize' for wireless channel assignment [14, 88].

However, the **optimization parameters** of combinatorial problems are usually estimated by some upstream models especially for the real-world system, such as edge computing. For instance, the job success probability, in edge computing, is estimated from the context information, including wireless network and computation conditions. The entire system can be treated as a "*Predict, then optimize*" architecture [35], as shown in Figure 4.1. Here, the estimation model can be either an empirical model or a DNN model pre-trained using supervised learning. As for the *learning to optimize* methods we are focusing on, the optimization model is also implemented with neural networks but trained with actor-critic approaches. Since the prediction error of optimization parameters may have strong impacts on respective actions and result in poor performance, we should consider a robust solution that provides solutions with acceptable performance even under the worst-case estimation results. In summary, the performance and its robustness should be considered when solving the respect optimization problems.

Our contributions. In this chapter, we propose a novel machine-learning assistant approach —RobustPN, to solve the robust combinatorial optimization problem. Leveraging the actor-critic platform and *learning to optimize* technique, two sets of DNN models are trained, which together can provide fast and robust solutions to CO problems. The key novelty of RobustPN is the introduction of machine learning for robust CO problems, while all previous works focus on CO problems without considering the robustness.

However, it is challenging to solve the robust CO problem directly using conventional machine learning(ML) methods, such as supervised learning. Firstly, ML algorithms usually require supervision labels, which are time-expensive to collect (calculate) for CO problems and even harder for robust CO problems. For instance, exponential computation time $O(a^N)$ is required to solve a single CO problem with N-dimensional actions. Besides, the ML algorithms usually require at least $>$ 10k labeled samples for training. Secondly, the robust CO problem can be formulated as a minimax problem (Eqn 4.2), which including two optimization processes. The Conventional 'learning to optimize'(L2O) method is developed for a single optimization process (e.g, minimization or maximization). Hence, L2O can not directly be applied to robust CO problems. Finally, the robust CO problem is formulated with two optimization variables: action a and robust error δ. While error δ is a continuous variable, a locates in a discrete space, which is another challenge when developing our machine learning-assistant algorithms.

To solve the aforementioned challenges, we propose a machine learning framework utilizing the policy gradient and L2O. This framework is motivated by the actor-critic architecture in reinforcement learning (RL), where the actor provides actions to the critic

and then gets respect feedback. Similarly, the RobustPN algorithm is comprised of two blocks: a policy model as the actor and a worst-case block as the critic. The policy model provides action probabilities given the respect problem parameters, while the robust relevant feedback (e.g, utility) is retrieved from the critic block. To solve the discrete action space issues, the reinforced policy gradient is utilized for policy model training, which doesn't require any labels. Note that there are some key differences between the proposed RobustPN and actor-critic of RL. 1). the actor and critic models are trained alternatively in RL due to the state correlation. However, the critic models can be pre-trained in RobustPN, because the worst-case error only is independent of the policy model. 2). our critic model is trained with L2O method which also doesn't require any labels, further reducing the labeling efforts. Hence, the proposed RobustPN learning framework is suitable to solve robust CO problems.

To verify the performance of RobustPN, we perform experiments on two CO problems: a synthesis toy problem with a small action space and a task offloading problem in vehicular edge computing systems. We evaluate the respect robust performances by comparing with other baseline algorithms and oracle solutions. Firstly, we validate that a robust solution is required for CO problems. Then, our results show that a robust policy can be obtained from well-trained DNN models (including policy and worst-case models), verifying the effectiveness of the proposed RobustPN. Additionally, we also investigate the impacts of the error budget on performance. The results demonstrate that the proposed RobustPN sacrifices true performance for robustness. With a smaller error budget, RobustPN can provide better performance on both true object values and robust values.

The rest of this chapter is organized as follows. Firstly, Section 4.2 introduces the general CO and Robust-CO problems. Then, the proposed RobustPN is described in section 4.3. To validate its effectivess, we run experiments on two applications in section 4.4 and section 4.5. The related works are presented in section 4.6. Finally, we make a conclusion in section 4.7.

4.2 Problem Formulation

We first show the formulation for general combinatorial optimization(CO), and then present the robust CO problem that we are focusing on.

4.2.1 Combinatorial Optimization

Combinatorial optimization (CO) problems widely exist in scientific and engineering applications, such as network resource allocation, scheduling, and planning. Typically, the goal of CO is to find an optimal selection out of a finite set. Many integer programming problems can also be viewed as CO. In general, a CO problem can be formulated as follows:

$$\min_{a \in \mathcal{A}} f(x.a) \tag{4.1}$$

where a is the decision variable, $\mathcal{A}$ is the eligible action space defined by the other constraints, $f(\cdot, \cdot)$ indicates the optimization object function with parameter x. Specifically, the optimization target is to find actions that minimizing the object f under a parameter x. Note that both x and a are presented as high-dimensional vectors in an CO problem.

Usually, problem (4.1) can be directly optimized by classic convex optimization methods when the object f is continuous and differentiable on a. But, the CO problem is

hard to solve optimally in polynomial time because of discrete action space and the non-convex object function. Therefore, most combinatorial problems (e.g, TSP) are solved by greedy algorithms as an approximation, which can provide greedy solutions within polynomial time [59, 161]. With the development of deep learning techniques, some machine learning-assisted methods are proposed recently, which can provide a better solution than heuristic methods with fast speed. A survey is provided in Section 4.6.

4.2.2 Robust Combinatorial Problem

For real-world applications, the optimization parameter x may not always be accurate, resulting in performance degradation. Naturally, a robust policy is desired for the combinatorial problem (4.1). For instance, a workload assignment policy is desired for lower latency and better user experience in edge computing systems. A real-time workload demand is required for the assignment policy, serving as the policy parameters. Since the workload demand is usually estimated from historical user demand, the parameter uncertainty is inevitable, resulting in the robustness concerns.

In this work, we consider a robust combinatorial problem that can be viewed as an extension of problem (4.1). Specifically, we targeting at a robust policy that achieves the best performance under the worst-case parameter x', considering a fixed error budget Δ of the true parameter x. Then, the robust combinatorial problem can be formulated as a minimax problem:

$$\min_{a \in \mathcal{A}} \max_{\delta \in \Delta} f(x + \delta, a) \tag{4.2}$$

where x is the uncertain parameter, δ denotes the parameter error, and Δ is respective error space defined with error budget. For instance, an L_2-norm error space with ϵ-bounded can be represented as $\Delta = \{\delta,\ |\delta|_2 \leq \epsilon\}$.

To achieve a robust policy, we need to solve the minimax problem (4.2), which is much harder than the general combinatorial problem due to the combination of minimax and discrete actions space. In the next section, we propose a machine learning-assisted algorithm — Robust Policy Network (RobustPN) — to approximate the optimal robust solution of Eqn (4.2) within polynomial time. Specifically, a set of deep neural networks $\{\mu_i\}$ are trained with policy gradient, which accepts an inaccurate/estimated parameter x and provides a probability distribution $p_i(a)$ for the respective action variable a. More detailed algorithm designs are presented in section 4.3.

4.3 Methodologies

In this section, we present the machine learning-assisted method for solving the CO and robust CO problem, including the architecture design and the respective learning algorithms. First, the policy gradient method is introduced to solve the general CO problem. Then, a robust learning framework is presented on top of the policy gradient and robust critic modules. In addition, some advanced techniques are utilized, including model ensemble and learning to optimize.

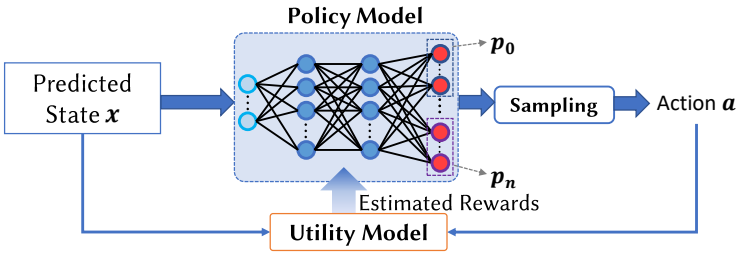


Figure 4.2: Policy gradient network architecture for combinatorial optimization problems.

4.3.1 Policy Gradient

The idea of policy gradient originates from reinforcement learning, targeting at modeling and optimizing the policy directly [110]. As a simplified version of reinforcement learning, policy gradient methods are also applicable for combinatorial problems when considering the objective function as the critic model. Here, we intend to model the policy with a parameterized neural network $\pi_\theta(a|x)$, where θ represents the model weight, which can be learned using machine learning methods. The respect learning framework is illustrated in Figure 4.2, including a policy model (implemented with neuron network), a sampling module, and a utility model as the critic.

As shown in Figure 4.2, the policy model accepts the optimization parameters x and directly provides a probability distribution $p(a)$ over the action space $\mathcal{A}$. A well-trained policy model should assign high probabilities to actions that minimizing the utility. For CO problems, the action space is significantly large (e.g, factorial $O(m^n)$). For instance, 10-dimensional x with 5 possible integer values will result in $5^{10} \simeq 9.5e^{13}$ actions. Hence, it is hard to build a traditional policy network, where each node in the output layer represents a single action. Instead, we design the policy network with n output groups and each group includes m nodes, as shown in Fig. 4.2. Here, we assume the combined action as

$\mathbf{a} = [a_1, a_2, ...a_n]^T$, where $a_i \in \{1, 2...m\}$ indicates the values for ith action component. For neuron network design, the softmax activation function is applied within each group of output nodes, producing a set of probabilities p_i. Using the chain rule, we can factorize the final action probability as Eqn (4.3).

$$P_\theta(\mathbf{a}|x) = \prod_{i=1}^{n} p_\theta(a_i|x) \tag{4.3}$$

Conventionally, the neural networks are trained by supervised learning methods with well-labeled training datasets. However, it is hard to obtain the solutions (true labels) for (robust) CO problems. Hence, we turn to learn weight θ with the policy gradient method, which doesn't require any labels. Here, we use one popular method — Monte-Carlo reinforced policy gradient — to update the policy parameter θ . The learning gradient can be summarized as Eqn (4.4), where $J(\theta)$ is the loss function, $V(x)$ represents the average value for parameter x, and P_θ indicates the action probability. Specially, we compute the probability distribution $P_\theta(a|x)$ through forward pass. Then, an action a_s is sampled and fed into the critical module. With the returned value for a_s, the gradient information can be calculated from back-propagation, using for model training.

$$\nabla_\theta J(\theta) = \mathbb{E}_x \left[-(f(x, a) - V(x))\nabla_\theta P_\theta(a|x)\right] \tag{4.4}$$

In Monte-Carlo reinforced policy gradient, action net benefits $f(x, a)) - V(x)$ is used for gradient calculation, instead of the absolute object $f(x, a)$. This can reduce the variance of gradient estimation and accelerate the training process [110]. Here, we calculate $V(x)$ as the average object of randomly sampled actions $V(x) = 1/|S| \sum_{a_s \in S}[f(x, a_s)]$,

Algorithm 2 Policy Gradient Baseline Algorithm.

Training dataset $\{x_i \in \mathcal{D}^T\}$**, training epochs** T**, batch size** B**, sampling size** $|S|$**.**

Initialize: policy network parameters θ

For $t = 1, 2...T$

  For $i \in 1, 2, ...B$

    Sample the input parameter x_i from $\mathcal{D}^T$.

    Predict probability distribution $P_\theta(a|x_i)$ according to Eqn. (4.3).

    Randomly sample a prediction action a based on the probability $P_\theta(a|x_i)$.

    Calculate the action's objective $f(x, a)$.

    Uniformly sampling $|S|$ actions from $\mathcal{A}$.

    Compute the baseline value as $V(x) = 1/|S| \sum_{a_s \in S}[f(x, a_s)]$.

    Compute gradient $g_i = \nabla_\theta J(\theta)$ in Eqn (4.4).

  Compute the batch gradient $g_\theta = 1/B \sum_{i=1}^{B} g_i$.

  Update parameters $\theta \leftarrow ADAM(\theta, g_\theta)$.

Return θ.

where a_s is random action and $|S|$ indicates the sampling size. The detailed learning process for reinforced policy gradient is summarized in Algorithm 2. Note that, Algorithm 2 will be considered as the baseline method(PN) in following parts.

4.3.2 Robust Policy Gradient

To solve the robust CO problem in Eqn 4.2, we proposed a Robust Policy Network (RobustPN) algorithm, based on conventional policy gradient method and actor-critic

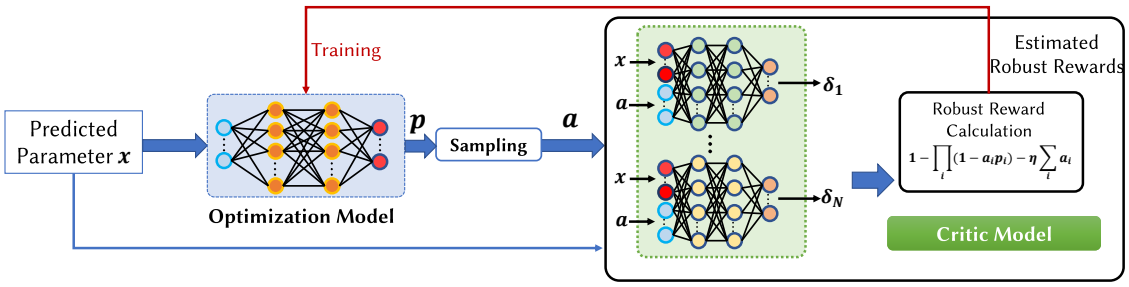


Figure 4.3: RobustPN architecture to solve robust combinatorial optimization problems.

platform. The RobustPN is comprised of two parts: a policy model and a worst-case critic module. The overall architecture for RobustPN is illustrated in Fig. 4.3. Here, the policy model can be viewed as actors, while the worst-case module can be treated as the critic. The detailed functionality for each module is presented as follows.

Policy Model. The policy module accepts the potentially erroneous optimization parameter x and provides the respective probability p for action candidates. Additionally, sampling modules are appended to the policy model, mapping the output probability p to action a. Then, sampled action can be further feed into critic modular for utility evaluation. To improve the model performance, a set of policy models can be trained independently and simultaneously without any modification for the current platform. The training algorithm for each policy model is summarized as Algorithm 3.

Worst-case Critic Module. The worst-case critic module targets generating the worst-case rewards for the actor (policy model). It comprises two parts: a set of worst-case deep neural networks (DNN) and a robust reward calculation module. The worst-case DNN models provides worst-case error δ, given the the input parameter x and action a. Mathematically, the worse-case DNN focuses on the inner constrained optimization of the

Algorithm 3 Robust Policy Algorithm

Input: Training dataset $(x_i) \in \mathcal{D}^T$, number of training epochs T, batch size B, sampling size S, number of policy networks $M = 1$, number of worst-case networks N, input parameter uncertainty bound ϵ, and object function $f(x, a)$.

Output: Policy network parameters θ_i^p and worst-case network parameters θ_j^w.

Initialize: policy network parameters $\theta_i^p, \forall i = 1, 2...M$

Generate greedy action $a_g = P_g(x)$ on training dataset $\mathcal{D}^T$.

For $i = 1, 2...N$

Initialize worst-case network parameters $\theta_j^w, \forall i = 1, 2...N$

Pre-train worst-case network on datasets $\mathcal{D}^T$ with greedy action a_g by minimizing loss function in Eqn (4.6).

For $t = 1, 2...T$

Randomly sample batched optimization parameter $x \in \mathcal{D}^T$.

For $i = 1, 2...M$

Calculate the baseline value $V(x)$ with Algorithm 4.

Predict the variable candidates distribution $P_i = P_{\theta_i}(a|x)$ from policy networks.

Randomly sample an prediction action a_i based on the probability P_i.

For $j = 1, 2...N$

Predict worst-case error δ_i^j for current action x_i using worst-case network with θ_j^w.

Calculate the robust objective value $g_i = \max_{j=1}^{N} \ f(x + \delta_i^j, a_i)$.

Apply gradient descent with $\nabla_{\theta_i^p} J = -\left[g_i - V(x)\right] \nabla_{\theta_i^p} P_{\theta_i^p}(a_i|x_i)$.

Return: policy network parameters θ_i^p and worst-case network parameters θ_j^w.

robust CO problem in Eqn (4.2). We can formulate the target of worst-case DNNs into problem 4.5. For simplicity, we assume a L_2-norm constraint in the following discussions.

$$\max_{\delta} \ f(x+\delta, a) \tag{4.5}$$
$$\textbf{s.t.} \ \ |\delta|_2 \leq \epsilon$$

where ϵ is the L_2-norm error budget. According to 'L2O method' and 'Lagrange multiplier method', the worst-case DNN models can be directly trained by minimizing the respect loss function in Eqn (4.6), which doesn't requires supervised labels.

$$\mathcal{L}(x, a, \delta) = -f(x+\delta, a) + \lambda \left[|\delta|_2 - \epsilon\right]^+ \tag{4.6}$$

where $[\cdot]^+ = \max(\cdot, 0)$, and λ is a hyper-parameter of Lagrange multiplier indicating the cost of constraint violation. Since the error δ only relies on (x, a), the worst-case networks can be pre-trained separately. Note that, some state-of-the-art (SOTA) methods (e.g., *scipy* optimization package [154]) can be also be applied when developing worst-case critic module.

Besides worst-case DNNs, **a calculation model** is implemented to provide feedback for policy models. The calculation model can provides the worst-case objective of current action a, by comparing the objects of all worst-case errors $\delta_1, ...\delta_N$. Specifically, the worst-case object can be calculated as $g = \max_j f(x+\delta_j, a)$. Besides, the baseline value $V(x)$ can be estimated with critic model, which is summarized as Algorithm 4.

To summarize, in the training phase, we first design the worst-case module and pre-train the respective worst-case networks using the *'learning to optimize'*. Then, the policy model (maybe ensembled) can be trained using the policy gradient and worse-case

Algorithm 4 Baseline Estimation Algorithm

Input: Input parameter x, variable probability P_i, worst-case network parameter θ_j^w $\forall j \in 1, 2, ...N$, sampling size S, object $f(x, a)$.

Output: Estimated Baseline Value $V(x)$.

For $s = 1, 2...S$

Sample action a_s according to P_i.

For $j = 1, 2...N$

Predict the worst-case error δ_s^j from worst-case network j with paramter θ_j^w.

Obtain worst-case object $g_s(x) = \max_{j=0}^{N} f(x + \delta_s^j, a_s)$

Calculate the baseline value $V(x) = 1/|S| \sum_{s=1}^{S} g_s(x)$.

Return: Baseline Value $V(x)$.

modules. In the inference phase, given an optimization parameter x, we first obtain the action probability p_a. Then, a set of candidate actions $\{a\}$ can be sampled and selected based on the critic model. The complete algorithm is shown in Algorithm 3.

4.4 Application I: A Toy Example

In this section, we show an illustrative example to validate the effectiveness of RobustPN. Specifically, the combinatorial problem under consideration is formulated as follows:

$$f(x, a) = -\sum_{i=1}^{3} \left[\sqrt{x_i} \cdot \frac{a_i^2}{1 + \sum_{j \neq i} a_j} \right] \tag{4.7}$$

where $x_i \in [0.5, 1]$ is the problem parameter, a_i is the discrete optimization variable (denoted as action), and $\mathcal{A}$ represents the constrained discrete action space $a_i \in \{0, 0.25, 0.5, 0.75, 1\}$. This function is abstracted from a simple discrete power control problem in interference channels with three users: a_i represents the transmit power level for user i, and $\sqrt{x_i}a_i$ is the respect scaling factor.

In this toy experiment, we consider a set of problems formulated in Eqn. (4.7), and target at solving the optimization problem with parameter x. Additionally, a robust solution is required, considering the uncertainty of optimization parameters x. Here, we consider a simple additive-noise uncertainty model for x — $f(x+\delta, a)$ with a L_1-norm error constraints $\|\delta\|_1 \leq 0.2$.

4.4.1 Setup

In this part, we introduce the baseline benchmarks and experiment setup details. All the experiments are executed in *Python* and *Tensorflow*.

Baselines. We propose three baseline algorithms for comparison: Random, Greedy, and PN. The Random benchmark selects random actions a_i from eligible space $\mathcal{A}$. The Greedy method obtains a solution by solving the optimization problem in a greedy fashion over a_i. Initially, the actions variables are set to zero. Then, we solve the problem in three steps. For each step, we optimize the a_i from the remaining variables,by minimizing Eqn. (4.7) with the previously optimized variables. The detailed implementation of Greedy is summarized in Algorithm 5 . Additionally, we propose another baseline algorithm — called PN. The PN learns a baseline policy via policy gradient as shown in Algorithm 3. Here, a similar policy network is used for PN.

Datasets. Considering the toy problem, we generate three datasets: a training dataset with 10k samples for policy training, a validation dataset with 5k samples for hyper-parameter tuning, and a test dataset with 5k samples for performance evaluation. Note that all samples are uniformly generated with $x_i \in [0.5, 1]$ and each sample can be viewed as a single minimax problem with different parameters.

Policy model and training. The policy network is implemented with neural networks with a shared embedding layer, an LSTM layer with 100 hidden nodes, and a fully-connected output layer with 5 nodes and the softmax activation function. The model is trained by Adam optimizer [79] as shown in Algorithm 3. To avoid gradient explosion, exponentially decreasing learning rate and gradient capping are employed. To be specific, the learning rate is initialized as $1e^{-3}$ and decreases by a factor of 0.9 for every 20 epochs. In the inference phase, 200 actions are sampled based on the output probability P_θ, and an optimal action is reported by the action selection module according to Fig 4.3. For simplicity, we only train one policy network ($M = 1$) for the toy example. Note that the PN is also implemented with a similar network architecture but trained with a critic module with the original objective (e.g., $f(x, a)$).

SOTA method and worst-case networks. The worst-case module involves a SOTA algorithm and several ensemble neural network models, which accepts the parameter-action pair (x, a) and outputs the worst-case error δ. Since f is continuous on $\delta \in \mathbb{R}^3$, we implement the SOTA with exhaustive search with a step size of 0.002. For high dimensional δ, the SOTA algorithm can be implemented using the constrained-minimization algorithm provided by the *SciPy* package [154].

Algorithm 5 Greedy Algorithm

Inputs: parameters x, number of action variables $L = 3$, and object function $f(x, a)$.

Outputs: greedy action a.

Initialization: actions $a = [0, 0, 0]$, undetermined action set $S = \{a_1, a_2...a_L\}$.

While S is not empty

Initialize best variable index $b = null$.

Initialize best action value $a_{value} = null$.

Initialize best current object $f_b = \infty$.

For $a_i \in S$

For a_i in $\{0, 0.25, 0.5, 0.75, 1\}$

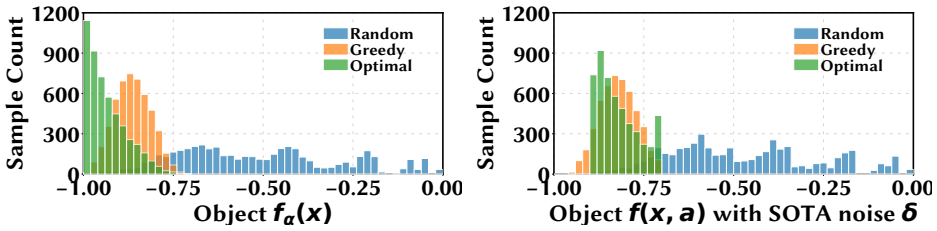
If $f(x, a) < f_b$

$b = i$, $a_{value} = x_i$, and $f_b = f(x, a)$.

$a[b] = a_{value}$, $S = S \backslash a_i$.

Return: action a.

Additionally, we pre-train four worst-case networks, to serve as a complementary of SOTA. Specifically, four networks are implemented with feed-forward neural networks with two hidden layers and 100 hidden nodes. The input of the worst-case network involves parameters x and actions a, while the output represents the worst-case error δ. Note that four models are optimized on the training dataset with the same parameters x but actions a from different baselines. The worst-case networks are trained by directly minimizing the loss function in Eqn. (4.6), with Adam optimizer ($lr = 1e^{-4}$) over 400 epochs [79]. The respective hyper-parameter λ is selected on the validation dataset for each model. Note that the exhaustive search SOTA can provide nearly-optimal worst noise but require extensive



(a) (b)

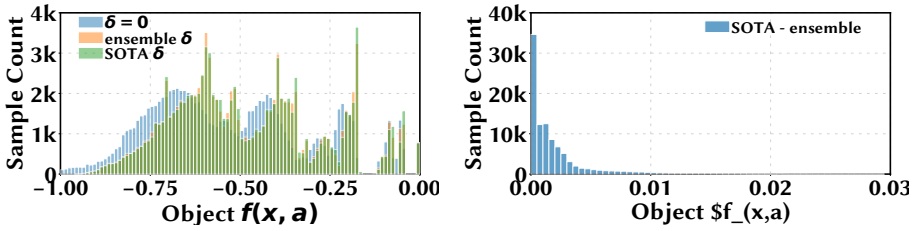
Figure 4.4: Distribution of objective values on test dataset for Random, Greedy and Optimal. (a) without noise ($\delta = 0$). (b) noise δ from SOTA.

computations. Here, we use worst-case network in the training phase and SOTA algorithm in the result evaluation phase. Note that the resulting δ from the worst-case network will be normalized to $L_1(\delta) = 0.2$ if and only if $L_1(\delta) > 0.2$.

4.4.2 Experiment Result

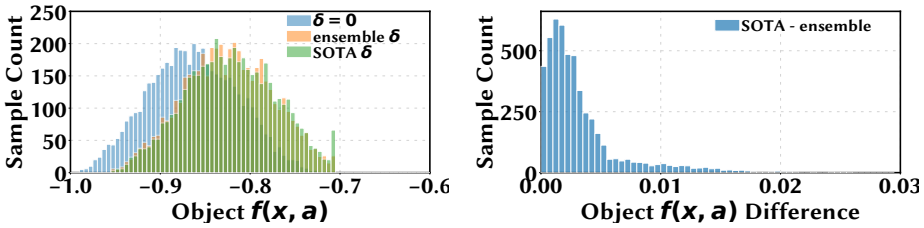
In this part, we first present the robust performance of baseline policies. For comparison, the optimal solution is also provided by the exhaustive search. Then, we discuss the performance of pre-trained worst-case networks. Finally, the object function values are evaluated on the test dataset under machine learning-assisted methods.

Benchmark evaluation. The object values of different baselines are shown in Figure 4.4(a). The result demonstrates that the Greedy algorithm presents a much better solution than the Random policy. Then, for each policy action, we calculate the worst-case noise δ under L_1 norm $||\delta||_1 \leq 0.2$. The worst-case performance is presented in Fig. 4.4(b), respectively. Although the optimal solution obtains smaller f, it is not robust to the SOTA



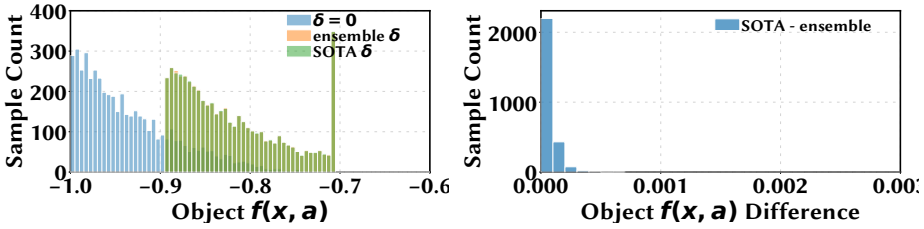
(a) (b)

Figure 4.5: Performance of the worst-case networks compared with the SOTA algorithm on test dataset with 20 randomly selected actions. (a) Distribution of objective values. (b) Distribution of objective value difference between the SOTA algorithm and ensemble networks.



(a) (b)

Figure 4.6: Performance of the worst-case networks compared with the SOTA algorithm on test dataset with **greedy actions**. (a) Distribution of objective values. (b) Distribution of objective value difference.



(a) (b)

Figure 4.7: Performance of the worst-case networks compared with the SOTA algorithm on test dataset with **optimal actions**. (a) Distribution of objective values. (b) Distribution of objective value difference between the SOTA algorithm and ensemble networks.

noise, resulting in significant performance degradation. This clearly provides the motivation to develop a policy that is robust to the SOTA noise.

Worst-case network performance. Here, we evaluate the performance of worst-case noise, produced by pre-trained neuron networks and SOTA, on the test dataset with random actions. Specifically, we randomly sample 20 available actions for each parameter x. Then, the worst-case δ is generated from pre-trained ensemble worst-case networks and SOTA algorithm separately. Finally, we visualize the histogram distribution of respective object values under different noise δ, as shown in Fig. 4.5(a). Additionally, the object difference when using SOTA's δ and ensemble networks' δ is presented in Fig. 4.5(b). Besides, the object distribution and the object difference between SOTA and ensemble networks are presented in Fig. 4.6 for Greedy actions and Fig. 4.7 for Optimal actions.

Since the SOTA outputs the optimal δ for input (x, a), the difference is always a positive value, indicating the performance loss of using the ensemble worst-case networks. The closer this value is to zero, the better for the worst-case networks. The results show that Optimal actions are very sensitive with SOTA noise, with the largest performance degradation among all actions, which further validates the necessity of robust actions for the combinatorial problem. Also, the results successfully verify that ensemble networks can provide a near-optimal worse-case noise δ. Although they have similar performance, we note that the ensemble networks can provide worst-case noise much faster than SOTA.

Performance of the learned robust policy. Now, we present the performance of RobustPN, including for the objective with no noise ($\delta = 0$) and with SOTA δ. Specifically, the objects $f(x+\delta, a)$ are calculated on the 5k test dataset and the average values are presented in Table 4.1 for each method, respectively. For better comparison, we also present the optimal results: Optimal method for best policy under $\delta = 0$ and Robust Optimal method

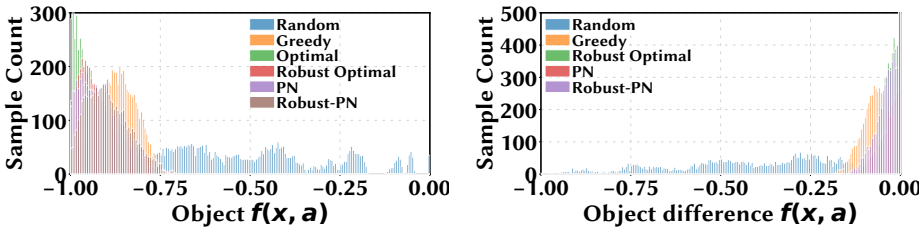
Methods	$f(x, a)$ for $\delta = 0$	$f(x, a)$ for SOTA δ
Random	-0.5197	-0.4763
Greedy	-0.8631	-0.8212
Optimal	**-0.9342**	-0.8220
Robust Optimal	-0.9158	**-0.8469**
PN	-0.9143	-0.8308
RobustPN	-0.8966	-0.8390

Table 4.1: Performance of learned policies: the average objective values under different δ.

for best robust policy under SOTA δ. Note that both Optimal and Robust Optimal are calculated by exhaustively searching over the entire eligible action space $\mathcal{A}$. In Robust Optimal, given input parameter x, we first calculate the SOTA robust object $f(x+\delta, a)$ for all eligible actions a, where δ is the SOTA noise given (x, a). Then, the best action for Robust Optimal is selected with the lowest robust object.

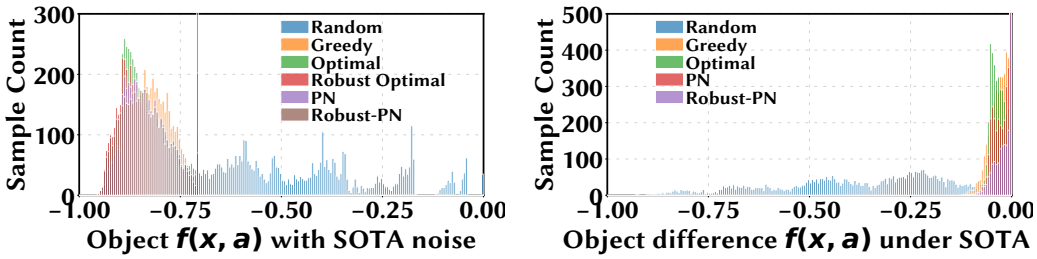
The results in Table 4.1 show that the proposed RobustPN provide better performance than Random and Greedy when the true parameter information is provided ($\delta = 0$). The average objective value of RobustPN is slightly worse than PN and Optimal for $\delta = 0$, which is often observed for a robust solution. Meanwhile, RobustPN can learn a robust policy that obtains smaller (better) objective values than other baselines under SOTA noise. Note that although Optimal can provide the best performance under $\delta = 0$, RobustPN is the best solution under the SOTA noise δ.

Additionally, we plot the histogram distribution of objective values in Figures 4.8(a) and 4.9(a) for zero noise and SOTA noise, respectively. For better visualization, we further plot the objective differences, as shown in Figures 4.8(b) and 4.9(b). Note that the Optimal policy is utilized as the base values to calculate the differences in Figure 4.8(b),



(a) (b)

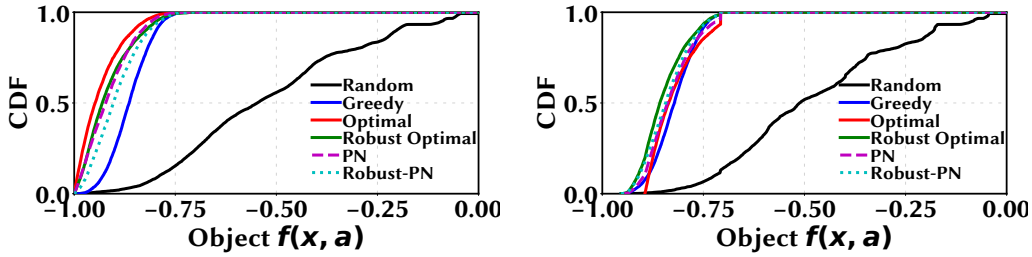
Figure 4.8: Distribution of objective values on test dataset for all methods($\delta = 0$). (a)Objective values. (b)Objective value differences. The difference is calculated between respective policy and the **Optimal policy**.



(a) (b)

Figure 4.9: Distribution of objective values on test dataset for all methods(SOTA noise). (a)Objective values. (b)Objective value differences. The difference is calculated between respective policy and the **Robust Optimal policy**.

while the Robust Optimal policy is utilized to calculate the differences with SOTA noise in Figure 4.9(b). For the objective differences, the closer to zero, the better. Then, Figure 4.10 illustrates the cumulative distribution function (CDF) for all policies with/without noise. Figure 4.10(a) shows that learned policies (PN and RobustPN) achieve better performance than other baselines without noise. Then, Figure 4.10(b) shows that RobustPN performs better than PN and other baselines with SOTA noise. Hence, the histograms and CDF plots further demonstrate that RobustPN can provide near-optimal performance under SOTA noise and at the same time maintain good performance under $\delta = 0$.



(a) (b)

Figure 4.10: CDF plots for objective values on test dataset for all methods. (a) without noise ($\delta = 0$). (b) with noise δ from the SOTA algorithm.

In summary, we successfully apply the proposed RobustPN to a toy example for resource allocation problem shown in Eqn. (4.7). The corresponding results demonstrate that the RobustPN is an effective and alternative approach for the minimax problem with discrete actions. By using machine learning-assisted methods, the proposed RobustPN can provide fast and robust solutions for general combinatorial problems. In next section, we will apply the proposed RobustPN to the replicated task offloading problem in edge computing.

4.5 Application II: Vehicular Computing Resource Allocation

Besides the toy example, the proposed RobustPN method is applied to the replicated task offloading problem in a vehicular edge computing (VEC) system. In this section, we firstly introduce the respect optimization problem existing in the current VEC system. Then, RobustPN is utilized for a robust solution. The robustness is discussed with two error budgets. Finally, the respective results are presented and discussed, further validating the effectiveness of the proposed RobustPN.

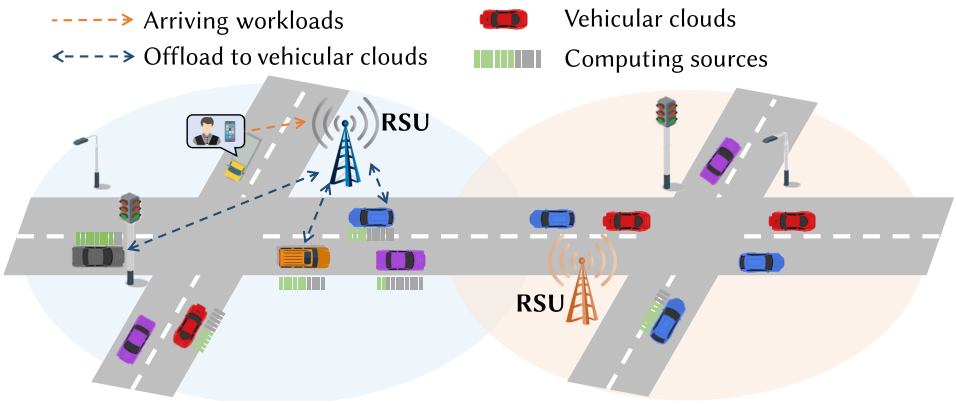


Figure 4.11: VEC system diagram and the offloading illustration.

4.5.1 Replicated Task Offloading in VEC

With the development of the internet of things (IoT), many modern mobile devices (e.g., smart vehicles, smartphones, etc.) are equipped with powerful computing resources, such as multi-core CPUs. Therefore, vehicular clouds are emerging as a promising edge computing architecture, as complementary to conventional edge computing. Vehicular cloud treats vehicles as data centers, providing computing services using extra resources [36].

Specifically, we consider a VEC system with a road-side-unit (RSU) and N vehicular clouds. The RSU is installed at a fixed location, which gathers computing tasks from edge devices and then offloads them to nearby vehicular clouds via wireless communication networks. Usually, the total service delay d is comprised of two parts: the data transmission delay d^t and service computing delay d^c in the respective vehicular cloud. By default, each task arrives with a deadline time (L^t) and the task fails when the service delay excesses L^t.

Due to the unstable environment (e.g., fast-changing distance), the success rate of each offloading is hard to estimate. To achieve a higher success rate, each arriving task

is replicated and offloaded to multiple vehicular clouds [20, 170]. Figure 4.11 illustrates a typical diagram for a VEC system with replicated offloading. Here, we are focusing on designing a robust policy for the RSU offloading process, considering the overall success rate. We define the success probability of each replication as x_i when offloading to vehicular cloud v_i. Then, the overall task success rate can be calculated via the *At-Least-One* rule as shown in Eqn (4.8), where a_i represents the offloading action.

$$P(\mathbf{x}, \mathbf{a}) = 1 - \prod_{i=1}^{N} (1 - x_i a_i) \tag{4.8}$$

Besides the success rate, we need to consider the cost of each replication. Here, we denote η_i as the unit cost for replication i. To simplify, we assume that the cost is the same for each vehicular cloud. The expected reward given the selected clouds can be formulated as Eqn (4.9). The optimal offloading policy can be obtained by solving the respect optimization: $\max_{\mathbf{a} \in \mathcal{A}} U(\mu, \mathbf{a})$, where $\mathcal{A}$ is the eligible action space.

$$U(\mathbf{x}, \mathbf{a}) = \left[1 - \prod_{i=1}^{N} (1 - x_i a_i) \right] - \sum_{i=1}^{N} \eta_i a_i \tag{4.9}$$

where $x_i \ \forall i \in \{1, 2...N\}$ is the problem parameter (success rate for vehicle i) and a_i represents the respect binary offloading decision.

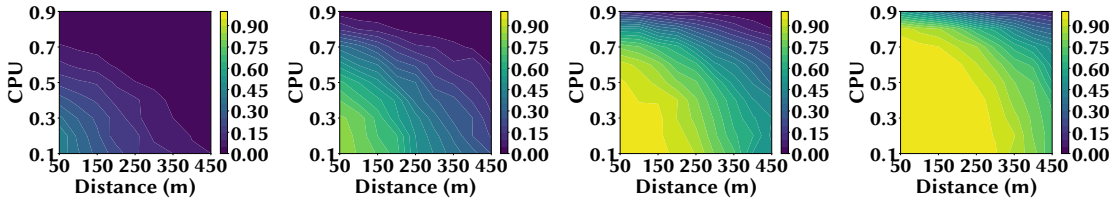
The optimization object (4.9) is a non convex function. And the expected total rewards is not a simple addition on each individual reward $(x_i - \eta)$. Instead, the first term of object is a sub-modular function [84]. Additionally, the problem(4.9) is even harder because of the discrete action space $\mathcal{A}$. Actually, it belongs to combinatorial optimization problems, which are NP-hard and cannot directly solved by conventional convex optimization method.

In Eqn (4.9), the success rate x_i for each replication is usually predicted by a estimation model based on other environment conditions. We denote these factors as context information c_i, which includes the distance l to vehicular cloud's, the CPU speed *cpu* of cloud server, and task's deadline time t. The entire system can be treated as a "*Predict, then optimize*". According to [35], the prediction errors may have huge impacts on the final performance. Similarly, we will consider a robust optimization (mini-max problem) in Eqn (4.10), where Δ is the error budget of the prediction model. Here, we consider a L2-norm budget, which can be represented as $\Delta(x^{true}) = \{x,\ \forall |x - x^{true}|_2 \leq \epsilon\}$. Respectively, we evaluate and discuss the performance using proposed RobustPN and other baselines for large and small error budget.

$$\underset{\mathbf{a}\in\mathcal{A}}{\operatorname{argmax}} \min_{\delta\in\Delta(\mathbf{x})} U(\mathbf{x}+\delta, \mathbf{a}) \tag{4.10}$$

4.5.2 Setup and Error Budgets

Under the "*Predict, then optimize*" platform, the error budget depends on the upstream prediction model. In a real-world application, the selection of prediction models is determined by the trade-off between interpretability and accuracy. For example, an empirical model is usually explainable but with a high error, whereas the elusive DNN model can provide low error prediction. In this part, we simulate the context dataset for replicated task offloading problem and discussed the error budget of two prediction models: linear regression and residual model implemented with DNN.



(a) $L_t = 0.25$ (b) $L_t = 0.5$ (c) $L_t = 0.75$ (d) $L_t = 1.0$

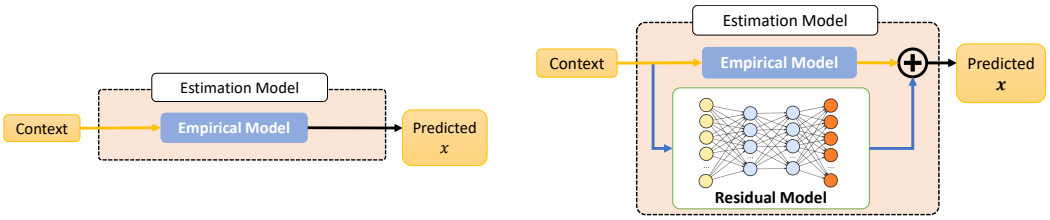
Figure 4.12: Success rate simulation result for different deadline time L.

Latency simulation model. The replicated task v_i successes when the computation is finished within the deadline time L: $d = d^t + d^c \leq L$. Here, the transmission time d^t can be calculated based on the wireless model in Eqn (4.11) [20].

$$d^t = \frac{S_{data}}{W \cdot \log_2\left(1 + \frac{P \cdot (d)^{-\alpha}}{\sigma^2 + I^t}\right)} \tag{4.11}$$

where S_{data} is the offloading data size, P is the transmission power, α is the channel gain factor with respect to distance, σ^2 is the random noise and I^t indicates the interference noise. Then, the *cpu* relevant computation time is simulated and obtained based on the M/M/1 queue model [112] using the dataset [124]. Our computation latency model can be simulated as $d^c = \frac{a}{b - cpu} + noise \simeq \frac{0.227}{2.15 - cpu} + 0.007 \cdot \mathcal{N}(0, 1)$. As for the transmission model, we set the parameter as followings: $S_{data} = 3Mb$, $W = 10M$,$P = 10dBm$,$\sigma^2 = -172dBm$, $I^t \in [-10, -30]dBm$ and $\alpha = 1.8$ in our simulation. Besides, the distance measurement error of vehicles is set as 3% (10m) according to the recent GPS manual [1].

Dataset. Utilizing the above latency model, we can generate the context dataset with the resulting success rate. The success rate of each offloading is as the average over 1000 simulation rounds. The respect simulation results are illustrated in Figure 4.12 for



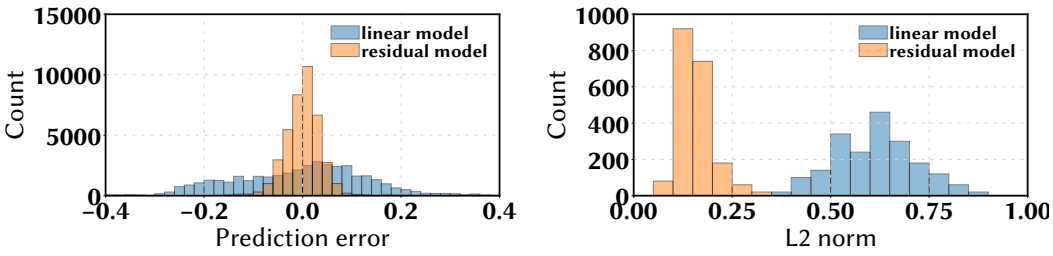
(a) Emperical model (b) Residual model

Figure 4.13: Success rate estimation models.

different context information. In total, we generate a training dataset with 15k instances, a validation dataset with 4k instances, and a test dataset with 6k instances. Since $N = 20$ is considered in offloading problem, each instance includes the information for 20 vehicles, indicating the context and respective success rate.

Estimation Models. We propose two estimation models, which predict the success rate x based on the context: $x = M_p(d, cpu, t)$. The empirical model in Figure 4.13(a), is implemented with linear regression $\hat{x} = w_d d + w_{cpu} cpu + w_L L + bias$, which is easy to understand. Besides, we also provide a residual model, including the pre-trained linear model and a parallel residual block. The residual block is implemented with a neural network, to learn the residual error. The detailed architecture of the residual model is illustrated in Figure 4.13(b).

In the training phase, both empirical and residual model is trained with the same training dataset. The empirical model is trained by *'LinearRegression'* module in *scikit-learn* package. Then, the residual model is implemented and trained using *'keras'* and *'tensorflow'* packages. Specifically, the residual block includes two hidden layers with 20



(a) Estimation error

(b) L2 norm of estimation error

Figure 4.14: Success rate estimation error and its L2-norm for linear and residual model.

neurons and *'relu'* activation function. The residual block is trained by minimizing the *'mean squared error'* loss function using the Adam optimizer (learning rate $lr = 1e - 4$).

Prediction error and discussion. The predicted success rate of the linear model is denoted as $\hat{x}^l$, while $\hat{x}^r$ represents the result of the residual model. Then, two errors are investigated: the individual error $\hat{x}_i - x_i$ and the l2-norm distance of 20 predictions ($\sum_i |\hat{x}_i - x|_2$) for each instance. The estimation errors on the test dataset are summarized in Figure 4.14 for individual error and l2-norm distance, respectively. The results validate that the residual model indeed reduces the estimation error. Therefore, we will consider the robust offloading problem with both a large error budget and a small error budget. Here, we consider the 99% percentile as the error budget for robustness, which is 0.71 for the empirical model and 0.27 for the residual model.

4.5.3 Results with A Large Error Budget

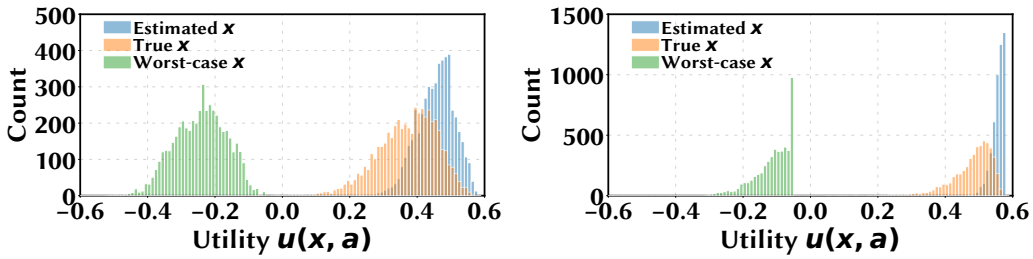
Firstly, we investigate the robustness of replicated task offloading with a large prediction error budget $\Delta = \{|\delta|_2 \leq 0.71\}$, when an empirical linear model is used for success rate prediction.

Setup. We generate three datasets from the linear prediction model: a 15k train dataset, a 5k validation dataset, and a 5k test dataset. One data sample includes 20 items, which represents the information for $N = 20$ eligible offloading vehicular clouds, including the context information c_i, true success rate x_i, and predicted success rate $\hat{x}_i$. Similar to the toy example, three baseline algorithms are used for comparison: Random Greedy and PN. The detailed implementation is presented in section xx. Besides, we proposed two oracle benchmarks using the brute-force method, which is time and computationally expensive (exponential time $O(2^N)$). The Weak Oracle provides actions based on predicted parameter $\hat{x}$, while the Oracle use the true parameter x.

Evaluation metrics. Considering the robustness, we define three evaluation metrics. The estimated utility is calculated with the predicted parameter $\hat{x}$, denoted as $U(\hat{x}, \hat{a})$. Then, the true utility is $U(x, \hat{a})$, where x is the true parameter value. Finally, the worst-case utility is represented as $\min_{\delta} U(\hat{x}+\delta, \hat{a})$, which can be solved using our worst-case module or the python optimization package.

Robustness demo. To verify the robustness concern in replicated task offloading, we perform a demo simulation for Myopic and Weak Oracle offloading policies on the test dataset. The respect histogram plots of evaluation metrics are shown in Figure 4.15. The results demonstrate that the true utility is lower than the estimation, because of the prediction errors. Then, the performance seriously degrades under the worst-case x, which further motivates the design of the robustness offloading policy.

Worst-case models. Additionally, we pre-train four worst-case models, to solve the inner worst-case part in Eqn (4.10). Specifically, the worst-case model provides a map-



(a) **Greedy** (b) **Weak Oracle**

Figure 4.15: Robustness demo for Greedy and Weak Oracle policies.

ping from parameters action pair x, a to the worst-case error δ. According to Eqn (4.9), utility is only impacts by δ_i with $a_i = 1$. Hence, we design the worst-case model using the feed-forward neural networks shown in Figure xx, including two hidden layers with 400 neurons and one customized production layer. The worst-case model is directly trained with the L2O method by minimizing the loss in Eqn 4.10. Note that four models are trained on the same training dataset but different λ. Besides, we validate our worst-case models using the solver in *scipy.optimize* package [154]. The worst-case results are presented in Figure 4.17 for Oracle policy. Figure 4.17(a) shows the utility with worst-case parameter x, and Figure 4.17(b) is the utility difference between ensemble method and solver method. The near-zero differences demonstrate that our ensemble worst-case model can approximate the solver. Since the solver is much slower ($\sim$70X) than the worst-case models, the ensemble worst-case networks are used for the model training and the performance evaluation in the following parts.

Policy Model. The robust optimization model (RobustPN) can be directly trained with the policy gradient method on the actor-critic architecture shown in Figure xx.

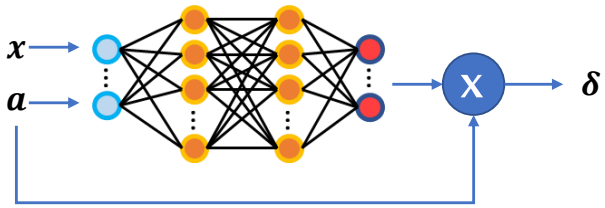
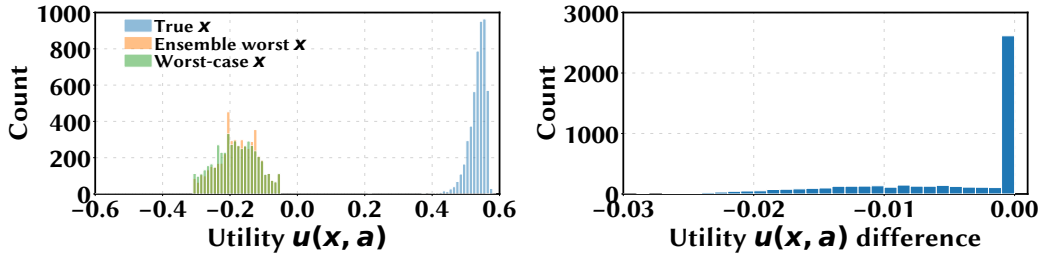


Figure 4.16: Worst-case DNN architecture.



(a) (b)

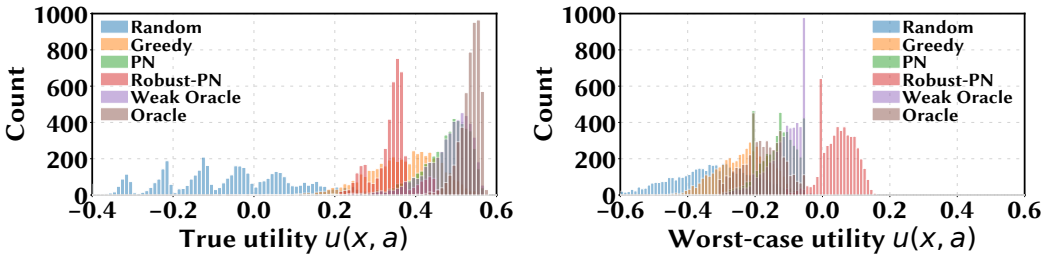
Figure 4.17: Worst-case networks performance on test dataset with Oracle policy.

Here, the policy model is implemented with neural networks with a shared embedding layer, an LSTM layer with 100 hidden nodes, and a fully connected output layer with 20 neurons. The 'sigmoid' activation function is used in the output layer. Similarly, the policy model is trained by Adam optimizer ($lr = 1e-4$) [79] using the Algorithm 3. When using the policy model, 1000 actions are randomly sampled on the output probability P_θ, and an optimal action is reported by the action selection module according to Fig 4.3. Note that the PN is also implemented with the same architecture.

Performance evaluation. We first present the performance of RobustPN for all the evaluation metrics, comparing with other benchmarks. The average performances on test dataset are summarized in Table 4.2. Here, we are focusing on the true utility (3rd column) and worst-case utility (4th column). The results show that the proposed RobustPN

Method	Estimated x	True x	Worst-case x
Random	-0.0314	-0.0716	-0.3237
Greedy	0.4554	0.3704	-0.2325
PN	0.5494	0.4719	-0.1347
RobustPN	0.3319	0.3054	**0.0467**
Weak Oracle	0.5568	0.4798	-0.1113
Oracle	0.5319	**0.5347**	-0.1765

Table 4.2: Performance of different algorithms under a **large** error budget.

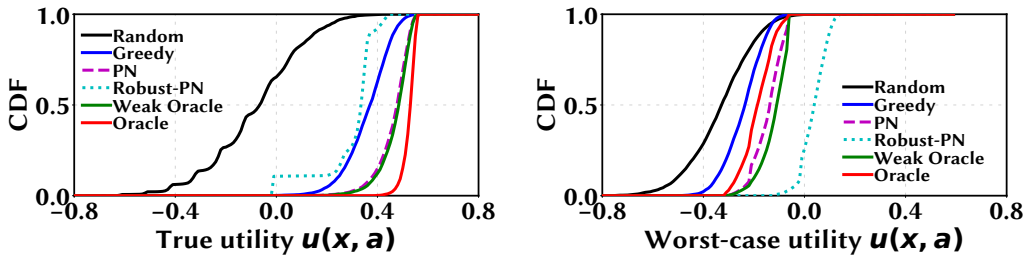


(a) (b)

Figure 4.18: Histogram visualization of utilities under a **large** error budget. (a) True utilities. (b) The worst-case utilities.

provides the best performance among all benchmarks (including the oracle methods) under the worst-case condition, demonstrating the robustness of RobustPN. Meanwhile, the true utility of RobustPN degrades comparing to PN. This is because RobustPN is focusing on worst-case robustness under a large error budget. As a side-product, the PN can provide near-optimal actions compared with Weak Oracle, validating that L2O can be used to solve combinatorial optimization problems.

Besides the average value, we visualize the respect distribution in Figure 4.18(a) for true utilities and 4.18(b) for worst-case utilities. For better visualization, Figure 4.19 illustrates the cumulative distribution function (CDF) for all methods. These results show that the PN methods can provide near-optimal results considering the true utility, but a



(a) (b)

Figure 4.19: CDF plots of utilities under a **large** error budget. (a) True utilities. (b) The worst-case utilities.

Method	Estimated x	True x	Worst-case x
Random	-0.067	-0.0716	-0.1597
Greedy	0.4236	0.4167	0.2383
PN	0.5321	0.5216	0.3442
RobustPN	0.5078	0.5008	**0.3727**
Weak Oracle	0.5393	0.5290	0.3594
Oracle	0.5336	**0.5347**	0.3501

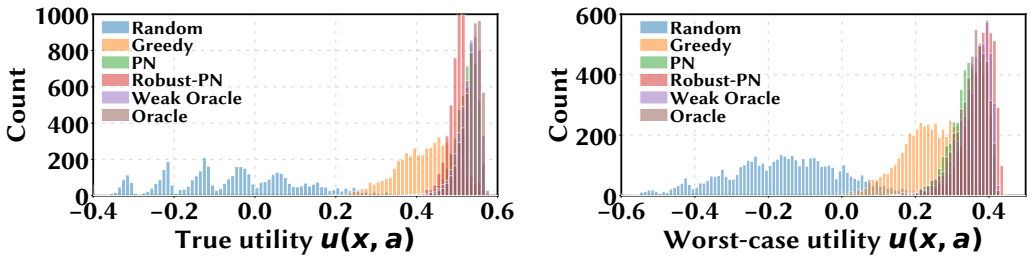
Table 4.3: Performance of different algorithms under a **small** error budget.

performance gap exists between PN and Oracle because of the large error budget. Hence, we will evaluate the performance with a small error budget in the next section. Further, the RobustPN can provide better robust performance than oracle algorithms.

In summary, the proposed RobustPN is applied to replicated task offloading problems, considering the robustness. The results validate that RobustPN is an effective method to solve robust combinatorial problems.

4.5.4 Results with A Small Error Budget

As shown in Table 4.2, the true utility degrades a lot for proposed RobustPN methods due to the large error budget. Now, we turn to investigate the performance under



(a) (b)

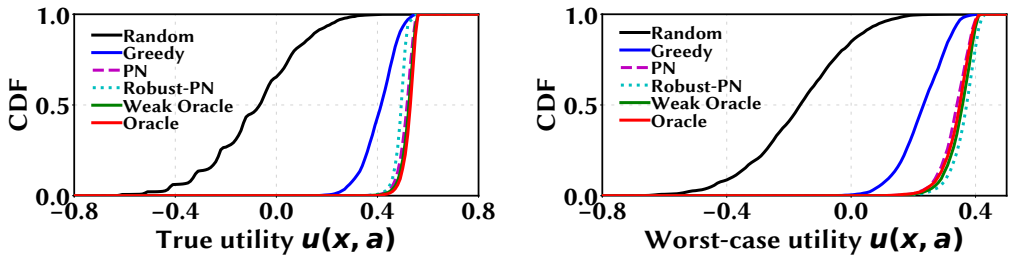
Figure 4.20: Histogram visualization of utilities under a **small** error budget. (a) True utilities. (b) The utilities under the worst-case x.

a small prediction error budget $\Delta = \{|\delta|_2 \leq 0.27\}$, according to the prediction result of the residual model.

Setup. The baselines and oracle policies are calculated based on the predicted $\hat{x}$ from the residual model. A large error budget is utilized for the training of worst-case models and the calculation of robust evaluation metrics. All the other settings, such as datasets and model architectures, are the same as section 4.5.3.

Performance evaluation. The performance of RobustPN is evaluated under small error budget, comparing with other algorithms. Table 4.3 summarizes the average utility under different scenarios, including different polices and different x. Similarly, the respect histogram plots are shown in Figure 4.20 and the cumulative distribution function (CDF) is illustrated in Figure 4.21.

Comparing to the results in the last section, the proposed RobustPN can still provide a more robust solution than other benchmarks including the oracle ones. But, the performance under true success rate x is much closer to oracle algorithms, because of the



(a) (b)

Figure 4.21: CDF plots of utilities under a **small** error budget. (a) True utilities. (b) The utilities under the worst-case x.

shrinkage of error budget $\Delta(x)$. This also demonstrates that our method with a smaller error budget can guarantee both the real performance and the robust performance.

Currently, we explore the offloading problem with the same cost η. In a real-world scenario, the η may be different across vehicular clouds. But, the cost η is usually a pre-known parameter, which will not have a big impact on our simulation results and be left as a research topic for future exploration.

4.6 Related Works

With advanced deep learning techniques, data-driven machine learning-assisted methods (e.g., deep neural network) are directly applied to the decision-making and resource-planning processes, such as combinatorial optimization and Markov decision process. In this section, we summarize the recent related works on these machine learning-assisted methods and their usage for various optimization problems.

Machine learning for combinatorial optimization. In general, combinatorial optimization (CO) can be formulated as a minimize optimization problem with discrete constraints (e.g, integer). Due to the huge search space, CO problem often requires a large

amount of time to solve directly [16, 83]. For instance, the traveling salesman problem (TSP) requires $O(2^n n^2)$ time to be fully solved, using the best dynamic programming algorithm. Hence, many heuristic algorithms are proposed to achieve near-optimal solutions in polynomial time. With deep learning's remarkable success in various industry areas, machine learning-assisted methods are proposed to solve CO problems directly. Overall, machine learning-assisted methods for CO problem falls into two main categories according to the learning techniques: supervised learning and reinforcement learning [16].

The supervised learning methods concentrate on approximating some heuristic policies by using DNN models, to alleviate the computation burden - called learning by imitation. For instance, [117] introduces cross-entropy (CE) method for CO problem, where the data is collected based on a specified policy. Also, [153] proposes Pointer Networks for the TSP problem, and the pointer network is trained with labels collected from an optimal or heuristic method. [76] designs a machine learning framework to approximate a well-performing but time-expensive branch&bound method — called "strong branching" — for mixed-integer problems. However, there are two obvious drawbacks to these supervised learning methods. Firstly, a large amount of labels is required for DNN model training. It is usually time-expensive to calculate the labels. Also, the DNN model's performance heavily depends on the labels (or the labeling algorithm). In general, the models learned by supervised learning cannot discover better solutions than the original labeling policy [15].

For the reinforcement learning method, the policy model is instead trained with the supervision rewards from the critic module/function [115]. Directly supervised by the rewards, we may discover better policies than supervised methods using the conventional

heuristic labels. For the TSP problem, a neural combinatorial network is proposed in [15] by using reinforcement learning and sampling, which achieves close to optimal performance. Besides, [32] introduces policy gradient to CO problem with attention mechanism. In [82], the author proposes an attention model reinforced with greedy baselines for CO. Besides, [11] applies reinforcement learning framework to Maximum Cut problem. [26] propose a novel graph embedding and reinforcement learning architecture to tackle CO problems over graphs, including Minimum Vertex Cover, Maximum Cut problems, etc. [152] use deep reinforcement learning to solve CO problem on graphs for networking applications. Although better policies can be trained, these works assume perfect input context information without considering the policy robustness, which is unrealistic for real-world applications. In contrast, we consider the combinatorial problem with imperfect parameters and target a robust policy by using machine learning-assisted methods.

Learning to optimize (L2O). Recently, many data-driven methods are proposed to train a parameterized model for general optimization problems — called L2O, where the model parameters are learned from well-prepared data using machine learning methods. Particularly, L2O targets solving similarly or repeated problems quickly via machine learning. [136] train a DNN model for wireless resource management problems via supervised learning. The DNN model is trained by minimizing the mean square error, and the labels are calculated from the WMMSE method [125]. [90] also proposes L2O for the resource management, but the allocation model is directly trained using an unsupervised method without labels. Since no labeled data is required, such unsupervised learning can be applied to the problem, where the correct labels are hard to collect. Similarly, [135] introduce unsu-

pervised deep learning method to wireless communication problems with quality-of-service (QoS) different constraints. Note that the worst-case networks are trained in a similar method in DeepPM, and an advanced ensemble model is explored for better performance.

Besides, some L2O works concentrate on learning an optimizer for continuous optimizations. This idea is firstly proposed simultaneously in [8] and [89], where a reinforcement learning framework is utilized to train DNN models as optimizers for unconstrained continuous optimization. The optimizer model is implemented with a feed-forward neural network in [89], while an RNN model is proposed in [8]. Then, [21] extend RNN L2O optimizer to black-box function without using the derivative information. [164] introduces a modified RNN architecture trained with ensemble losses, minimizing the per-parameter overhead for optimizer training. In [19], the ensemble LSTM models and attention mechanisms are utilized to train L2O optimizers, particularly for Bayesian swarm optimization. These L2O works mainly focus on a continuous optimization problem, but our work targets the discrete optimization problem, which usually is a harder one. Additionally, L2O has been introduced to adversarial training problems, specifically for the noise generation process [72,166]. Here, the noise generation process is similar to the worst-case critic module in DeepPM. But our proposed method focuses on robust discrete optimization, which also includes the policy learning for outer minimize process besides the worst-case noise generation.

Mini-max optimization. The robust resource allocation can be formulated as a mini-max optimization problem, which is an NP-hard problem. To solve it, many optimization algorithms are proposed including conventional methods and machine learning-assisted methods. The conventional methods are usually developed based on the gradient descent

method. [105] proposes a **g**radient **d**escent and gradient **a**scent (GDA) method, by alternatively optimize the min function and max function. Due to the convergence and oscillation, more stable GDA algorithm are proposed, including K-Beam [56], optimistic-GDA [27,118], and Follow-the-Ridge [160]. K-Beam provides stable performance by simultaneously tracking K candidate solutions. But more computations are required when K increases.

Due to either low performance or high computation cost, machine learning-assisted methods are emerging as a new solution for mini-max problems. [9] proposes a "learning to optimize" method to solve mini-max problem — called *Twin*-L2O. In [9], two DNN models are trained with reinforcement learning, serving as optimizers for alternative gradient descent and gradient ascent. Specifically, the author trains two LSTM models using a self-defined reward function. And each LSTM model accepts the current variable location (x or y) and provides the respective gradient information (Δx or Δy).

Although our work also focuses on mini-max problems with machine learning-assisted methods, several key differences exist compared to *Twin*-L2O. Firstly, we mainly focus on mini-max for robust combinatorial problems, while *Twin*-L2O is designed for continuous mini-max problems. Note that combinatorial problems are harder than continuous mini-max problems, because of the discrete constraints on optimization variables. Additionally, we propose a different learning method — ensemble policy gradients — to directly solve the mini-max problem, while *Twin*-L2O focuses on optimizer training with rewards. Finally, *Twin*-L2O focuses on optimizer training, which provides results with several iterations. But our algorithm focus on policy networks, conditional on parameters of mini-max, which can directly output the solutions to the mini-max problem.

4.7 Conclusion

In this chapter, we investigate the robust combinatorial optimization problems in computing systems. A learning-based algorithm — called RobustPN, is proposed to achieve fast and robust solutions for combinatorial problems. The key novelty of RobustPN is the introduction of machine learning for robust CO problems. By utilizing the *actor-critic* framework and *learning to optimize* techniques, RobustPN can be trained directly by minimizing the customized loss function, where no supervision labels are required. The results on two applications further validate that a robust solution can be well trained by the proposed learning framework.

Chapter 5

Conclusions

To conclude this dissertation, we provide innovative machine learning-assisted solutions for efficient resource management in edge computing systems. Previously, artificial policies are deployed for resource allocation problems in computing systems. However, they are not suitable to manage multiple resources simultaneously, considering the fast environment fluctuations in edge computing systems. Leveraging the powerful machine learning techniques, more efficient and robust policies are obtained than previous methods. The thesis can be summarized as two parts: part I provides the management of battery and cooling resources using reinforcement learning, which is presented in Chapters 2 and 3; part II is focusing on the general resource allocation problem and its robustness concerns, which are investigated in Chapter 4.

The key challenge for battery and cooling resources management is the correlation between resource state and policy actions. This is because these resources have memories. Hence, we tackle the respect management problem from the perspective of MDP and rein-

forcement learning. For better power efficiency and capacity utilization, the pre-installed battery and cooling resources are utilized as a benefit to boosting the performance of the edge data center in Chapter 2. A deep reinforcement learning method —DeepPM, is proposed to estimate the data center thermal behavior in a model-free manner and utilized it to manage the battery resources while keeping server inlet temperature within safe operating limits. Then, the battery resource can be used in a malicious way, as a weapon by injecting thermal attacks, which is investigated in Chapter 3. The results validate that thermal attack is possible for edge co-location data centers, emphasizing the respect for thermal security concerns and the urgent requirement of a dedicated thermal management in edge data centers.

As for the combinatorial optimization problem in edge computing, the key challenges are rising from its NP-hard nature, the discrete action space, and the robustness concerns. We propose a machine learning framework — RobustPN, solving the robust CO problem efficiently. Here, the discrete action is handled by the policy gradient method with an actor-critic platform. The robust combinatorial optimization is first formulated as a minimax problem and then solved by training ensemble worst-case models with *'learning to optimize'*. According to our experiment results on two applications, the proposed RobustPN can generate a robust policy for combinatorial optimization efficiently.

Finally, we hope that this dissertation can motivate readers to explore more on the following topic in the future:

- Currently, our proposed DeepPM is focusing on centralized battery management for performance boosting, considering only one tenant. A possible future study is to

extend the DeepPM algorithm to the multi-tenant data center, where a distributed resource management problem should be considered.

- We explore the thermal vulnerability in edge co-location data centers, which provides a peek at the security problem of data center cooling systems. Then, how to manage the cooling resources and identify the malicious tenant still require more investigation.

- Our thesis verifies that the machine learning methods guide to solve the general optimization problems. However, current methods don't utilize the prior knowledge of the previous study. Hence, knowledge-based learning for the robust combinatorial problem is one promising topic for future exploration.

- Besides, our RobustPN is designed for general optimization problem without state memories. Then, learning robust solutions for MDP problems would be another promising but difficult topic for future exploration

Bibliography

[1] Global positioning system (gps) standard positioning service (sps) performance standard. https://www.gps.gov/technical/ps/2020-SPS-performance-standard.pdf, 2020.

[2] CloudSuite - The Search Benchmark, http://cloudsuite.ch/.

[3] Poweredge r740xd rack server, https://www.dell.com/en-us/work/shop/povw/poweredge-r740xd.

[4] Calb 100 ah se series lithium iron phosphate battery, https://www.evwest.com/catalog/product_info.php?products_id=51.

[5] 365DataCenters. Master services agreement, http://www.365datacenters.com/master-services-agreement/.

[6] B. Aksanli, T. Rosing, and E. Pettis. Distributed battery control for peak power shaving in datacenters. In *IGCC*, 2013.

[7] Baris Aksanli, Eddie Pettis, and Tajana Rosing. Architecting efficient peak power shaving using batteries in data centers. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2013.

[8] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

[9] Anonymous. Learning a minimax optimizer: A pilot study. In *Submitted to International Conference on Learning Representations*, 2021. under review.

[10] Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, and Sujit Dey. Vehicular and edge computing for emerging connected and autonomous vehicle applications. In *DAC*, 2020.

[11] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[12] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[13] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[14] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[15] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[16] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[17] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv.*, 49(3):41:1–41:27, September 2016.

[18] Luigi Brochard, Vinod Kamath, Julita Corbalán, Scott Holland, Walter Mittelbach, and Michael Ott. *Energy-Efficient Computing and Data Centers*. John Wiley & Sons, 2019.

[19] Y Cao, T Chen, Z Wang, and Y Shen. Learning to optimize in swarms. *Advances in Neural Information Processing Systems*, 2019.

[20] Lixing Chen and Jie Xu. Task replication for vehicular cloud: Contextual combinatorial bandit with delayed feedback. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 748–756. IEEE, 2019.

[21] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, 2017.

[22] Mingxi Cheng, Ji Li, and Shahin Nazarian. DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *ASP-DAC*, 2018.

[23] M. Chiang and T. Zhang. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[24] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 2018.

[25] Mehiar Dabbagh, Ammar Rayes, Bechir Hamdaoui, and Mohsen Guizani. Peak shaving through optimal energy storage control for data centers. In *2016 IEEE International Conference on Communications (ICC)*, 2016.

[26] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[27] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training gans with optimism. *arXiv preprint arXiv:1711.00141*, 2017.

[28] DatacenterKnowledge. NTT plans global data center network for connected cars, `http://www.datacenterknowledge.com/archives/2017/03/27/ntt-plans-global-data-center-network-for-connected-cars`.

[29] DatacenterKnowledge. Vapor IO to sell data center colocation services at cell towers, `http://www.datacenterknowledge.com/archives/2017/06/21/vapor-io-to-sell-data-center-colocation-services-at-cell-towers`.

[30] Dell. Integrated dell remote access controller 9 (iDRAC9) version 3.00.00.00.

[31] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer, 2018.

[32] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer, 2018.

[33] Bingqian Du, Chuan Wu, and Zhiyi Huang. Learning resource allocation and pricing for cloud profit maximization. In *AAAI*, 2019.

[34] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 2021.

[35] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 2021.

[36] Mohamed Eltoweissy, Stephan Olariu, and Mohamed Younis. Towards autonomous vehicular clouds. In *International Conference on Ad hoc networks*, 2010.

[37] Dell EMC. idrac 8/7 v2.40.40.40 user's guide. 2017.

[38] Equinix. Colocation services and SLA, `https://enterprise.verizon.com/service_guide/reg/cp_colocation_equinix_data_centers_sla.pdf`.

[39] Abadi Martin et, al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[40] Tony Evans. The different technologies for cooling data centers. *APC white paper*, 2012.

[41] Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Journal of Machine Learning Research*, 5(Dec):1–25, 2003.

[42] Songchun Fan, Seyed Majid Zahedi, and Benjamin C. Lee. The computational sprinting game. In *ASPLOS*, 2016.

[43] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, 2007.

[44] Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982, 2014.

[45] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices*, 47(4):37–48, 2012.

[46] Peter Xiang Gao, Andrew R. Curtis, Bernard Wong, and Srinivasan Keshav. It's not easy being green. *SIGCOMM Comput. Commun. Rev.*, 2012.

[47] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. ContainerLeaks: Emerging security threats of information leakages in container clouds. In *DSN*, 2017.

[48] Xing Gao, Zhang Xu, Haining Wang, Li Li, and Xiaorui Wang. Reduced cooling redundancy: A new security vulnerability in a hot data center. In *NDSS*, 2018.

[49] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D. Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *ASPLOS*, 2015.

[50] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

[51] Google. Google's Data Center Efficiency. `http://www.google.com/about/datacenters/`, 2019.

[52] Google. Heat containment, `http://www.google.com/about/datacenters/efficiency/external/`.

[53] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *ISCA*, 2011.

[54] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *ASPLOS*, 2012.

[55] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Aggressive datacenter power provisioning with batteries. *ACM Trans. Comput. Syst.*, 31(1):2:1–2:31, February 2013.

[56] Jihun Hamm and Yung-Kyun Noh. K-beam minimax: Efficient optimization for deep adversarial learning. *arXiv preprint arXiv:1805.11640*, 2018.

[57] Md E. Haque, Yong hun Eom, Yuxiong He, Sameh Elnikety, Ricardo Bianchini, and Kathryn S. McKinley. Few-to-many: Incremental parallelism for reducing tail latency in interactive services. In *ASPLOS*, 2015.

[58] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 2010.

[59] Refael Hassin and Ariel Keinan. Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp. *Operations Research Letters*, 2008.

[60] Liang He, Eugene Kim, and Kang G. Shin. *aware charging of lithium-ion battery cells. In *ICCPS*, 2016.

[61] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing - a key technology towards 5g. *ETSI white paper*, 2015.

[62] Hyeong Soo Chang, P. J. Fard, S. I. Marcus, and M. Shayman. Multitime scale markov decision processes. *IEEE Trans. on Automatic Control*, 48(6):976–987, 2003.

[63] Million Insights. Edge computing market worth 43.4 billion by 2027. `https://www.grandviewresearch.com/press-release/global-edge-computing-market`, 2020.

[64] Intel. Intel cloud builders guide to power management in cloud design and deployment using Supermicro platforms and NMView management software. 2013.

[65] Internap. Colocation services and SLA, `http://www.internap.com/internap/wp-content/uploads/2014/06/Attachment-3-Colocation-Services-SLA.pdf`.

[66] M. A. Islam, Hasan Mahmud, S. Ren, and X. Wang. Paying to save: Reducing cost of colocation data center via rewards. In *HPCA*, 2015.

[67] M. A. Islam, Xiaoqi Ren, Shaolei Ren, Adam Wierman, and Xiaorui Wang. A market approach for handling power emergencies in multi-tenant data center. In *HPCA*, 2016.

[68] Mohammad A. Islam and Shaolei Ren. Ohm's law in data centers: A voltage side channel for timing power attacks. In *CCS*, 2018.

[69] Mohammad A. Islam, Shaolei Ren, and Adam Wierman. Exploiting a thermal side channel for power attacks in multi-tenant data centers. In *CCS*, 2017.

[70] Mohammad A. Islam, Xiaoqi Ren, Shaolei Ren, and Adam Wierman. A spot capacity market to increase power infrastructure utilization in multi-tenant data centers. In *HPCA*, 2018.

[71] Mohammad A. Islam, Luting Yang, Kiran Ranganath, and Shaolei Ren. Why some like it loud: Timing power attacks in multi-tenant data centers using an acoustic side channel. In *SIGMETRICS*, 2018.

[72] Haoming Jiang, Zhehui Chen, Yuyang Shi, Bo Dai, and Tuo Zhao. Learning to defense by learning to attack. *arXiv preprint arXiv:1811.01213*, 2018.

[73] Penny Jones. Overheating brings down microsoft data center. *Datacenter Dynamics*, 2013, `https://www.datacenterdynamics.com/news/overheating-brings-down-microsoft-data-center/`.

[74] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 1996.

[75] Keysight Technology. Learn to connect power supplies in parallel for higher current output, `https://www.keysight.com/main/editorial.jspx?cc=US&lc=eng&ckey=520808&nid=-11143.0.00&id=520808`.

[76] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[77] Youngjae Kim, Jeonghwan Choi, Sudhanva Gurumurthi, and Anand Sivasubramaniam. Managing thermal emergencies in disk-based storage systems. Dec 2008.

[78] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[79] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[80] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*. Citeseer, 2000.

[81] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M. Tullsen, and Tajana Simunic Rosing. Managing distributed UPS energy for effective power capping in data centers. In *ISCA*, 2012.

[82] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[83] Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Springer, 2012.

[84] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 2014.

[85] Alok Kumbhare, Reza Azimi, Ioannis Manousakis, Anand Bonde, Felipe Frujeri, Nithish Mahalingam, Pulkit Misra, Seyyed Ahmad Javadi, Bianca Schroeder, Marcus Fontoura, and Ricardo Bianchini. Prediction-based power oversubscription in cloud platforms. 2020.

[86] Kyuin Lee, Neil Klingensmith, Suman Banerjee, and Younghyun Kim. Voltkey: Continuous secret key generation based on power line noise for zero-involvement pairing and authentication. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 3(3), September 2019.

[87] Chao Li, Zhenhua Wang, Xiaofeng Hou, Haopeng Chen, Xiaoyao Liang, and Minyi Guo. Power attack defense: Securing battery-backed data centers. In *ISCA*, 2016.

[88] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[89] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[90] Fei Liang, Cong Shen, Wei Yu, and Feng Wu. Towards optimal power control via ensembling deep neural networks. *IEEE Transactions on Communications*, 2019.

[91] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[92] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[93] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[94] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, 2011.

[95] Paul Lin, Simon Zhang, and Jim VanGilder. Data center temperature rise during a cooling system outage. *White Paper of Schneider Electric Data Center Science Center*, 2013.

[96] Paul Lin, Simon Zhang, and Jim VanGilder. Data center temperature rise during a cooling system outage. *APC White Paper 179*, 2014.

[97] Longjun Liu, Chao Li, Hongbin Sun, Yang Hu, Juncheng Gu, Tao Li, Jingmin Xin, and Nanning Zheng. HEB: Deploying and managing hybrid energy buffers for improving datacenter efficiency and economy. In *ISCA*, 2015.

[98] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *ICDCS*, 2017.

[99] Sulav Malla, Qingyuan Deng, Zoh Ebrahimzadeh, Joe Gasperetti, Sajal Jain, Parimala Kondety, Thiara Ortiz, and Debra Vieira. Coordinated priority-aware charging of distributed batteries in oversubscribed data centers.

[100] Ioannis Manousakis, Íñigo Goiri, Sriram Sankar, Thu D. Nguyen, and Ricardo Bianchini. Coolprovision: Underprovisioning datacenter cooling. In *SoCC*, 2015.

[101] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *ISCA*, 2011.

[102] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 2015.

[103] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[104] David L. Moss. Dynamic control optimizes facility airflow delivery. *Dell White Paper*, March 2012.

[105] Angelia Nedić and Asuman Ozdaglar. Subgradient methods for saddle-point problems. *Journal of optimization theory and applications*, 142(1):205–228, 2009.

[106] John Niemann. Hot aisle vs. cold aisle containment. *American Power Conversion, West Kingston, RI, APC White Paper*, 2008.

[107] NVidia. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3080/.

[108] Parallel Workloads Archive, `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[109] Suhas V Patankar. Airflow and cooling in a data center. *Journal of Heat Transfer*, 132(7):073001, July 2010.

[110] Jan Peters and J Andrew Bagnell. Policy gradient methods. *Scholarpedia*, 2010.

[111] Ponemon Institute. 2016 cost of data center outages, 2016, `https://www.ponemon.org/blog/2016-cost-of-data-center-outages`.

[112] Peter Purdue. The m/m/1 queue in a markovian environment. *Operations Research*, 1974.

[113] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *SIGCOMM*, 2009.

[114] L. Ramos and R. Bianchini. C-Oracle: Predictive thermal management for data centers. In *HPCA*, 2008.

[115] Or Rivlin. Reinforcement learning for combinatorial optimization. `https://towardsdatascience.com/reinforcement-learning-for-combinatorial-optimization-d1402e396e91`, 2019. Accessed: 2021-01-28.

[116] Michael T Rosenstein, Andrew G Barto, Jennie Si, Andy Barto, and Warren Powell. Supervised actor-critic reinforcement learning. *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, 2004.

[117] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[118] Ernest K Ryu, Kun Yuan, and Wotao Yin. Ode analysis of stochastic gradient methods with optimism and anchoring for minimax problems and gans. *arXiv preprint arXiv:1905.10899*, 2019.

[119] Varun Sakalkar, Vasileios Kontorinis, David Landhuis, Shaohong Li, Darren De Ronde, Thomas Blooming, Anand Ramesh, James Kennedy, Christopher Malone, Jimmy Clidaras, and Parthasarathy Ranganathan. Data center power oversubscription with a medium voltage power plane and priority-aware capping. In *ASPLOS*, 2020.

[120] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[121] Zhihui Shao, Mohammad A Islam, and Shaolei Ren. A first look at thermal attacks in multi-tenant data centers. *ACM SIGMETRICS Performance Evaluation Review*, 2019.

[122] Zhihui Shao, Mohammad A Islam, and Shaolei Ren. Deeppm: Efficient power management in edge data centers using energy storage. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020.

[123] Zhihui Shao, Mohammad A Islam, and Shaolei Ren. Heat behind the meter: A hidden threat of thermal attacks in edge colocation data centers. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[124] Shashank Shekhar, Ajay Dev Chhokra, Anirban Bhattacharjee, Guillaume Aupy, and Aniruddha Gokhale. Indices: exploiting edge resources for performance-aware cloud-hosted services. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017.

[125] Qingjiang Shi, Meisam Razaviyayn, Zhi-Quan Luo, and Chen He. An iteratively weighted mmse approach to distributed sum-utility maximization for a mimo interfering broadcast channel. *IEEE Transactions on Signal Processing*, 2011.

[126] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[127] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[128] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 2016.

[129] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, 2014.

[130] Matt Skach, Manish Arora, Chang-Hong Hsu, Qi Li, Dean Tullsen, Lingjia Tang, and Jason Mars. Thermal time shifting: Leveraging phase change materials to reduce cooling costs in warehouse-scale computers. In *ISCA*, 2015.

[131] Matt Skach, Manish Arora, Dean Tullsen, Lingjia Tang, and Jason Mars. Virtual melting temperature: Managing server load to minimize cooling overhead with phase change materials. In *ISCA*, 2018.

[132] S. Skorobogatov. Local heating attacks on flash memory devices. In *Workshop on Hardware-Oriented Security and Trust*, 2009.

[133] D. R. Song, C. Yang, C. McGreavy, and Z. Li. Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge. In *ICARCV*, 2018.

[134] Robin A. Steinbrecher and Roger Schmidt. Data center environments: Ashrae's evolving thermal guidelines. *ASHRAE Technical Feature*, pages 42–49, December 2011.

[135] Chengjian Sun and Chenyang Yang. Learning to optimize with unsupervised learning: Training deep neural networks for urllc. In *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2019.

[136] Haoran Sun, Xiangyi Chen, Qingjiang Shi, Mingyi Hong, Xiao Fu, and Nikos D Sidiropoulos. Learning to optimize: Training deep neural networks for wireless resource management. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2017.

[137] Supermicro. Battery backup power - evolutionary design to replace UPS, `http://www.supermicro.com/products/nfo/files/bbp/f_bbp.pdf`.

[138] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[139] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[140] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[141] Yevgeniy Sverdlik. Google to build and lease data centers in big cloud expansion. In *DataCenterKnowledge*, April 2016.

[142] Qinghui Tang, Sandeep K. S. Gupta, and Georgios Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *CLUSTER*, 2007.

[143] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1458–1472, 2008.

[144] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 1994.

[145] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[146] Uber and Lyft. Uber and lyft dataset boston - from 11-26-2018 to 12-18-2018. `https://www.kaggle.com/brllrb/uber-and-lyft-dataset-boston-ma`, 2019.

[147] Uptime Institute. Data center industry survey, 2018, `https://uptimeinstitute.com/2018-data-center-industry-survey-results`.

[148] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely. Dynamic resource allocation and power management in virtualized data centers. In *NOMS*, 2010.

[149] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[150] Vapor IO. The edge data center, `https://www.vapor.io/`.

[151] Vertiv. Smartmod modula data center infrastructure.

[152] Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 2020.

[153] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 2015.

[154] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 2020.

[155] Di Wang, Sriram Govindan, Anand Sivasubramaniam, Aman Kansal, Jie Liu, and Badriddine Khessib. Underprovisioning backup power infrastructure for datacenters. In *ASPLOS*, 2014.

[156] Di Wang, Chuangang Ren, and Anand Sivasubramaniam. Virtualizing power distribution in datacenters. In *ISCA*, 2013.

[157] Di Wang, Chuangang Ren, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Hosam Fathy. Energy storage in datacenters: what, where, and how much? In *SIGMETRICS*, 2012.

[158] Guosai Wang, Shuhao Wang, Bing Luo, Weisong Shi, Yinghang Zhu, Wenjun Yang, Dianming Hu, Longbo Huang, Xin Jin, and Wei Xu. Increasing large-scale data center capacity by statistical power control. In *EuroSys*, 2016.

[159] Xiaodong Wang, Xiaorui Wang, Guoliang Xing, and Cheng xian Lin. Leveraging thermal dynamics in sensor placement for overheating server component detection. In *IEEE International Green Computing Conferenc*, 2012.

[160] Yuanhao Wang, Guodong Zhang, and Jimmy Ba. On solving minimax optimization locally: A follow-the-ridge approach. *arXiv preprint arXiv:1910.07512*, 2019.

[161] Zhenchao Wang, Haibin Duan, and Xiangyin Zhang. An improved greedy genetic algorithm for solving travelling salesman problem. In *2009 Fifth International Conference on Natural Computation*. IEEE, 2009.

[162] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[163] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[164] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, 2017.

[165] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Raymond Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's data center-wide power management system. In *ISCA*, 2016.

[166] Yuanhao Xiong and Cho-Jui Hsieh. Improved adversarial training via learned optimizer. In *European Conference on Computer Vision*, 2020.

[167] Jie Xu, Lixing Chen, and Shaolei Ren. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. *IEEE Trans. on Cognitive Communications and Networking*, 3(3):361–373, September 2017.

[168] Jie Xu, Lixing Chen, and Shaolei Ren. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. *IEEE Transactions on Cognitive Communications and Networking*, 3(3):361–373, 2017.

[169] Zhang Xu, Haining Wang, Zichen Xu, and Xiaorui Wang. Power attack: An increasing threat to data centers. In *Network and Distributed System Security Symposium*, 2014.

[170] Gongjun Yan, Ding Wen, Stephan Olariu, and Michele C Weigle. Security challenges in vehicular cloud computing. *IEEE Transactions on Intelligent Transportation Systems*, 2012.

[171] L. Yang, X. Chen, J. Zhang, and H. V. Poor. Optimal privacy-preserving energy management for smart meters. In *INFOCOM*, 2014.

[172] D. Yi, X. Zhou, Y. Wen, and R. Tan. Efficient compute-intensive job allocation in data centers via deep reinforcement learning. *IEEE Trans. on Parallel and Distributed Syst.*, 31(6):1474–1485, 2020.

[173] Shui Yu, Yonghong Tian, Song Guo, and Dapeng Oliver Wu. Can we beat ddos attacks in clouds? *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2245–2254, September 2014.

[174] Linquan Zhang, Shaolei Ren, Chuan Wu, and Zongpeng Li. A truthful incentive mechanism for emergency demand response in colocation data centers. In *INFOCOM*, 2015.

[175] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *CCS*, 2012.

[176] Wenli Zheng, Kai Ma, and Xiaorui Wang. Exploiting thermal energy storage to reduce data center capital and operating expenses. In *HPCA*, 2014.