UNIVERSITY OF CALIFORNIA, MERCED

Cyber-Physical Systems Optimization with Reinforcement Learning: Methods and Applications

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering and Computer Science

by

Xianzhong Ding

October 2023

The dissertation of Xianzhong Ding is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Professor Stefano Carpin

Professor Alberto Cerpa

Professor Wan Du, Chair

University of California, Merced

October 2023

DEDICATION

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Prof. Wan Du and Prof. Alberto Cerpa for their guidance during my graduate studies. Without their support, my progress in this endeavor would not have been possible. In addition, I would like to thank my committee members Prof. Stefano Carpin, Prof. Wan Du, and Prof. Alberto Cerpa for their insight across my various stages of research. Next, I must thank my many past and current colleagues, including Miaomiao Liu, Weifeng Gao, Haiqiao Wu, Nasit Sony, Jothi Prasanna Shanmuga Sundaram, Kang Yang, Yuning Chen, Sikai Yang, Zhiyu An, and the undergraduates who have helped along the way.

VITA

2010 - 2014	B. S. in Computer Science and Technology, Taishan University
2015 - 2018	M. S. in Computer Science and Technology, Shandong University
2018 - Now	Ph. D. in Electrical Engineering and Computer Science, University of California, Merced

PUBLICATIONS

Xianzhong Ding et al., “Reinforcement Learning for Virtual Machines Rescheduling in Cloud Data Centers”, 2023, Workshop on ML for Systems at NeurIPS 2023, December 16, New Orleans.

Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto E. Cerpa, and Wan Du, “HPC-GPT: Integrating Large Language Model for High-Performance Computing”, In Workshops of The International Conference on High Performance, Computing, Network, Storage, and Analysis (SC-W 2023).

Le Chen, **Xianzhong Ding**, Murali Emani, Tristan Vanderbruggen, Pei-hung Lin, and Chuanhua Liao, “Data Race Detection Using Large Language Models”, In Workshops of The International Conference on High Performance, Computing, Network, Storage, and Analysis (SC-W 2023).

Hamid Rajabi, **Xianzhong Ding**, Wan Du, Alberto Cerpa, “TODOS: Thermal sensOr Data-driven Occupancy Estimation System for Smart Buildings”, ACM BuildSys, 2023.

Zhiyu An, **Xianzhong Ding**, Arya Rathee, Wan Du, “CLUE: Safe Model-Based RL HVAC Control Using Epistemic Uncertainty Estimation”, ACM BuildSys, 2023. **Best Paper Runner-Up Award**

Hamid Rajabi, Zhizhang Hu, **Xianzhong Ding**, Shijia Pan, Wan Du, Alberto Cerpa, “MODES: Multi-sensor Occupancy Data-driven Estimation System for Smart Buildings”, ACM International Conference on Future Energy Systems, 2022.

Xianzhong Ding and Wan Du, “DRLIC : Deep Reinforcement Learning for Irrigation Control”, *The International Conference on Information Processing in Sensor Networks (IPSN)*, Italy, May 2022.

Xianzhong Ding and Wan Du, “Smart Irrigation Control Using Deep Reinforcement Learning”, *The International Conference on Information Processing in Sensor Networks (IPSN Poster)*, Italy, May 2022.

Devanshu Kumar, **Xianzhong Ding**, Wan Du and Alberto Cerpa, "Building Sensor Fault Detection and Diagnostic System, *ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys)*, Coimbra, Portugal, November 2021.

Xianzhong Ding, Wan Du, and Alberto Cerpa, "MB^{2}C : Model-Based deep reinforcement learning for Multi-zone Building Control", *ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys)*, Japan, November 2020. **Best Paper Runner-Up Award, Best Presentation Award**.

Miaomiao Liu, **Xianzhong Ding**, Wan Du, "Continuous, Real-Time Object Detection on Mobile Devices without Offloading", *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Singapore, December 2020.

Xianzhong Ding, Wan Du, and Alberto Cerpa, "OCTOPUS : Deep Reinforcement Learning for Holistic Smart Building Control", *ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys)*, New York, November 2019.

ABSTRACT OF THE DISSERTATION

Cyber-Physical Systems Optimization with Reinforcement Learning: Methods and Applications

by

Xianzhong Ding
Doctor of Philosophy in Electrical Engineering and Computer Science
University of California Merced, October 2023
Professor Wan Du, UC Merced, Chair

Cyber-Physical Systems rely on many control and decision-making algorithms. However, the majority of cyber-physical systems today are still operated by simple rule-based and feedback controls such as on-off control or proportional-integral derivative (PID) control. These prescriptive and reactive control strategies do not take into consideration predictive information on disturbances, such as weather changes [4], making their performance sub-optimal. Optimal control strategies such as MPC address these drawbacks by iteratively optimizing an objective function over a receding time horizon. However, despite many successful cyber-physical systems applications of MPC (e.g., [5, 6]), its wide-spread adoption has been limited by the need of accurate models [4, 7]. This is especially challenging because cyber-physical systems are heterogeneous [8]. Thus, custom models are required for cyber-physical systems, limiting the scalability of MPC.

To overcome the above challenges, we investigate deep reinforcement learning (DRL) for solving challenging cyber-physical system optimization problems. Deep reinforcement learning is a data-driven control method and learns an optimal control policy directly from data, without using any pre-programmed control rules or explicit assumptions about the cyber-physical systems. DRL also allows us to use domain knowledge to train a control agent (a neural network) without labeled data. The generalization ability of the neural network enables the control agent to better handle the dynamically-varying factors (e.g., weather changing).

We use this approach to build a series of cyber-physical systems for important applications including building control, irrigation control, and virtual machine rescheduling. In the first building control application, we take a holistic approach to deal with the trade-offs between energy use and comfort in commercial buildings. Further, we investigate the common data-efficiency problem when leveraging DRL to cyber-physical systems and propose two data-efficient model-based RL methods for multi-zone building control. In the second irrigation control application, we design a DRL-based irrigation system that generates optimal irrigation control commands according to current soil water content, current weather data, and forecasted weather information. In the third virtual machine rescheduling application, we design a two-stage DRL agent framework to balance rescheduling performance and solution speed. In future work, we will use the insights from these systems to identify common problems and develop new data-efficient reinforcement learning techniques for cyber-physical systems.

Chapter 1

Introduction

Cyber-physical systems (CPS) represent a category of embedded systems tightly integrated with computer algorithms, computation, and communication. In today's fast-evolving digital landscape, CPS systems are gaining widespread prevalence, driven by the rapid growth of the internet and information technology. These systems find application in various domains, including building energy management, irrigation control, and data-center infrastructure. Within these diverse CPS applications, the primary focus often centers on optimizing energy and water efficiency and refining scheduling algorithms. However, as the complexity of physical dynamics within buildings, irrigation systems, and data centers continues to increase, the design of robust management systems for energy, water, and virtual machine scheduling, as well as the development of highly efficient control and scheduling algorithms, has become increasingly critical and challenging. In this dissertation, we introduce deep reinforcement learning methodologies aimed at enhancing the energy and water efficiency of smart buildings and irrigation systems, along with improving scheduling performance in data centers, to address these evolving demands of CPS applications in our dynamic technological landscape.

In Chapter 2, we introduce a holistic approach to deal with the trade-offs between energy use and comfort in commercial buildings. We developed a system called OCTOPUS, which employs a novel deep reinforcement learning (DRL) framework that uses a data-driven approach to find the optimal control sequences of *all* building's subsystems, including HVAC, lighting, blind and window systems. The DRL architecture includes a novel reward function that allows the framework to explore the trade-offs between energy use and users' comfort, while at the same time enabling the solution of the high-dimensional control problem due to the interactions of four different building subsystems. In order to cope with OCTOPUS's data training requirements, we argue that calibrated simulations that match the target building operational points are the vehicle to generate enough data to be able to train our DRL framework to find the control solution for the target building. In our work, we trained OCTOPUS with 10-year weather data and a building model that is implemented in the EnergyPlus building simulator, which was calibrated using data from a real production building.

In Chapter 3, we conduct a set of experiments to analyze the limitations of current MBRL-based HVAC control methods, in terms of model uncertainty and controller effectiveness. Using the lessons learned, we develop MB^{2}C, a novel MBRL-based HVAC control system that can achieve high control performance with excellent sample efficiency. MB^{2}C learns the building dynamics by employing an ensemble of environment-conditioned neural networks. It then applies a new control method, Model Predictive Path Integral (MPPI), for HVAC control. It produces candidate action sequences by using an importance sampling weighted algorithm that scales better to high state and action dimensions of multi-zone buildings. We evaluate MB^{2}C using EnergyPlus simulations in a five-zone office building.

In Chapter 4, we introduce a deep reinforcement learning (DRL)-based irrigation system. *DRLIC* uses a neural network (DRL control agent) to learn an optimal control policy that takes both current soil moisture measurement and future soil moisture loss into account. We define an irrigation reward function that facilitates the control agent to learn from past experience. Sometimes, our DRL control agent may output an unsafe action (e.g., irrigating too much water or too little). To prevent any possible damage to plants' health, we adopt a safe mechanism that leverages a soil moisture predictor to estimate each action's performance. If it is unsafe, we will perform a relatively-conservative action instead. Finally, we develop a real-world irrigation system that is composed of sprinklers, sensing and control nodes, and a wireless network.

In Chapter 5, we introduce a virtual machines rescheduling algorithm using deep reinforcement learning in data centers. Modern industry-scale data centers receive thousands of virtual machine (VM) requests per minute. Due to the continual creation

and release of VMs, many small resource fragments are scattered across physical machines (PMs). To handle these fragments, data centers periodically reschedule some VMs to alternative PMs. Despite the increasing importance of VM rescheduling as data centers grow in size, the problem remains understudied. We first show that, unlike most combinatorial optimization tasks, the inference time of VM rescheduling algorithms significantly influences their performance, causing many existing methods to scale poorly. Therefore, we develop a deep reinforcement learning system for VM rescheduling, VMR^{2}L, which incorporates a set of customized techniques, such as a two-stage framework that accommodates diverse constraints and workload conditions, as well as an effective feature extraction module. Our experiments on an industry-scale data center show that VMR^{2}L can achieve a performance comparable to the optimal solution, but with a running time of seconds.

Chapter 2

Deep Reinforcement Learning for Holistic Smart Building Control

Recently, significant efforts have been done to improve quality of comfort for commercial buildings' users while also trying to reduce energy use and costs. Most of these efforts have concentrated in energy efficient control of the HVAC (Heating, Ventilation, and Air conditioning) system, which is usually the core system in charge of controlling buildings' conditioning and ventilation. However, in practice, HVAC systems alone cannot control every aspect of conditioning and comfort that affects buildings' occupants. Modern lighting, blind and window systems, usually considered as independent systems, when present, can significantly affect building energy use, and perhaps more importantly, user comfort in terms of thermal, air quality and illumination conditions. For example, it has been shown that a blind system can provide 12%∼35% reduction in cooling load in summer while also improving visual comfort.

In this paper, we take a holistic approach to deal with the trade-offs between energy use and comfort in commercial buildings. We developed a system called OCTOPUS, which employs a novel deep reinforcement learning (DRL) framework that uses a data-driven approach to find the optimal control sequences of *all* building's subsystems, including HVAC, lighting, blind and window systems. The DRL architecture includes a novel reward function that allows the framework to explore the trade-offs between energy use and users' comfort, while at the same time enable the solution of the high-dimensional control problem due to the interactions of four different building subsystems. In order to cope with OCTOPUS's data training requirements, we argue that calibrated simulations that match the target building operational points are the vehicle to generate enough data to be able to train our DRL framework to find the control solution for the target building. In our work, we trained OCTOPUS with 10-year weather data and a building model that is implemented in the EnergyPlus building simulator, which was calibrated using data from a real production building. Through extensive simulations we demonstrate that OCTOPUS can achieve 14.26% and 8.1% energy savings compared with the state-of-the art rule-based method in a LEED Gold Certified building and the latest DRL-based method available in the literature respectively, while maintaining human comfort within a desired range.

2.1 Introduction

Energy saving in buildings is important to society, as buildings consume 32% energy and 51% electricity demand worldwide [9]. Rule-based control (RBC) is widely used to set the actuators (e.g., heating or cooling temperature, and fan speed) in the HVAC (heating, ventilation, and air-conditioning) system. The "rules" in RBC are usually set as some static thresholds or simple control loops based on the experience of engineers and facility managers. The thresholds and simple control rules may not be optimal and have to be adapted to new buildings at commissioning time. Many times these rules are updated in an ad-hoc manner, based on experience and feedback from occupants and/or trial and error performed by HVAC engineers during the operational use of the building. As a result, many model-based approaches have been developed to model the thermal dynamics of a building and execute a control algorithm on top of the model, such as Proportional Integral Derivative (PID) [10] and Model Predictive Control (MPC) [11]. However, the complexity of the thermal dynamics and the various influencing factors are hard to be precisely modeled, which is why the models tend to be simplified in order deal with the parameter-fitting data requirements and computational complexity when solving the optimization problem [11].

To tackle the limitations of the model-based methods, some model-free approaches have been proposed based on reinforcement learning (RL) for HVAC control, including Q-learning [12] and Deep Reinforcement Learning (DRL) [13]. With RL, an optimal control policy can be learned by the trial-and-error interaction between a control agent and a building, without explicitly modeling the system dynamics. By adopting a deep neural network as the control agent, DRL-based schemes can handle large state and action space in building control [14]. Some recent work [15, 13] has shown that DRL can provide real-time control for building energy efficiency. However, all existing methods only consider a single subsystem in buildings, e.g., the HVAC system [15] or the heating system [13], ignoring some other subsystems that can affect performance from the energy use and/or user comfort point of view.

At present, more and more buildings are been equipped with automatically-adjustable windows and blinds. For example, motor-operated windows and blinds, like the intelligent products from GEZE, have been installed using an effective natural ventilation strategy. In addition, researchers have studied the potential of energy saving by jointly controlling the HVAC system and another subsystem, like blind [16], lighting [17], and window [18]. For example, the energy consumed by HVAC can be reduced by 17%∼47% if window-based natural ventilation is enabled [18].

In this work, we argue that a *holistic approach* that considers *all available subsystems* (HVAC, blinds, windows, lights) in buildings, which have complex and non-trivial interactions should be used in coordination to achieve a specific energy efficiency/comfort goal. Figure 2.5 shows a depiction of a modern building that includes multiple subsystems (e.g., HVAC, window, blind and lighting) that work together to guarantee human comfort goals, including thermal comfort, visual comfort, and indoor air quality goals. For example, indoor temperature can be influenced by three subsystems, like setting the HVAC temperature (adjusting the discharge temperature set points at the VAV level), and/or adjusting blind slats (allowing external sunlight to heat indoor air) and/or the window system (enabling exchange of indoor and outdoor air).

To achieve more efficient energy management in buildings, we propose to study the joint control problem of four subsystems of a building to meet three human comfort metrics as depicted in Figure 2.2. The energy consumption of a building is determined by four subsystems and their interaction. It is challenging to control four subsystems jointly, since they may have opposite outcomes on different human comfort metrics. For example, opening the window can improve indoor air quality and save the energy consumed by the HVAC system for ventilation, but it may also reduce (in winter) or increase (in summer) indoor temperature. To handle the temperature

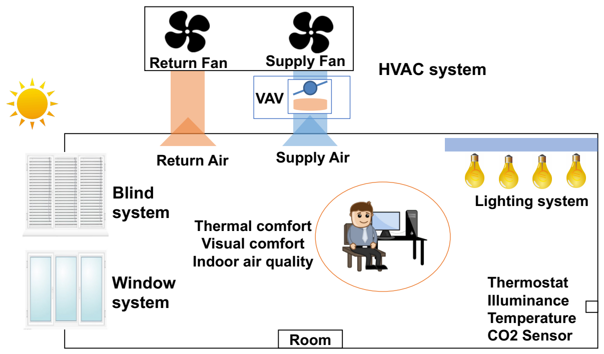


Figure 2.1: Four Subsystems in a Typical Building.

variation caused by the open window, the HVAC system may need to spend more energy rather than the energy saved by natural ventilation.

This paper presents a customized DRL-based control system, named OCTOPUS, which controls four subsystems of a building to meet three human comfort requirements with the best energy efficiency. It leverages all the advantages of DRL-based control, including fast adaptation to new buildings, real-time actuation and being able to handle a large state space. However, to control four subsystems jointly in a unified framework, we need to tackle three main challenges:

High-Dimension Control Actions. With a uniform DRL framework, OCTOPUS needs to decide a control action for four subsystems jointly and periodically, including the heating/cooling air temperature of the HVAC system, the brightness level of electric lights, the blind slat range and the open proportion of the window. Each subsystem adds one dimension in the action space. The goal of OCTOPUS is to select the best action combination A_s from the set of all possible combinations A_{all} that meet the requirement of human comfort with the lowest energy consumption. Since each subsystem can set its actuator to a large number of discrete values, e.g., we have 66 possible values to set the zone temperature by the HVAC system, the set of all possible action combinations A_{all} is extremely large, i.e., 2,371,842 actions in our case. To solve this problem, we leverage a novel neural architecture featuring a shared representation followed by four network branches, one for each action dimension. In addition, from the shared representation, a state value is obtained that links the joint interrelations in the action space, and it is added to the output of the four previous branches. This approach achieves a linear increase in the number of network outputs by allowing independence for each action dimension.

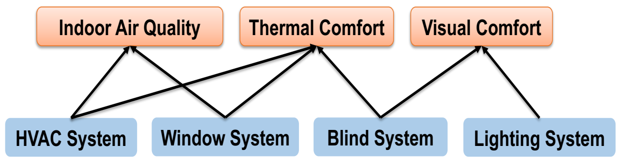


Figure 2.2: Relationship between Four Subsystems and Three Human Comfort Metrics

Reward Function. To explore the potential energy saving energy across four subsystems while considering three human comforts, we formulate this problem into an optimization problem. We define a reward function in our DRL framework to solve the optimization problem. The novel reward function jointly combines energy consumption, thermal comfort, visual comfort, and indoor air quality, offering better control and more flexibility to meet the unique requirement of specific users.

Data Training Requirements. While model-free approaches in general, and RL techniques in particular, are very powerful, their main weakness is the amount of data [19] required to train them properly. The amount of training data should be in proportion to the action space, which in our case it is very large. This issue is very important since we cannot expect building stakeholders to have years of building data readily available so we can use OCTOPUS. Instead, we use a calibrated building simulator combined with weather data that is readily available, in order to generate as much training data as we needed. We trained our OCTOPUS system with 10-year of weather data of two areas; one is Merced, CA, and the other one in Chicago, IL, due to their distinct weather characteristics. The critical point is that this method allows to train OCTOPUS for any building under any weather profile, as long as there is a repository of weather data for the location, and a few months of building data to perform the calibration of the simulator.

We would like to highlight the main contributions of the paper:

- To the best of our knowledge, this is the first work that leverages DRL to balance the tradeoff between energy use and human comfort in a holistic manner.
- OCTOPUS adopts a special reward function and a new DRL architecture to tackle the challenges imposed by the combined joint control of four subsystems with a very large action space.
- We tackle the issue of data training requirement by adopting a simulation strategy for data generation, and spending effort in calibrating the simulations to make them

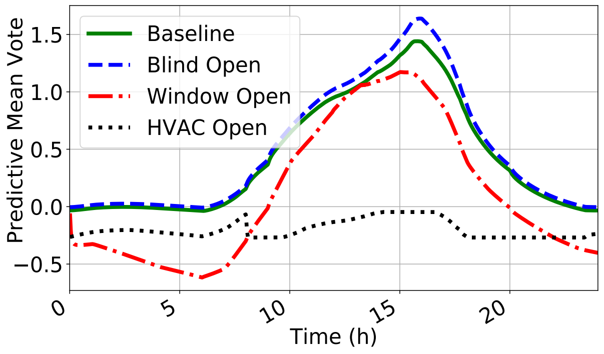


Figure 2.3: Thermal Comfort, PMV.

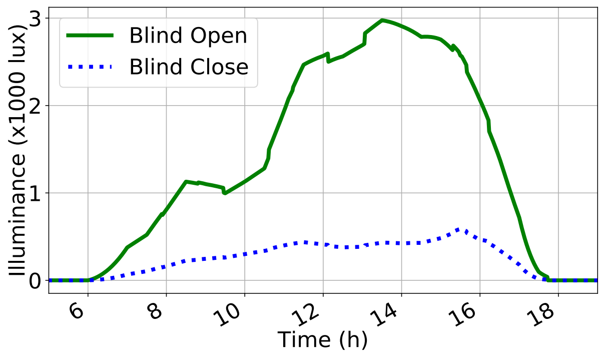


Figure 2.4: Visual Comfort, Illuminance.

as close as possible to the target building. This allows our system to generate as much data as needed within a finite amount of time.

2.2 Related work

Conventional control of the HVAC system. Model predictive control (MPC) models have been developed for HVAC control. For example, complex models are commonly used to model building temperature response [11]. However, MPC control only works well for low-order system dynamics, and its control variables must be carefully set for different buildings [20].

Conventional control of multiple subsystems. Kolokotsa et al. [21] develop an energy efficient fuzzy controller based on a genetic algorithm to control four subsystems (HVAC, lighting, window, and blind) and meet the occupant requirements of human comfort. However, the genetic algorithm requires a few minutes to hours to generate one control action. It is not practical to be used in real building control.

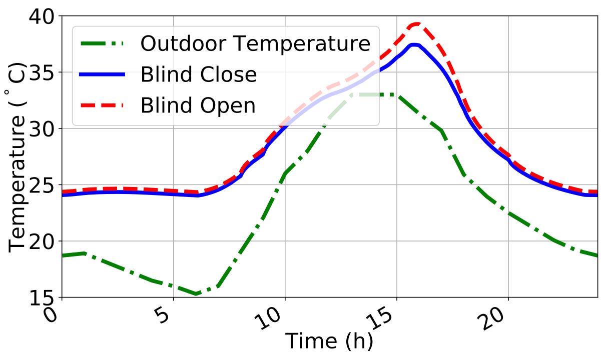


Figure 2.5: Temperature Effect.

RL-based control of the HVAC system. Li et al. [12] adopt Q learning for HVAC control. Dalamagkidis et al. [22] design a Linear Reinforcement Learning Controller (LRLC) using linear function approximation of the state-action value function to meet the thermal comfort with minimal energy consumption. However, the tabular Q learning approaches are not suitable for problems with a large state space, like the state of four subsystems.

DRL-based control of the HVAC system. Wei et al. [15] develop a data-driven deep reinforcement learning approach to intelligently learn an effective strategy for HVAC control. Zhang et al. [13, 23] implement and deploy a DRL-based control method for radiant heating systems in a real-life office building. Although the above works can improve the performance of HVAC control, they require discretization of the state-action space and are only focused on HVAC subsystem.

2.3 Design of Octopus

In this section, we describe in detail the design of OCTOPUS, including a system overview, DRL-based building control, branching dueling Q-Network, and reward function calculation.

2.3.1 OCTOPUS Overview

The design goal of OCTOPUS is to meet the requirement of human comfort by energy efficient control of four subsystems in a building [24]. Our goal is to minimize the energy E consumed by all subsystems in the building, including the energy used in

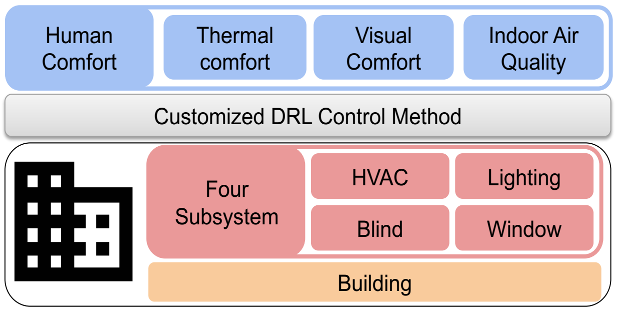


Figure 2.6: OCTOPUS architecture

heating/cooling coils to heat and cool the air, the electricity used in the water pumps and flow fans in the HVAC system, electricity used by the lights, and the electricity used by the motors to adjust the blinds and windows. The value of E is constantly being affected by the vector A_s, which is an action combination for four subsystems, which belongs to the vector A_{all} that is all the possible action combinations.

In addition to the minimization of energy, we would like to maintain the human comfort metrics within a particular range. This can be expressed as $P_{min} \leq PMV \leq P_{max}, V_{min} \leq V \leq V_{max}$, and $I_{min} \leq I \leq I_{max}$. PMV is a parameter that measures thermal comfort; V measures visual comfort; and I measures indoor air quality. The consumed energy E and the human comfort metrics (PMV, V, and I) are determined by the current state of all four subsystems, the outdoor weather and the action we are about to take. They can be measured in real buildings or calculated in a building simulator, like EnergyPlus, after the action is executed.

The achieved human comfort results should fall into an acceptable range to meet the requirements of users. We use $[P_{min},\ P_{max}]$, $[V_{min},\ V_{max}]$, $[I_{min},\ I_{max}]$ to present the accepted range for thermal comfort, visual comfort and indoor air quality. They can be set by individual users according to their preference, or by facility managers based on building standards. The details on calculation of the above parameters (E, PMV, V and I), the definition of an action (A_s) and the settings of the human comfort ranges (e.g., $[P_{min},\ P_{max}]$) will be introduced in Section 2.3.4.

Our goal is to find the best A_s from A_{all} for each action interval (15 mins in our implementation). The best A_s should maintain the three human comfort metrics in their acceptable ranges for the entire control interval with the lowest energy consumption (E). To achieve this goal, we implement a DRL-based control system for buildings. Figure 2.6 shows the overview of OCTOPUS as a building control system.

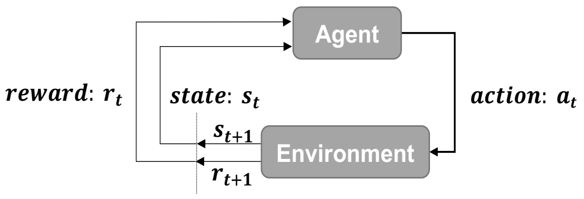


Figure 2.7: Deep Reinforcement Learning Model.

It consists of three layers, i.e., building layer, control layer, and user demand layer. The building layer is composed of the real building or a building simulation model, and the sensor data management components. It provides sensor data to the control layer and executes the control actions generated by the latter. The user demand layer quantifies the user requirement of three human comfort metrics. The range of each human comfort metric is then passed to the control layer, which searches for the optimal control to meet the human comfort ranges with minimal energy consumption.

2.3.2 DRL-based Building Control

Basics for DRL and DQN

In a standard RL framework, as shown in Figure 2.7, an agent learns an optimal control policy by trying different control actions for the environment. In our case, the environment is a building simulation model due to the extensive data requirements to train the system. With DRL, the agent is implemented as a deep neural network (DNN). The agent-environment interactions of one step can be expressed as a tuple (S_t, A_t, S_{t+1}, R_{t+1}), where S_t is the environment's state at time t, A_t is the control action performed by the agent at time t, S_{t+1} is the resulting environment's state after the agent has taken the action, R_{t+1} is the reward received by the agent from the environment. The goal of DNN agent training is to learn an optimal control policy to maximize the accumulated returned reward by taking different control actions.

State in OCTOPUS

The state is what the DRL agent takes as input for each control step. In this study, the state is a stack of the current and historical observations, as shown below:

$$S = \{ob_t, ob_{t-1}, ..., ob_{t-n}\}\,, \tag{2.1}$$

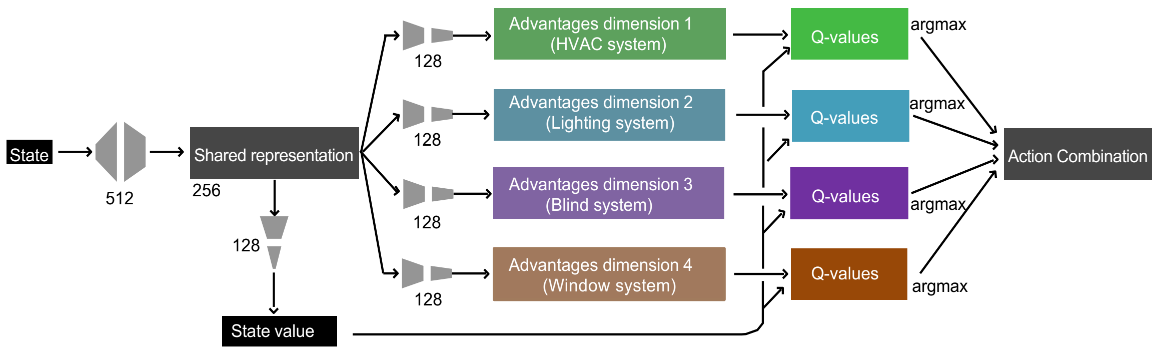

State
512
Shared representation
256
128
State value
128
Advantages dimension 1 (HVAC system)
128
Advantages dimension 2 (Lighting system)
128
Advantages dimension 3 (Blind system)
128
Advantages dimension 4 (Window system)
Q-values
argmax
Action Combination

Figure 2.8: The Specific Action Branching Network Implemented for the Proposed BDQ Agent

where t is the current time step, n is the number of the historical time steps to be considered, and each ob consists of the following 15 items: outdoor air temperature (°C), outdoor air relative humidity (%), indoor air temperature(°C), indoor air relative humidity (%), diffuse solar radiation (W/m^2), direct solar radiation (W/m^2), solar incident angle (°), wind speed (m/s), wind direction (degree from north), average PMV (%), heating setpoint of the HVAC system (°C), cooling setpoint of the HVAC system (°C), the dimming level of lights (%), the window open percentage (%), and the blind open angle (°). All the values we can be calculated by the EnergyPlus simulation model. Min-max normalization is used to convert each item to a value within 0-1.

Action in OCTOPUS

The action is how the DRL agent controls the environment. Given the state, we want the agent to find the most suitable action combinations among HVAC, lighting, blind and window system to balance energy consumption and three human comfort metrics. There are four action dimensions when considering these four subsystems, represented as

$$A_t = \{H_t, L_t, B_t, W_t\}\,, \tag{2.2}$$

where A_t is the action combination of four subsystems at time t. H_t is the temperature set-point of the HVAC system, which can be set to 66 values. L_t is the dimming level of electric lights. B_t is the blind slat angle. The range of blind slat can be adjusted from $0\ ° \sim 180\ °$. W_t is the open percentage of the window. Each of the above three actuation parameters can be set to 33 values in our current implementation to achieve a proper balance between control granularity and calculation complexity. According to Equation 2.2, the total number of possible actions in the action space is 2,371,842 ($66 \times 33 \times 33 \times 33$). Existing DRL architectures, like Deep Q-Network (DQN) in [15] and Asynchronous Advantage Actor-Critic (A3C) in [13], cannot work efficiently in our problem, because the large number of actions requires to be explicitly represented in the agent DNN network and it will significantly increase the number of DNN parameters to be learned and consequently the training time [25]. To solve this problem, we leverage a novel neural architecture featuring a shared representation followed by four network branches, one for each action dimension.

Reward Function in OCTOPUS

Reward illustrates the immediate evaluation of the control effects for each action under a certain state. Both human comfort and energy consumption should be incorporated. To define the reward function, a common approach is to use the Lagrangian Multiplier function [26] to first convert the constrained formulation into an unconstrained one:

$$\begin{aligned} R = -[\rho_1 Norm(E) + \rho_2 Norm(T_c) \\ +\rho_3 Norm(V_c) + \rho_4 Norm(I_c)], \end{aligned} \tag{2.3}$$

where ρ_1, ρ_2, ρ_3 and ρ_4 are the Lagrangian multipliers. E is energy consumption, Tc is thermal comfort, Vc is visual comfort and Ic is Indoor air quality. $Norm(x)$ is a normalization process, i.e., $Norm(x)$ = (x - x_{min})/(x_{max} - x_{min}) to transform energy and three human comfort to the same scale. This reward function merges the objective (e.g. energy consumption) and constraint satisfaction (e.g. human comfort). The reward consists of four parts, namely, the penalty for the energy consumption of the HVAC and lighting system, the penalty for the occupants' thermal discomfort, the penalty for the occupants' visual discomfort and the penalty for the occupants' indoor air condition discomfort. Specifically, the reward should be less, if more energy is consumed by the HVAC system or the occupants feel uncomfortable about the building thermal, visual and indoor air condition. The details about how to define and formulate energy consumption E, thermal comfort Tc, visual comfort Vc and indoor air condition Ic are explained in Section 2.3.4.

2.3.3 Branching Dueling Q-Network

To solve the high-dimensional action problem described in Section 4.3.3, OCTOPUS adopts a Branching Dueling Q-Network (BDQ), which is a branching variant of the dueling Double Deep Q-Network (DDQN). BDQ is a new neural architecture featuring a shared decision module followed by several network branches, one for each action dimension. BDQ can scale robustly to environments with high dimensional action spaces and even outperform the Deep Deterministic Policy Gradient (DDPG) algorithm in the most challenging task [27]. In our current implementation, we use a simulated building model developed in EnergyPlus as the environment for training and validation. Our BDQ-based agent interacts with the EnergyPlus model. At each control step, it processes the state (building and weather parameters) and generates a combined action set for four subsystems.

Figure 2.8 demonstrates the action branching network of BDQ agent. When a state is inputted, the shared decision module computes a latent representation that is then used for the calculation of the state value and the output of the network (Advantages dimension in Figure 2.8) for each dimension branch. The state value and the factorized advantages are then combined, via a special aggregation layer, to output the Q-values for each action dimension. These Q-values are then queried for the generation of a joint-action tuple. The weights of the fully connected neural layers are denoted by the gray trapezoids and the size of each layer (i.e. number of units) is depicted in the figure.

Training Process: The training process of the BDQ-based control agent is outlined in Algorithm 1. At the beginning, we first initialize a neural network Q with random weight θ. Another neural network Q^{-} with the same architecture is also created. The outer "for" loop controls the number of training episodes, and the inner "for" loop performs control at each control time step within one training episode. During the training process, the recent transition tuples (S_t, A_t, S_{t+1}, R_{t+1}) are stored in the replay memory Λ from which a mini-batch of samples will be generated for neural network training. The variable A_t stores the control action in the last step, and S_t and S_{t+1} represent the building state in the previous and current control time steps, respectively. At the beginning of each time slot t, we first update four actions and obtain the current state S_{t+1}. In line 7, the immediate reward R_{t+1} is

Algorithm 1 The Training Process of Our BDQ-Based Agent

```
Input: The range of human comfort metrics and maximum acceptable energy consumption
Output: A trained DRL agent
Initialize BDQ's prediction Q with random weights $\theta$
Initialize BDQ's target $Q^{-}$ with weight $\theta^{-} = \theta$
for episode =0,1,...,M do
    Obtain the initial state $S_t$ and $A_t$ randomly
    for control time step $t$ = 0,1,...,T do
        Update $H_t, L_t, B_t, W_t$ by the control action, $A_t$
        Calculate reward $R_{t+1}$ by Equation 2.3
        Obtain current state observation $S_{t+1}$
        Store ($S_t$, $A_t$, $S_{t+1}$, $R_{t+1}$) in reply memory $\Lambda$
        Draw mini-batch sample transitions from $\Lambda$
        Calculate the target vector and update weights in neural network Q
        Update target network $Q_d^{-}(s, a_d)$ using Equation 2.5
        Perform greedy descent iteratively to tune BDQ by Equation 2.6.
    end
end
```

calculated by Equation 2.3. A training mini-batch can be built by randomly drawing some transition tuples from the memory.

We calculate the target vector and update the weights of the neural network Q by using an Adam optimizer for every control step t. Formally, for an action dimension $d \in 1, ...N$ with n discrete actions, a branch's Q-value at state $s \in S$ and with action $a_d \in A_d$ is expressed in terms of the common state value $V(s)$ (the result of the shared representation layer in Figure 2.8) and the corresponding (state-dependent) action advantage $A_d(s, a_d)$ of each branch (the result of the each advantage dimension in Figure 2.8) by:

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d) - \tfrac{1}{n} \textstyle\sum_{a_d^{'} \in A_d} A_d(s, a_d^{'})). \tag{2.4}$$

The target network Q^{-} will be updated with the latest weights of the network Q every c control time steps. c is set to 50 in our current implementation. Q^{-} is used for inferring the target value for the next c control steps. We use y_d to represent the maximum accumulative reward we can obtain in the next c steps. y_d can be calculated by temporal-difference (TD) targets in a recursive fashion:

$$y_d = R + \gamma \tfrac{1}{N} \textstyle\sum_d Q_d^{-}(s^{'}, \arg\max_{a_d^{'} \subseteq A_d} Q_d(s^{'}, a_d^{'})), \tag{2.5}$$

where Q_d^- denoting the branch d of the target network Q^-; R is the reward function result; and γ is discount factor.

Finally, at the end of the inner "for" loop, we calculate the following loss function every c control steps:

$$L = \mathbb{E}_{(s,a,r,s\prime)} \sim D \left[\textstyle\sum_d (y_d - Q_d(s, a_d))^2 \right], \tag{2.6}$$

where D denotes a (prioritized) experience replay buffer and a denotes the joint-action tuple $(a_1, a_2, ..., a_N)$. The loss function L should decrease as more training episodes are performed.

2.3.4 Reward Calculation

This section describes how we calculate the reward function in Equation 2.3, including energy cost E, thermal comfort T, visual comfort V and indoor air condition I.

Energy Consumption

The energy consumption of a building includes heating coil power P_h and cooling coil power P_c and fan power P_f from the HVAC system and electric light power P_l from the lighting system. We calculate the reward function for energy consumption E during a time slot as

$$E = (P_h + P_c + P_f + P_l) \tag{2.7}$$

The heating and cooling coil are used to cool or heat the air and the fan is used to distribute the heating air or cooling air to the zone. The electric lights are used for normal work in the zone. They are calculated by EnergyPlus simulator in our training and evaluation. In our current implementation, we ignore the power consumed by the water pumps and the motors to adjust blinds and windows, because it is relatively small compared with the power consumption of the HAVC system or the lighting systems, and can be safely ignored (less than 1% total).

Human Comfort.

We define and explain the measurement of the three human comfort metrics.

Thermal Comfort: It is determined by the index PMV (Predictive Mean Vote) that is calculated by Fanger's equation [28]. PMV predicts the mean thermal sensation vote on a standard scale for a large group of persons. The American Society of Heating Refrigerating and Air Conditioning Engineers (ASHRAE) developed the thermal comfort index by using coding -3 for cold, -2 for cool, -1 for slightly cool, 0 for natural, +1 for slightly warm, +2 for warm, and +3 for hot. PMV has been adopted by the ISO 7730 standard [29]. The ISO recommends maintaining PMV at level 0 with a tolerance of 0.5 as the best thermal comfort. We calculate the reward function for thermal comfort T_c during a time slot as

$$T_c = \begin{cases} 0, & PMV \leqslant P \\ |PMV - P|, & PMV > |P| \end{cases} \tag{2.8}$$

The occupants can feel comfort when PMV value is within an acceptable range. We denote the range as $[-P, P]$, where P is the threshold for PMV value. If the PMV value lies within $[-P, P]$, it will not incur a penalty. Otherwise, it will incur a penalty for the occupants' dissatisfaction with the building thermal condition.

Visual Comfort: The research on visual comfort is dominated by studies analyzing the presence of an adequate amount of light where discomfort can be caused by either too low or too high level of light as glare. In this paper, the major glare metric is illuminance range [30]. The illuminance source includes daylight and electrical light. Thus, the main subsystems that can have an impact on visual comfort are blind system and lighting system. We calculate the reward function for visual comfort V_c during a time slot as

$$V_c = \begin{cases} -F - M_L, & F < M_L \\ 0, & M_L \leqslant F \leqslant M_H \\ F - M_H, & F > M_H \end{cases} \tag{2.9}$$

The occupants can feel comfort when illuminance value F is within an acceptable range. We denote the range as $[M_L,\ M_H]$, where M is the threshold for illuminance value. If the illuminance value lies within $[M_L,\ M_H]$, it will not incur a penalty. Otherwise, it will incur the penalty for the occupants' dissatisfaction with the building illuminance condition.

Indoor Air Quality: Carbon dioxide (CO_2) concentration in a building is used as a proxy for air quality [31]. The carbon dioxide concentration comes from building's users. There are various other sources of pollution (NOx, Total Volatile Organic Compounds (TVOC), respirable particles, etc.). Ventilation is an important means for controlling indoor air quality (IAQ) in buildings [32]. Ventilation in this work mainly comes from the HVAC system and the window system. We calculate the reward function for indoor air condition I_c during a time slot as

$$I_c = \begin{cases} -C - A_L, & C < A_L \\ 0, & A_L \leqslant C \leqslant A_H \\ C - A_H, & C > A_H \end{cases} \tag{2.10}$$

The occupants can feel comfort when carbon dioxide concentration value C is within an acceptable range. We denote the range as $[A_L, A_H]$, where A is the threshold for dioxide concentration value. If the dioxide concentration value lies within $[A_L, A_H]$, it will not incur a penalty. Otherwise, it will incur a penalty for the occupants' dissatisfaction with the building indoor air quality.

2.4 Implementation of Octopus

In this section, we illustrate in detail the implementation of OCTOPUS including platform setup, HVAC modeling and calibration, and OCTOPUS training.

2.4.1 Platform setup

Our building model is rendered using SketchUp [33]. It replicates a LEED Gold Certified Building in our University Campus. Using OpenStudio, the HVAC, lighting, blind and window system are installed in the building/zones. The control scheme - OCTOPUS is implemented using Tensorflow, which is an open-source machine learning library for Python. Using the Building Control Virtual Test Bed (BCVTB), a Ptolemy II platform that enables co-simulation across different models [34], we implement the control of each zone temperature set points, blinds, lighting and window schedule during each action time in EnergyPlus for our Building alongside weather data. OCTOPUS is modeled using EnergyPlus version 8.6 [35]. We train OCTOPUS based on 10-year weather data from two different cities, Merced, CA and Chicago, IL due to their distinct weather characteristics. The weather data for Merced has intensive solar radiation and large variance in temperature, while Chicago is classified as hot-summer humid continental with four distinct seasons. To train our model, we define an "episode" as one inner for loop of Algorithm 1.

2.4.2 Rule Based Method

We implement a rule-based method based on our current campus building control policy. This policy was first set up at commissioning time by a mechanical engineering company, and then it was further optimized by two experienced HVAC engineers when going over the LEED certification process.

First, we assign different zone temperature setpoints. Each zone has a separate heating and cooling setpoint. The heating setpoint is set to 70 °F, and the cooling setpoint to 74 °F during the warm-up stage. The cooling setpoint is limited between 72°F and 80°F, and the heating setpoint is limited between 65°F and 72°F. Second, we set control restrictions and actuator limits and control inputs are subject to the following constraints: the heating setpoint should not exceed the cooling setpoint minus 1 °F. The adjustment will move both the existing heating and cooling setpoints upwards or downwards by the same amount unless the limit has been reached. Third, for the control Loops: two separate control loops operate to maintain space temperature at setpoint, the Cooling Loop and the Heating Loop. Both loops are continuously active.

2.4.3 HVAC Modeling and Calibration

The purpose of the calibration is to ensure the energy model can generate energy use results close to the measured values in the target building using actual inputs, including weather, occupancy schedule, and the HVAC system parameters and controls.

The first step of the calibration is to collect the real weather data from a public weather station for the period to be tested. We use a Dark Sky's API, a public weather website, to collect real weather data for three months. The second step is to replace the default occupancy schedules in the simulator with the actual occupancy

Table 2.1: Model Calibration Parameters

Parameter	Range	Adoption
Infiltration Rate	0.01 m^3 $\sim$ 0.5 m^3	0.05 m^3
Window Type/Area	Single Pane/$1m^2 \sim 4m^2$	$2m^2$
Window Thickness	$3mm \sim 6mm$	$3mm$
Fan Efficiency	0.5 $\sim$ 0.8	0.7
Blind Type/Thickness	Interior Blind/$1mm \sim 6mm$	$1mm$

Table 2.2: Modeling Error after Calibration

	MBE	CVRMSE
February (hourly temperature)	-1.48%	5.32%
March (hourly temperature)	-0.26%	4.95%
April (hourly temperature)	1.20%	5.06%
May (hourly temperature)	0.48%	4.38%
February - May(monthly energy)	-3.83%	12.33%

schedules collected from the real target building using ThermoSense [36]. This system was installed in the target building on our campus and allows the collection of fine grain occupancy data at the zone level in the building, allowing the evaluation using accurate occupancy patterns. We used the hourly occupancy data from 3 months as the occupancy schedule in our simulated building by EnergyPlus. The third step is to calibrate certain system and control parameters to match those in the target building we want to replicate. This involves multiple issues, including (a) the selection of the parameters to be calibrated, (b) the range of those parameters, and (c) the step used within the range. In our work, we use an N-factorial design with 5 parameters and ranges to be tested based on operational experience. We tested different combinations of HVAC system parameters (Infiltration rate) and control (mass flow rate, heating, and cooling setpoints) and found the combination that minimized the calibrated error (see below). The selected calibration parameters are listed in Table 2.1 with their calibration ranges and values selected. The final step is to compare the calibrated error between the calibrated model and the actual measured zone temperature and energy consumption stored in the operational building database. The whole calibration process of modeling our building takes nearly one month.

ASHRAE Guideline 14-2002 [37] defines the evaluation criteria to calibrate BEM models. According to the Guidelines, monthly and hourly data can be used for cal-

Table 2.3: Human Comfort Statistical Results for Rule Based, DDQN-HVAC and OCTOPUS Schemes

Location	Method	Metric	PMV		Illuminance (lux)		CO_2 Concentration (ppm)		Energy Consumption (kWh)	
			January	July	January	July	January	July	January	July
Merced	Rule Based Method	Mean	0.03	-0.25	576.78	646.45	623.61	668.03		
		Std	0.11	0.13	152.54	157.11	120.64	181.22	1990.99	3583.03
		Violation rate	0	2%	0.94%	0	0.3%	3.629%		
	DDQN HVAC [13]	Mean	-0.19	0.28	576.78	646.45	625.62	648.01		
		Std	0.21	0.11	152.54	157.11	122.62	120.57	1859.10	3335.58
		Violation rate	2.99%	4.4%	0.94%	0	0	0.2%		
	OCTOPUS	Mean	-0.31	0.27	587.12	569.88	594.77	612.33		
		Std	0.2	0.10	382.27	75.83	111.59	110.35	1756.24	2941.46
		Violation rate	5.7%	2.5%	0.26%	0.2%	1.31%	0.33%		
Chicago	Rule Based Method	Mean	-0.28	-0.15	583.27	637.07	610.26	638.33		
		Std	0.11	0.02	163.96	151.37	63.94	151.37	3848.61	3309.56
		Violation rate	3.09%	0	1.1%	0	0	0		
	DDQN HVAC [13]	Mean	-0.32	0.24	583.27	637.07	612.74	649.32		
		Std	0.08	0.07	163.96	151.37	65.09	90.16	3605.21	3078.67
		Violation rate	3.7%	2.9%	1.1 %	0	0	0		
	OCTOPUS	Mean	-0.4	0.29	598.34	544.09	640.31	633.71		
		Std	0.1	0.11	259.88	55.37	99.85	111.04	3496.54	2722.03
		Violation rate	4.2%	1.47%	1.6 %	0	1%	1.31%		

ibration. Mean Bias Error (MBE) and Coefficient of Variation of the Root Mean Squared Error (CVRMSE) are used as evaluation indices. The guideline states that the model should have an MBE of 5% and a CVRMSE of 15% relative to monthly calibration data. If hourly calibration data are used, these requirements should be 10% and 30%, respectively. In our case, hourly data is used to calculate the error metrics for the average zone temperature. We choose monthly data to calculate energy error metrics because energy data can only be obtained monthly. The calibration results for zone temperature and energy consumption are shown in Table 4.17. It is shown that less than 2% NMBE and less than 6% CVRMSE for the zone temperature can be achieved with the optimal parameter setting. We found that both the CVRMSE for the monthly heating and cooling energy demand are relatively large, but the NMBE and CVRMSE are still within the acceptable range. This means the model can achieve accurate calculations for the monthly energy.

2.4.4 OCTOPUS Training

10-year weather data for training from the two locations tested (Merced, CA and Chicago, IL) is randomly divided, with eight years used for training and the

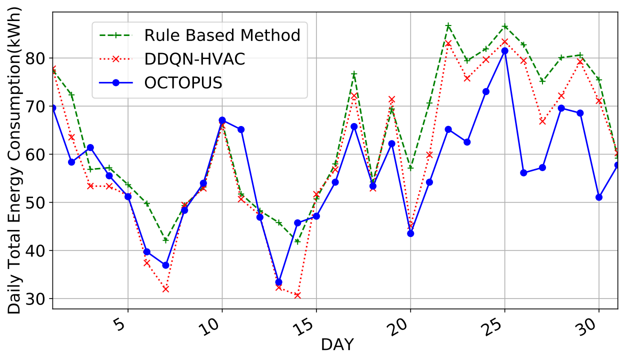


Figure 2.9: Daily Energy Consumption of Control Methods.

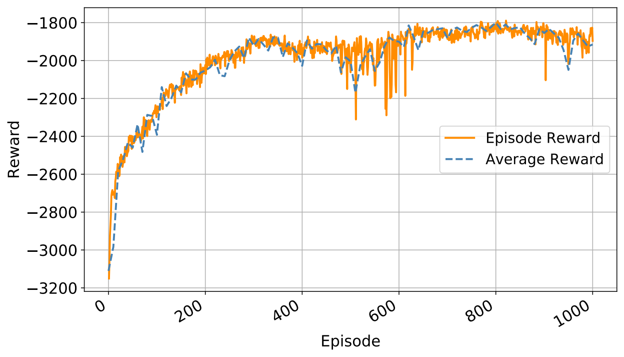


Figure 2.10: The Convergence of OCTOPUS.

remaining two years used for testing. In our implementation of OCTOPUS, we use the Adam optimizer [38] for gradient-based optimization with a learning rate of 10^{-4}. We train the agent with a minibatch size of 64 and a discount factor $\gamma = 0.99$. The target network is updated every 10^3 time steps. We use the rectified non-linearity (or ReLU) [39] for all hidden layers and linear activation on the output layers. The network has two hidden layers with 512 and 256 units in the shared network module and one hidden layer per branch with 128 units. The weights are initialized using the Xavier initialization [40] and the biases were initialized to zero. We used the prioritized replay with a buffer size of 10^6. To explore actions well in our building environment, we sample actions with a Gaussian noise throughout the training. The duration of each time (action) slot is 15 minutes. We achieved convergence of our reward function after 1000 episodes as explained in Section 3.5.3.

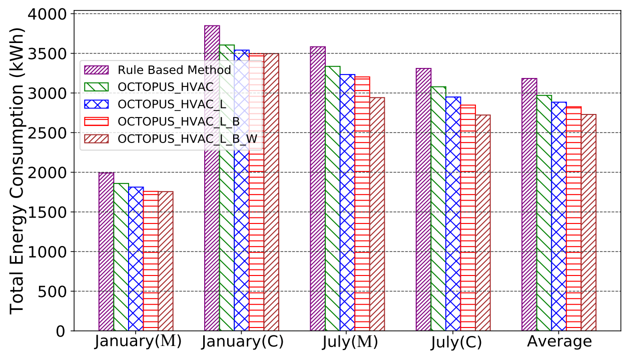


Figure 2.11: Performance Contribution of Each Subsystem.

2.5 Evaluation

In this section, we compare the performance of OCTOPUS with the rule-based method and the latest DRL-based method.

2.5.1 Experiment Setting

The implementation of the rule-based HVAC control has been introduced in Section 2.4.2. The rule-based method only controls the HVAC system. For the conventional DRL-based method, we implement the dueling DQN architecture used in [13], which controls the water-based heating system. We name that work as DDQN-HVAC in our comparison. Since these two benchmarks do not control the light system, for a fair comparison, we initialize the lights on in all experiments. OCTOPUS may dim the lights if the blind is open during the day. In addition, the two benchmarks always leave the blind and window system closed.

The three human comfort metrics are measured by PMV, Illuminance, and carbon dioxide concentration. We set the acceptable range of three human comfort metrics according to building standards and previous experiences in related work. The comfort range of PMV is set to -0.5 to 0.5 [41]. The comfort range of illuminance is set to 500-1000 lux [30]. The comfort range of carbon dioxide concentration is set to 400-1000 ppm [32].

We use three control methods to control the building we modeled in Section 2.4 for two months (January and July) and at two places with distinct weather patterns. Table 3.1 shows the human comfort results of three control methods and their energy consumption. The violation rate is calculated as the time when the value of a human comfort metric falls beyond its acceptable range divided by the total simulated time. Other quality of service metrics, including the amount by the which the violation occurred, or combination of amount and time will be explored in future work.

2.5.2 Human Comfort

From the results in Table 3.1, we see that all three methods can maintain the PMV value in the desired range for most of the time since the violation rate is low. The average PMV violation rate of OCTOPUS and DDQN-HVAC is higher than the rule-based method by 2.19% and 2.22% respectively. The reason for this is that the DRL-based methods try to save more energy by setting the PMV to a value close to the boundary of the acceptable range. It can be observed in Table 3.1 that the average PMV value of OCTOPUS and DDQN-HVAC (-0.36 and -0.26) is closer to the range boundary (-0.5), compared with the rule-based method (-0.13).

For both visual comfort and indoor air quality, the three control methods provide a very small violation rate. For illuminance, the mean illuminance value of OCTOPUS and DDQN-HAVC is 590.69 lux and 610.89 lux respectively. OCTOPUS saves energy by utilizing natural light as much as possible. For indoor air quality, the average of CO_2 concentration of OCTOPUS, DDQN-HVAC, and rule-based method is 620.28 ppm, 633.92 ppm, and 635.06 ppm. OCTOPUS adjusts both window system and HVAC system to maintain the CO_2 concentration level within the desired range. DDQN-HVAC and the rule-based method only use the HVAC system.

2.5.3 Energy Efficiency

The results in Table 3.1 reveal that OCTOPUS save 14.26% and 8.1% energy on average, compared with the rule-based control method and DDQN-HVAC. In both cities, OCTOPUS achieves similar performance gain. OCTOPUS reduces the energy consumption of HVAC by using the other subsystems. Figure 2.9 shows a daily energy consumption of three control methods in January at Merced. In most days, OCTOPUS consumes less energy than the other two methods; however, OCTOPUS is not always the best although we see clear gains towards the second half of the month due to a change in weather temperature. The average range of outdoor temperature changes from 2°C$\sim$ 13°C in the first half of the month to -1°C $\sim$ 18°C in the second half of the month. OCTOPUS could use external air with the window open for more natural ventilation.

In Table 3.1, compared to the rule-based method and DDQN-HVAC, OCTOPUS saves more energy in July (17.6% and 11.7%) than in January (10.05% and 3.9%). In July, the outdoor air temperature range at Merced and Chicago is 15°C $\sim$ 42°C and 15°C $\sim$ 40°C respectively. The window can be opened when the temperature is within the acceptable range, in order to save the energy consumed by the HVAC system. However, in January, due to the cold weather at both places, the windows stay closed most of the time and cannot make much contribution to energy savings.

2.5.4 Performance Decomposition

We implement four versions of OCTOPUS to study the energy saving contribution of each subsystem, i.e., OCTOPUS just with the HVAC system (OCTOPUS _HVAC), OCTOPUS with HVAC and lighting (OCTOPUS _HVAC_L), OCTOPUS with HVAC, lighting and blind (OCTOPUS _HVAC_L_B) and OCTOPUS with all four subsystems (OCTOPUS _HVAC_L_B_W). Figure 2.11 depicts the energy consumption of these four versions in two different months and at two different places (Merced and Chicago). Compared with the rule-based method, OCTOPUS _HVAC can save 6.16% more energy by only considering HVAC. When the lighting system is added in OCTOPUS _HVAC_L, 2.73% more energy can be saved. If the blind system is further added in OCTOPUS _HVAC_L_B 1.93% more energy can be saved. Finally, when the window system is added in OCTOPUS _HVAC_L_B_W, 3.44% more energy can be saved. Four subsystems make different contributions to energy saving in January and July. In January, four subsystems (i.e., HVAC, lighting, blind and window) make 6.16%, 2.73%, 1.93% and 0% contribution of energy savings respectively. In July, the contribution of these subsystems changes to 5.9 %, 3.31 %, 1.99%, and 6.4% respectively. The most obvious difference between these two months is made by the window system (6.4%). The reason for this has been explained above. In January, the windows are closed almost all the time. In July, the cold outdoor air is used to cool down the building instead of using the HVAC system.

2.5.5 Convergence of OCTOPUS training

Figure 3.6 shows that the accumulated reward of OCTOPUS in each episode during a training process. We calculate the reward function every control time step (15 minutes), and thus one episode (one month) contains 2880 time steps. The accumulated reward of one episode (episode reward in Figure 3.6) is the sum of the rewards of 2880 time steps. From the results in Figure 3.6, we see that the episode reward increases and tends to be stable as the number of training episodes increases. When the episode reward does not change much, it means that we cannot do further to improve the learned control policy and thus the training process converges. As indicated in Figure 3.6, the training reward fluctuates between two adjacent episodes, because the number of time steps is large in one episode, i.e., 2880. The rewards calculated at some of these 2880 time steps may vary dynamically because we randomly choose some time steps by an exploration rate (determined by a Gaussian distribution with a standard deviation of 0.2). At these time steps, we do not use the action generated by the agent, but randomly choose an action to avoid local minimum convergence. If we smooth the episode reward using a sliding window of 10 episodes, the average reward in Figure 3.6 is more stable during the training.

2.6 Discussion

Deploying in a Real Building. Although we have developed a calibrated simulation model of a real building on our campus for training and evaluation, we have not deployed OCTOPUS in the building, because we do not have access to automatic blind and window system at the moment. We are seeking financial support to work with our facility team for a possible upgrade. OCTOPUS is designed for real deployment in buildings. For a new building, we need to build an EnergyPlus model for it and calibrate the model using real building operation data. After training the OCTOPUS control agent using the calibrated simulation model and real weather data, we can deploy the trained agent in the building for real-time control. For a certain action interval (e.g., every 10 mins), the OCTOPUS control agent takes the state of the building as input and generates the control actions of four subsystems. OCTOPUS can provide real-time control, as one inference only takes 22 ms. We plan to deploy OCTOPUS in a real building in our future work.

Scalability of OCTOPUS. OCTOPUS can work in a one-zone building with one HVAC system, lighting zone, blind and window. However, a realistic building (or even a small home) is usually equipped with many lighting zones, blinds and windows which may take different actions in one subsystem. OCTOPUS may solve this scalability problem by increasing the number of BDQ branches, i.e., each branch corresponds to one subsystem in each zone of a building. We will tackle this scalability problem in our future work.

Building Model Calibration. A critical component of our architecture is the use of a calibrated building model that is close to the target building, allowing us to generate sufficient data for our training needs. However, getting a calibrated model ”right” is a tedious process of trial-and-error over a large number of parameters. Out of the thousands of parameters available in EnergyPlus, we use our experience and consulted experts to determine both the most important parameters and a sensible range of values to explore (it took us four weeks to get it ”right”). However, there is no magic bullet, and this may become a problem, especially for unusual building architectures or specialized HVAC systems that may not be trivial to replicate in a simulation environment.

Accepting Users' Feedback. Some existing work [42] allows users to send their feedback to the control server. The feedback can represent a user's personalized preference on different human comfort metrics and will be considered in the control decision process. OCTOPUS can easily accept users' feedback to train a better agent model by making a small modification, i.e., changing the calculated comfort values in the reward function by the users' feedback. This can be used for the initial training or for updated training (once deployed). For example, the OCTOPUS control agent can be trained incrementally with a certain time interval (e.g., one month). The newly-trained agent will be used for real-time.

2.7 Conclusions

This paper proposes OCTOPUS, a DRL-based control system for buildings that holistically controls many subsystems in modern buildings (e.g., HVAC, light, blind, window) and manages the trade-offs between energy use and human comfort. As part of our architecture, we develop a system that addresses the issues of large action state, a novel reward function based on energy and comfort, and data requirements for training using existing historical weather data together with a calibrated simulator for the target building. We compare our results with both the state-of-art rule-based control scheme obtained from a LEED Gold certified building, a DRL scheme used for optimized heating in the literature, and show that we can get 14.26% and 8.1% energy savings while maintaining (and sometime even improving) human comfort values for temperature, air quality and lighting.

Chapter 3

Model-Based Deep Reinforcement Learning for Multi-zone Building Control

Deep reinforcement learning has been widely studied for controlling Heating, Ventilation, and Air conditioning (HVAC) systems. Most of the existing works are focused on Model-Free Reinforcement Learning (MFRL), which learns an agent by extensively trial-and-error interaction with a real building.

However, one of the fundamental problems with MFRL is the very large amount of training data required to converge to acceptable performance. Although simulation models have been used to generate sufficient training data to accelerate the training process, MFRL needs a high-fidelity building model for simulation, which is also hard to calibrate.

As a result, Model-Based Reinforcement Learning (MBRL) has been used for HVAC control. While MBRL schemes can achieve excellent sample efficiency (i.e. less training data), they often lag behind model-free approaches in terms of asymptotic control performance (i.e. high energy savings while meeting occupants' thermal comfort).

In this paper, we conduct a set of experiments to analyze the limitations of current MBRL-based HVAC control methods, in terms of model uncertainty and controller effectiveness. Using the lessons learned, we develop MB^{2}C, a novel MBRL-based HVAC control system that can achieve high control performance with excellent sample efficiency. MB^{2}C learns the building dynamics by employing an ensemble of environment-conditioned neural networks. It then applies a new control method, Model Predictive Path Integral (MPPI), for HVAC control. It produces candidate action sequences by using an importance sampling weighted algorithm that scales better to high state and action dimensions of multi-zone buildings. We evaluate MB^{2}C using EnergyPlus simulations in a five-zone office building. The results show that MB^{2}C can achieve 8.23% more energy savings compared to the state-of-the-art MBRL solution while maintaining similar thermal comfort. MB^{2}C can reduce the training data set by an order of magnitude (10.52$\times$) while achieving comparable performance to MFRL approaches.

3.1 Introduction

Buildings account for 40% of energy usage in the US and 50% of that energy goes to Heating, Ventilation, and Air Conditioning (HVAC) [43]. Rule-based Control (RBC) is widely used to set actuators (e.g., heating or cooling temperature, and fan speed) in HVAC systems [44]. One of the main advantages is that they are easy to understand. However, RBC "rules" are usually set some if-then rules using many times static thresholds based on the rule-of-thumb rules and the experience of engineers and facility managers. They have two fundamental problems: first, they do not scale well with the problem size, as the buildings become larger and more complex, rules must be added; second, they do not handle incomplete or incorrect information very well, an occurrence common in buildings in practice; and finally, they do not necessarily provide a guarantee of optimal control.

Model Predictive Control (MPC) has been widely studied to address these drawbacks by finding optimal control actions based on an analytical building model [11, 45]. Normally, an optimization problem is formulated with the building model and some constraints, and analytic gradient computation is used to optimize over actions and building states simultaneously. However, this often requires convexification of the cost function and first or second-order approximations of building dynamics [46] in order to solve the optimization problem fast and to scale well. As a result, the models used in current solutions are simplified to deal with the parameter-fitting data requirement and computational complexity [11, 45].

Deep Reinforcement Learning (DRL) has been widely studied for HVAC control [47, 13, 48, 49]. Current solutions mainly adopt Model-Free Reinforcement Learning (MFRL), which learns an optimal HVAC control policy by trial-and-error interactions with a real building. However, MFRL requires a large amount of interactions to converge, e.g., in our experiments, it requires 500,000 timesteps (5200 days) to achieve a high control performance. Although a simulated building model can be used to accelerate the training process, it needs a high-fidelity model, which is hard to calibrate [13, 47]. Recently, Model-Based Reinforcement Learning (MBRL) has been tested for HVAC control to achieve high data efficiency [50]. The HVAC system dynamics is first learned using a neural network based on historical HVAC data. Based on the learned building dynamics model, an MPC controller tries to find the optimal control action by using a Random Shooting (RS) method [50]. For controlling a single-zone HVAC system, an MBRL-based approach saves approximately 10$\times$ training time of the MFRL approach, while achieving comparable performance [50]. However, most of the commercial buildings are multi-zone buildings [51]. In addition to the above scheme not being suitable for multi-zone HVAC systems, MBRL often lags behind the MFRL schemes in terms of control performance (high energy saving while meeting the thermal comfort of occupants).

To overcome these limitations, this paper presents MB^{2}C, a novel MBRL-based HVAC control approach that can achieve both the data/sample efficiency of MBRL and the control performance of MFRL. The design goal of MB^{2}C is to meet the thermal comfort requirements of the occupants while saving as much energy as possible. The energy consumed by a building HVAC system and the thermal comfort of occupants are determined by a set of factors, including current state of all zones, the outdoor weather and the control actions we are about to take (e.g. temperature setpoints). In a multi-zone building, the control actions can be represented as a vector A_s, which is a combination of control actions for all thermal zones. MB^{2}C finds the best A_s from all possible action combinations A_{all} for each control cycle. The best A$_s$ maintains the thermal comfort in its acceptable range for the entire control interval with the lowest energy consumption. MB^{2}C is mainly composed of two parts: (a) a building dynamics model, and (b) an HVAC control algorithm.

Our building dynamics model employs an ensemble of environment-conditioned neural networks. We use a neural network model that takes the current state of the building and the action to perform as input, and outputs a prediction of the next state of the building. To capture model uncertainty, we design a novel weighted ensemble learning algorithm that aggregates the results of multiple building dynamics models by dynamically adjusting the weight of each model according to their accuracy. We also adopt an environment-conditioned neural network architecture by separating the action-depended state items (e.g., zone temperature) and the environment-related state items (e.g., outside temperature), since the latter cannot be actuated by control actions.

Based on a learned building dynamics model, a flexible way to solve the control optimization problem is a shooting method that samples stochastic action trajectories for a number of incoming time-steps [52]. An action trajectory is a set of actions for incoming H time-steps. Every time, H time-steps are evaluated, but only the first action will be executed at the next time-step. For example, RS has been used in the latest MBRL-based HVAC control solution [50], which entails sampling candidate actions from a uniform distribution. However, RS is insufficient to find the best action trajectory, because randomly-shot action trajectories may not include it. We adopt Model Predictive Path Integral (MPPI) control method, which has shown promising performance in robotics control [53]. MPPI derives an optimal control action as the first action of a noise-weighted average over sampled control action trajectories by changing the initial control input and variance of the sampling distribution. We customize MPPI control for building HVAC control under the MBRL-based framework with the best parameter setting.

We implement MB^{2}C in Tensorflow, an open-source machine learning library in Python, with a 3-layer neural network as the building dynamics model and an MPPI-based control algorithm. We study the performance of MB^{2}C and compare it with benchmark methods by controlling a building of five thermal zones. We conduct a variety of simulations in EnergyPlus for evaluation. Extensive simulations reveal that MB^{2}C outperforms the latest model-based DRL method by 8.23% in total energy consumption of the building, without scarifying thermal comfort. Compared with the model-free DRL approach, we reduce the training convergence time by 10.52$\times$, more than an order of magnitude improvement.

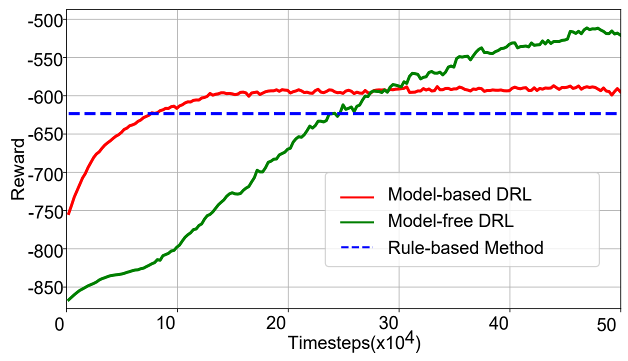


Figure 3.1: Convergence time and the achieved reward.

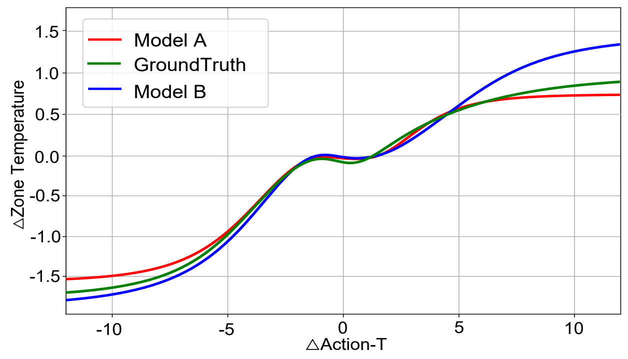


Figure 3.2: Uncertainty of the building dynamics model.

3.2 Related work

Model Predictive Control for HVAC. MPC solves an optimal control problem iteratively over a receding time horizon. [11] proposed an MPC approach for HVAC control, which minimizes energy use while satisfying occupant comfort constraints. A very recent MPC work, OFFICE [45], proposed a novel MPC framework that optimally manages the trade-off between energy cost and quality of comfort to the building users. OFFICE uses a gray-box approach, where a parametrized first-principled model is used, and the parameter of the model are dynamically learned and updated over time. In our case, we use a black-box approach, where the neural network learns from scratch the relationships between inputs and outputs in the system. Also, the MPC controller used is also different. While OFFICE uses an interior-point method based on a derivable function to find the optimal solution, we use an MPPI controller, which uses sample noise for the exploration around the default values as a search mechanism to find the best optimization solution.

Model-free DRL for HVAC control. Deep Reinforcement Learning has been

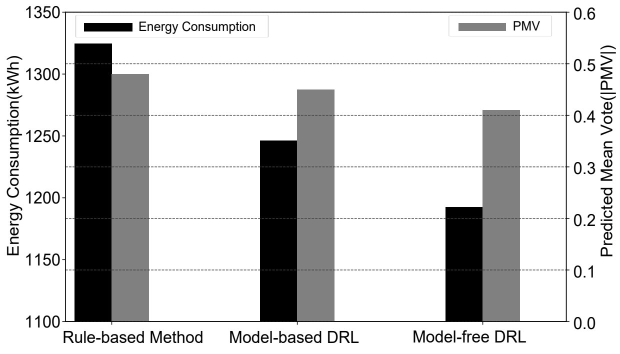


Figure 3.3: Random shooting in the model-based DRL method.

applied to many areas [54, 55, 56, 57, 14, 58, 59, 60]. In particular, MFRL techniques have demonstrated the potential optimal HVAC controls. In MFRL schemes, the agent learns the policy by extensively trial-and-error interaction with the environment. [49] leveraged RL to calculate thermostat set-points to balance between occupant comfort and energy efficiency. [13] implemented and deployed a DRL-based control method for radiant heating systems in a real-life office building. A holistic building control accounting for HVAC, lighting, window opening and blind inclination was studied using branching dueling Q-network (BDQ) in [47]. However, practical application of RL was limited by its sample complexity, i.e. the long training time required to learn control strategies, especially for tasks associated with a large state-action space. Gnu-RL [61] adopted a differentiable MPC policy, which encodes domain knowledge on planning and system dynamics, making it both data-efficient and interpretive. However, they assumed that dynamics of a water-based radiant heating system can be locally linearized. The assumption worked for the problems they considered, but it may not extrapolate to more complex problems like ours.

Model-based DRL for HVAC control. To reduce sample complexity, researchers have adopted model-based deep reinforcement learning for HVAC control [50]. In this work, they proposed an MBRL approach that learns the system dynamics using a neural network. Then, they adopt MPC using the learned system dynamics to perform control with RS method. MBRL method works well when the action and state dimension is low, like single-zone building. They often cannot achieve the final performance as model-free method when they are applied to high state and action dimensions of multi-zone buildings.

3.3 Motivation

To understand the performance of a state-of-the-art MBRL method [50], we perform a set of simulations in EnergyPlus for a building with five zones. All system settings are the same as [50], except the state and action dimension is higher for the five-zone building, i.e. a multi-zone building instead of a single zone. We also implement a simple MFRL-based method, Proximal Policy Optimization (PPO) [62], for comparison in this preliminary evaluation. Thermal comfort is measured by PMV [28], which should be controlled within the range (-0.7∼0.7). The simulations are conducted with weather data for the month of January. The building is 463 m^2 in Fresno CA. It has windows in all four facades and glass doors in south and north facades. The south-facing glass is shaded by overhangs. For our 5-zone building, the state dimension is 37, including indoor air temperature, humidity, PMV, energy consumption for each zone and related outdoor environmental parameters; and the action dimension is 10, including cooling and heating set points for each zone.

Experiment results. Figure 3.2 shows the energy-saving performance of model-based and model-free DRL control method with 50×10^4 time-steps of training data. The reward means the energy-saving performance under the reasonable thermal comfort that is defined in Section 4.3.3. We evaluate the accumulated reward every 2976 time-steps (one month). The performance of the rule-based method is a straight line, because its reward does not change as the weather data and building environment are deterministic.

From Figure 3.2, we can see that model-based DRL and PPO method need 7.5×10^4 and 23.75×10^4 time-steps to behave better performance than rule-based method. For converge time, the model-based method needs 11.5×10^4 and the PPO method needs 50×10^4 time-steps. The model-based method is 4.38× more data-efficient than PPO method. However, in the long run, the model-free method eventually outperforms the model-based method. It's easy to see that the model-free method is a trial and error method and the performance increases when using more training data. However, in this case, our model-based method cannot achieve the same performance as model-free method as the training data increases. The model-based method performs well when the action and state dimension is low (e.g., 9 in [50]). However, both the building dynamics model and the control method may not be efficient when

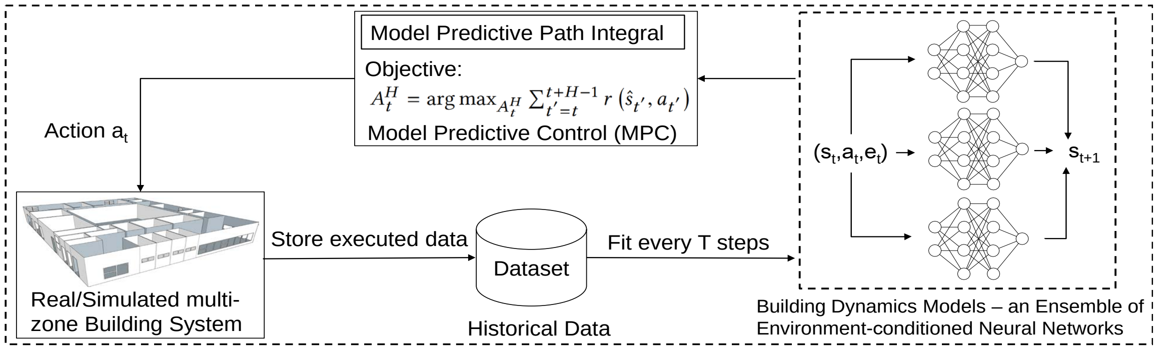

Model Predictive Path Integral
Objective:
$A_t^H = \arg\max_{A_t^H} \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, a_{t'})$
Model Predictive Control (MPC)
Action a_t
Real/Simulated multi-zone Building System
Store executed data
Dataset
Historical Data
Fit every T steps
(s_t, a_t, e_t)
s_{t+1}
Building Dynamics Models – an Ensemble of Environment-conditioned Neural Networks

Figure 3.4: Overall of the Proposed Building Energy Control Framework

the state and action dimension is high, like 47 in our 5-zone building.

Challenge 1 - Model Uncertainty. Neural network models may have epistemic uncertainty, due to the lack of sufficient data to uniquely model the underlying system [63, 64, 65, 66]. In an MBRL-based HVAC control system, a building dynamics model predicts the next state of the building, given the current state (e.g. current zone temperature) and a control action (e.g. actuators' temperature set-points). Even a small bias of the building dynamics model may significantly impact the decision of the controller [64, 65]. We conduct an experiment to study this uncertainty of the existing building dynamics model. We use 8000 historical data points to train the model, and 2000 data points for testing.

Figure 3.1 shows the predictive zone temperature as a function of the action performed. The x-axis shows the temperature differential between the supply temperature (action) and the zone temperature at time t, and the y-axis shows the temperature differential between the zone temperature after and before actuation. The figure depicts the predicted temperatures of two neural network models and the ground truth. These two models have the same architecture and are trained with the same training data, but their training processes start with different initialization states. In the middle region of Figure 3.1, we have sufficient data, since most of the actions in the historical data do not change the state sharply. In this region, both models can accurately predict the next state. However, when the actions intend to change the state much, we do not have sufficient data for training, and the performance of the two models diverges.

Challenge 2 - Controller Effectiveness. RS generates N independent random action sequences $\{a_t, ...a_{t+H-1}\}$, where each sequence $A_i = \{a_0^i ... a_{H-1}^i\}$ for

$i = 1...N$ is of length H action. Given a reward function $r(s, a)$ that defines the task, and given future state predictions $\hat{s}_{t+1} = s_t + f_\theta(\hat{s}_t, a_t)$ from the learned dynamics model f_θ, the optimal action sequence A_{i^*} is selected as the one with the highest predicted reward: $i^* = \arg\max_i R_i = \arg\max_i \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, \hat{a}_{t'})$.

Figure 3.3 studies the energy consumption and thermal comfort of three HVAC control methods, including a rule-based method, a model-based method and a model-free method. To eliminate the impact of model uncertainty for the model-based method, we use the ground-truth states of the building as the results of the building dynamics model (i.e. perfect future state prediction). From the Figure 3.3, we can see that all three methods can meet the requirement of thermal comfort with same level of PMV value (0.48, 0.45, 0.41). The energy consumption of the model-based method is 4.70% higher than the model-free method. It is caused by RS control, because the building dynamics model used in the model-based method is perfect in this experiment.

Based on the previous observations, our main goal is to overcome the drawbacks of model uncertainty and controller effectiveness and find a method that is able to match the high performance of model-free methods while having the sample/data-efficiency of model-based methods.

3.4 Design of MB^{2}C

In this section, we describe the design of MB^{2}C, including model-based DRL for a multi-zone building control, the building dynamics model and its training details, online control action planning and in-situ update of the building dynamics model [67].

3.4.1 MB^{2}C Overview

Figure 4.4 shows the overview of MB^{2}C as a model-based DRL control approach [65] for multi-zone building HVAC systems. At a high level, MB^{2}C includes two key components, i.e., a building dynamics model and a Model Predictive Path Integral (MPPI) based controller. Our building dynamics model is built by an Ensemble of multiple Environment-conditioned Neural Networks (ENN). It takes the current state of the building HVAC system and a specific control action as input, and outputs the next state of the building HVAC system. Based on the historical data, we train the building dynamics model as a supervised learning process. With the trained building dynamics model, our MPPI-based controller can evaluate different control actions and find the best control action for next time step, which meets the thermal comfort requirement with minimal energy consumption.

When we deploy the system in a building, MB^{2}C executes the best control action by setting corresponding actuators every control cycle. At the same time, we accumulate building data traces, i.e., the next HVAC state determined by the current HVAC state and the executed control action. With the newly collected building traces, we can perform in-situ updating of the building dynamics model periodically (e.g., every week) with a sliding window of 2-months to improve its accuracy, as the seasonality of the data changes during the year. One iterative training process takes 25.32 minutes to finish using a laptop with Intel 4-core i7-6700 CPU and Nvidia GTX 960M GPU, and it can be performed in parallel when the current model is being used in the building; thus, the overhead of the iterative training process does not impact the usage of MB^{2}C in real buildings.

3.4.2 Model-Based Deep Reinforcement Learning for Multi-zone Building Control

We extend the current MBRL-based method to multi-zone building HVAC control, including the design of those key components.

Preliminaries for DRL

The goal of deep reinforcement learning is to learn a policy that maximizes the sum of future rewards. At each time step t, the controller is in state $s_t \in S$, executes some action $a_t \in A$, receives reward $r_t = r(s_t, a_t)$, and transitions to the next state s$_{t+1}$ according to some unknown dynamics function $f : S \times A \rightarrow S$. The goal at each time step is to take the action that maximizes the discounted sum of future rewards, given by $\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'})$, where $\gamma \in [0, 1]$ is a discount factor that prioritizes near-term rewards. Note that performing this policy extraction requires knowing the underlying reward function $r(s_t, a_t)$ that we use for planning actions under the learned model.

In model-based reinforcement learning, a model of the dynamics is used to make predictions, which is used for action selection. Let $f_\theta(s_t, a_t)$ denote a learned discrete-time dynamics function, parameterized by θ, that takes the current state s_t and action a_t and outputs an estimate of the next state at time $t + \Delta t$. We can then choose actions by solving the following optimization problem:

$$(a_t, ...a_{t+H-1}) = \arg\max_{a_t,...a_{t+H-1}} \sum_{t'=t}^{t+H-1} \gamma^{t'-t} r\left(s_{t'}, a_{t'}\right) \tag{3.1}$$

In other words, we will pick the action sequence that maximizes the discounted sum of reward of future H time-steps. In practice, it is often desirable to solve this optimization at each time step, execute only the first action from the sequence, and then re-plan at the next time step with updated state information. Such a control scheme is often referred to as model predictive control (MPC), and is known to compensate well for errors in the model.

State Design

The state is what the building dynamics model takes as input for the next prediction step. In this study, we separate the state into 2 parts: (a) the building state (s_{ti}), which are the state variables that change with our control actions; and (b) the environment state (e_{ti}), which are the state variables that do not change with our control actions.

Building State (s_{ti}) The building state vector that changes over time t for the ith zone consists of the following items: indoor air temperature(°C), indoor air relative humidity (%), PMV, heating energy consumption (kWh) and cooling energy consumption (kWh).

Environment State (e_{ti}) The environment state vector that changes over time t for the ith consists of the following items: outdoor air temperature (°C), outdoor air relative humidity (%), diffuse solar radiation (W/m^2), direct solar radiation (W/m^2), solar incident angle (°), wind speed (m/s), wind direction and occupancy flag (0 or 1). The occupancy flag is an indicator to detect whether there are people in the ith zone, and it is the only element in the vector that changes per zone.

Taking our 5-zone building as an example, the state dimension is 37 including the building, and environment state variables.

Action Design

The action vector (a_{ti}) shows the actuation variables used by the controller to control the building state (s_{ti}). The action state vector that changes over time t for the ith zone consists of the following items: cooling temperature set-point and the heating temperature set-point (both in °C). Given the current state (s_{ti} and e_{ti}) and action (a_{ti}), we want the controller to find the most suitable action combinations $(a_{(t+1)i})$ for all the zones to balance energy consumption and thermal comfort metrics. The action dimension is 10 in our five-zone building.

Reward Design

The reward function controls the optimization parameters that want to be maximized when the agent performs an action (a_{ti}) to transition from the building state s_{ti} to $s_{(t+1)i}$. Both thermal comfort and energy consumption should be incorporated. The reward function is defined as follows:

$$R = -\sum_{i=1}^{N} \left(\rho Norm(|PMV_i|) + Norm(E_i)\right), \tag{3.2}$$

where E is heating and cooling energy consumption for each zone, we use Fanger's formula for the Predictive Mean Vote (PMV) [28] to estimate comfortable temperature bounds for the "standard" occupant within the current seasonal conditions, as defined by ASHRAE standard 55 [68]. The maximum high/low end of the comfort range for Class C environments has PMV values of +/- 0.7. ρ is used to balance the relative importance between energy consumption and thermal comfort. We use $\rho = 4$ during occupied periods and 0.1 during unoccupied periods since the range of human comfort and energy consumption is different during occupied and unoccupied periods. The reward evaluates the actions to meet the requirement of thermal comfort of all the occupants in the building. N is the number of zones. In the following sections, we will remove the i index for each zone to simplify the notation.

3.4.3 Learning the Building Dynamics

We require a parameterization of the building dynamics model that can cope with high-dimensional state and action spaces, and the complex dynamics of a multi-zone

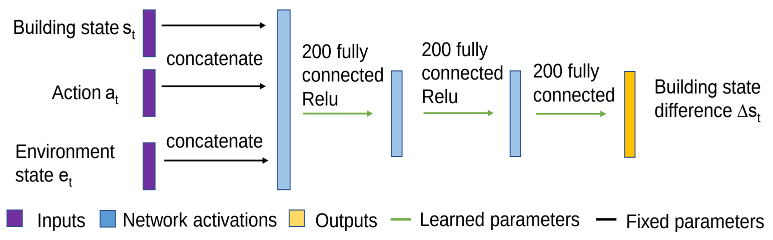


Figure 3.5: Environment-conditioned neural network for our Building Dynamics Model.

building. Therefore, we represent the dynamics function $\hat{f}_\theta(s_t, a_t)$ as a multi-layer neural network, parameterized by θ. This function outputs the predicted change in state that occurs as a result of executing action a_t from state s_t, over the time step duration of Δt. Thus, the predicted next state is given by: $\hat{s}_{t+1} = s_t + \hat{f}_\theta(s_t, a_t)$. While choosing too small of a Δt leads to too small of a state difference to allow meaningful learning, increasing the Δt too much can also make the learning process more difficult because it increases the complexity of the underlying continuous-time dynamics.

Environment-conditioned Neural Network Architecture

We define a neural network model $\hat{f}_\theta(s_t, a_t)$ for the building dynamics. In order to make the model achieve both good predictive accuracies and tractable computational optimization, we propose a simple and highly effective method for incorporating environment information. We formulate an environment-conditioned dynamics model $\hat{f}_\theta(s_t, a_t, e_t)$ that takes as input not only the current building state s_t and action a_t, but also the current environment state e_t. The model architecture is shown in Figure 3.5. The building state vector s_t, the action vector a_t and the environment state vector e_t are concatenated together and then are passed through two hidden layers and a final output layer. As opposed to a straightforward outputting of all the related states (building and environment), we produce a prediction of building state difference $\Delta\hat{s}_t$. This reduces the burden of the model to learn the changes in the environment that are not necessary. We provide the ground truth value for environment state, e.g., weather data and occupancy [61].

Weighted Ensemble Learning

As prior work [64, 65] has shown, capturing epistemic uncertainty in the network weights is important in model-based RL, especially with high capacity models that are liable to over-fit to the training set and extrapolate erroneously outside of it. To solve epistemic uncertainty, we propose a weighted ensemble learning algorithm, which approximates the posterior $p(\theta|D)$ with a set of M models, each with parameters θ_i. For deep models, it is sufficient to simply initialize each model θ_i with a different random initialization θ_i^0 and use different batches of data D_i at each training step.

We have M environmental-conditioned models. The input for all M models is the same and it includes the building and environment states and actions. To evaluate the performance of each model, we calculate the mean square error (MSE) of the past C timesteps (4 in our case) for each model compared to the ground truth for N states using Equation 3.3.

$$MSE = \textstyle\sum_{i=1}^{C} \sum_{j=1}^{N} \phi^C \left| f_\theta \left(s_{i,j}, a_{i,j}\right) - \hat{f}_{true} \right|^2 \tag{3.3}$$

We introduce a temporal discount factor ϕ (0.9 in our case) that is used to evaluate how important past model error to the current model error. The temporal discount factor is a value between 0 and 1 since recent prediction cases are more important to the performance of current prediction. After we have the MSE for each model of past C timesteps, we first normalize the MSE to 0-1 scale. $Norm(x)$ is a normalization process, i.e., $Norm(x) = (x - x_{min})/(x_{max} - x_{min})$. Then we calculate the weight ratio W for all models by Equation 3.4.

$$W = \tfrac{1 - Norm(MSE_i)}{\sum_{i=1}^{M}(1 - Norm(MSE_i))} \tag{3.4}$$

The sum of all model's weight is 1. After that, we leverage Equation 3.5 to predict the next state.

$$s_{t+1} = \textstyle\sum_{i=1}^{M} W_i f_{\theta_i}\left(s, a\right) \tag{3.5}$$

This allows our method to dynamically adjust the weights in aggregating the M models (M=5 in our case) during the prediction. As the states result in unequal prediction accuracy, our method is more robust against this variance.

3.4.4 Training the Building Dynamics Model

In this section, we illustrate how we pre-process training data, and train the proposed ENN model.

Data Collection

We collect the training dataset $D(s_t, a_t, s_{t+1})$ by executing the rule-based controller at each time step, and recording the resulting data $\tau = (s(0), a(0), s(1), a(1), ..., s(T-2), a(T-2), s(T-1)))$ of length T. We note that these data are very different from the data the controller will end up executing when planning with this learned dynamics model and a given reward function $r(s_t, a_t)$ (Section 3.4.5), showing the ability of model-based methods to learn from off-policy data.

Data Preprocessing

We slice the collected data $\{\tau\}$ into training data inputs (s_t, a_t) and corresponding output labels $s_{t+1} - s_t$. In building HVAC control, states can be temperature, humidity ratio, energy consumption, etc. These measurements have various ranges and the weights of the losses will be different if we feed the raw values directly to train the neural network model. Thus, we subtract the mean of the states/action and divide by the standard deviation $x' = \frac{x-\bar{x}}{\sigma(x)}$, where x stands for state or action.

Training the ENN Dynamics Model

ENN model consists of an ensemble of models. To make sure the models behave differently on the same dataset D, we randomly initialize model parameter $\theta_1, \theta_2, ..., \theta_M$ for all the dynamics models and use different batches of data D at each training step. We train the dynamics model $\hat{f}_\theta(s_t, a_t)$ using stochastic gradient descent [69] by minimizing the Mean Square Error (MSE) between predicted delta observation and ground truth delta observation as follows:

$$\varepsilon(\theta) = \frac{1}{D} \sum_{(s_t, a_t, s_{t+1}) \in D} \frac{1}{2} \|(s_{t+1} - s_t) - \hat{f}_\theta(s_t, a_t)\|^2 \tag{3.6}$$

We use 5-year weather data from Fresno, CA and Chicago, IL for the ENN model training and a completely different one-year for testing in this study. We provide the

ENN model with ground truth information on future environment state, i.e. weather and occupancy [61]. In our implementation of ENN, we use the Adam optimizer [38] for gradient-based optimization with a learning rate of 10^{-3}. We train the ENN model with a batch size of 512 and a discount factor $\gamma = 0.99$. The number of epochs is 40. Each dynamics model consists of a neural network of two fully-connected hidden layers of size 200 with relu being nonlinear and a final fully-connected output layer. The weights and biases are initialized using the Xavier initialization process [40]. The number of samples for MPC controllers (RS, CEM, and MPPI) is 1000. The control cycle (timestep) is 15 minutes that is widely used in classic HVAC control [70]. We achieve convergence by 4.75x10^4 time-steps as explained in Section 3.5.3.

3.4.5 Online Control Action Planning

In our method, we use online planning with MPC to select actions via our model predictions. Given the building state s_t at time t, the prediction horizon H of the MPC controller, and an action sequence $a_{t:t+H} = \{a_t, ..., a_{t+H}\}$, the proposed ENN model $\hat{f}_\theta(s_t, a_t)$ produces a prediction over the resulting data $s_{t:t+H}$. At each time step t, the MPC controller applies the first action a_t of the sequence of optimized actions $A_t^H = \arg\max_{A_t^H} \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, a_{t'})$. We adopt the MPPI control method [53] to compute the optimal action sequence.

Model Predictive Path Integral (MPPI) Controller. MPPI control method has been applied to autonomously control a vehicle and get good performance. MPPI is an importance-sampling weighted algorithm and considers an update rule that more effectively integrates a larger number of samples into the distribution update. As derived by recent model-predictive path integral work [53], this general update rule takes the following form for time step t, from each of the K predicted trajectories:

$$a_t^{i+1} = a_t^i + \textstyle\sum_{k=1}^{K} \omega(\varepsilon^k)\epsilon_t^k \tag{3.7}$$

Where ω is the importance-sampling weight for each trajectory and ϵ is the noise for exploration. The action for timesteps t of $(i+1)th$ trajectory is the sum of the action for timesteps t of ith trajectory and the noise-weighted average over sampled trajectories.

As shown in the algorithm 2, an initial control sequence is done either by initializing the input buffer with zeros or by using a secondary controller such as rule-based method and using its inputs as the initial control sequence. We first sample H noise from a normal distribution. Then, we compute K trajectories for H finite horizon with Brownian motion. For each trajectory generated, a cost is computed and stored in memory (line 2-7).

Algorithm 2 MPPI Controller

```
Input: ENN dynamics model $\hat{f}_\theta(s_t, a_t)$  K: Number of samples, H: Length of horizon
       $(a_0, a_1, ...a_{H-1})$: Initial control sequence  $\lambda$:Control hyper-parameter
Output: The control sequence $a_{t:t+H}$
$s_0 \leftarrow GetStateEstimate()$
for $k$ = 0,1,...,K -1 do
    s $\leftarrow s_0$  Sample noise $\varepsilon^k = \{\epsilon_0^k, \epsilon_0^k, ...\epsilon_{H-1}^k\} \sim \mathbb{N}(\mu, \sigma)$
    for $t$ = 1,...,H do
        $s_t \leftarrow \hat{f}_\theta(s_{t-1}, a_{t-1} + \epsilon_{t-1}^k)$   $Cost(\varepsilon^k)$ += $-reward$ defined by equation 4.5
    end
end
$\beta \leftarrow min_k[Cost(\varepsilon^k)]$   $\eta \leftarrow \sum_{k=0}^{K-1} exp(-\frac{1}{\lambda}(Cost(\varepsilon^k) - \beta))$   for $k$ = 0,1,...,K -1 do
    $\omega(\varepsilon^k) \leftarrow \frac{1}{\eta} exp(Cost(\varepsilon^k) - \beta)$
end
for $t$ = 0,1,...,H -1 do
    $a_t^* = a_t + \sum_{k=1}^{K} \omega(\varepsilon^k)\epsilon_t^k$;
end
SendToActuators($a_0$)  for $t$ = 0,1,...,H -1 do
    $a_{t-1} = a_t$;
end
$a_{t-1} = Initialize(a_{t-1})$;
```

In model predictive control, optimization and execution take place simultaneously: a control sequence is computed, and then the first element of the sequence is executed. This process is repeated using the un-executed portion of the previous control sequence as the importance sampling trajectory for the next iteration. In order to ensure that at least one trajectory has non-zero mass (i.e., at least one trajectory has a lowest cost), we subtract the minimum cost of all the sampled trajectories from the cost function (line 9). Note that subtracting by a constant has no effect on the location of the minimum. In the second loop, we get the noise weighted average over K sampled trajectories (lines 10-11). The third loop computes an optimal input sequence using least cost of the trajectories for H finite horizons (lines 12-13). The top of the stack value is given to the actuators (line 14). After that, the whole input control sequence is left shifted by 1 (lines 15-16). To maintain the length of buffer, a_{init} is appended to the input control sequence (line 17). The states are then updated from the ENN model.

3.4.6 Putting It All Together

We summarize the working flow of MB^{2}C as follows. We first gather historical dataset D using a rule-based policy and randomly initialize model parameter $\theta_1, \theta_2, ..., \theta_M$ for ENN. Then we train the ENN model using this dataset by Equation 3.6. Finally, we deploy the learned ENN model and our MPPI controller in the real building for HVAC control.

For one control execution, we first obtain the current building state from sensors (e.g., zone temperature from a temperature sensor). After that, the best action sequence is sampled by MPPI controller with H horizon and the state is propagated by ENN model by solving the optimization problem defined in Equation 3.1. We execute the first action of the optimal action sequence in the building by setting corresponding actuators.

When MB^{2}C is running in the building, we can also collect building operation data, which is composed of control action execution records $D(s_t, a_t, s_{t+1})$, including current state, control action, and next state. We add the newly collected data into a sliding window for two months of data and train the ENN model from scratch again. We use a sliding window to adapt to the seasonality of the data, especially weather data. We randomly divide the training data set into a set of batch and update the weight through forward and backward propagation by feeding the data into the model. This process is called one epoch training after traversing all the batch of data. We will repeat this process for multiple epochs (40 in our current implementation) until the model converges. This is an iterative in-situ updating process to improve the accuracy of our building dynamic model.

3.5 Evaluation

In this section, we conduct a variety of experiments in EnergyPlus to evaluate the performance of MB^{2}C and three baselines by a set of performance metrics.

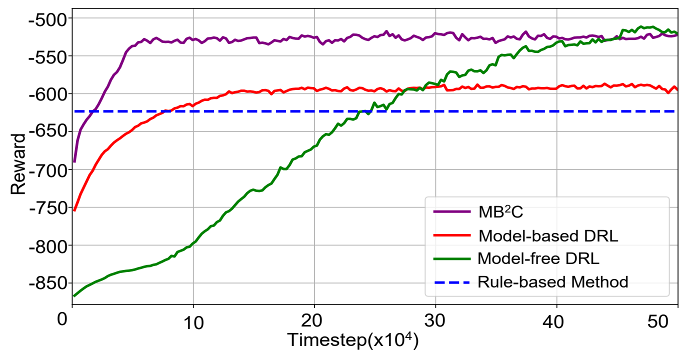


Figure 3.6: MB^{2}C Achieves both Data-Efficiency and High Performance.

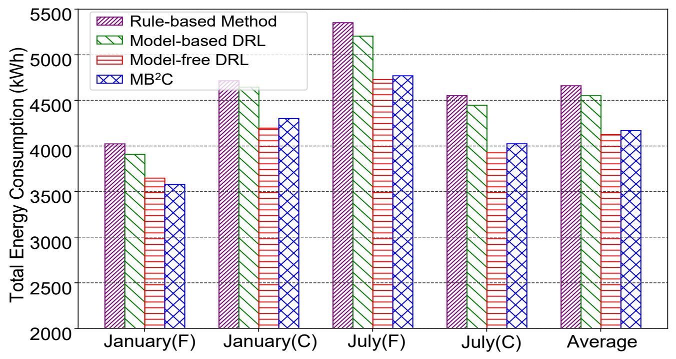


Figure 3.7: Energy Consumption of MB^{2}C and the Other Baselines.

3.5.1 Platform Setup

Building Example and its Dynamics Model in EnergyPlus In this work, we evaluate the performance of MB^{2}C in a building of 463 m^2 at Fresno, California. It is a single floor rectangular building of 5 thermal zones- 4 exterior zones, 1 interior zone. There are windows on all 4 facades. The HVAC system is single duct terminal reheat, which is composed by an Air Handler Unit (AHU) and Variable Air Volume (VAV) boxes. The AHU includes a fan, heating and cooling coils that can change the air's temperature. The VAV boxes take this pre-conditioned air from the main duct, heat it if necessary, and control the airflow provided to each zone.

Since we cannot conduct control experiments in the real building, we leverage a building model in EnergyPlus version 8.6 and conduct simulations with Typical Meteorological Year 3 (TMY3) weather data. In our implementation, the AHU set-point is set by default EnergyPlus control logic, and we only control the heating and cooling set-point in the VAV box.

EnergyPlus has been widely used to evaluate the HVAC control algorithm [13,

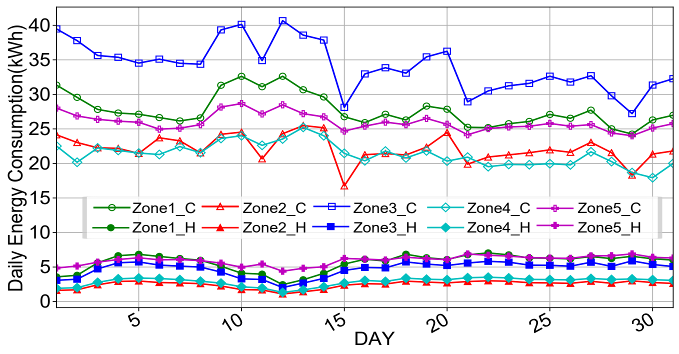


Figure 3.8: Daily Energy Consumption for Five Zones.

61, 50, 47]. There are four reasons why we choose EnergyPlus. First, we do not have one real building that allows us to conduct experiments. MB^{2}C could be deployed in a real building after we finish the ENN model training. Second, it is convenient to generate enough historical training data of rule-based method to train the ENN model. Third, in order to compare with a model-free DRL, we need a significant training data set to train these models since MFRL is not sample efficient. In our case, we need 5200 days (14+ years) of training data, which is unreasonable to obtain from real buildings. Finally, it is easy for us to evaluate the performance of different control algorithms under different locations, seasons and weather profiles.

MB^{2}C System Components As shown in Figure 4.4, MB^{2}C system includes two main parts: the building dynamics model ENN and the MPPI controller. We also need to store the newly collected building operation data for in-situ update of the building dynamics model. All these three components are all implemented in Tensorflow, which is an open-source machine learning library in Python. We use the building control virtual testbed (BCVTB) [34] for establishing a connection between EnergyPlus and MB^{2}C. We execute the control action by setting the temperature to a specific set point for each zone of our EnergyPlus building model during each control cycle.

3.5.2 Experiment Setting

We train ENN model based on the weather data from two different cities, Fresno, CA and Chicago, IL due to their distinct weather characteristics. The weather data for Fresno has intensive solar radiation and large variance in temperature, while

Chicago is classified as hot-summer humid continental with four distinct seasons.

We compare MB^{2}C with the three baselines. We execute these four control methods to control the building HVAC system using the same weather data for simulation.

Rule-based Method: We implement a rule-based method according to our current campus building control policy for training data generation and comparison evaluation. We assign different zone temperature set-points. Each zone has a separate heating and cooling set-point. The heating set-point is set to 70 °F, and the cooling set-point to 74 °F during the warm-up stage. The cooling set-point is limited between 72°F and 80°F, and the heating set-point is limited between 65°F and 72°F.

Model-free DRL: We implement Proximal Policy Optimization (PPO) [62] that is the default deep reinforcement learning algorithm at OpenAI because of its ease of use and good performance.

Model-based DRL with RS: For the conventional model-based method, we implement the deterministic neural network to model the building dynamics and RS method to choose the heating and cooling setpoints [50].

3.5.3 Experiment Results

We compare MB^{2}C with the above baselines by a set of performance metrics, including convergence analysis, energy efficiency and thermal comfort. We also study the performance of MB^{2}C, including its daily energy consumption for each zone, the performance gain of its key components, and its parameter setting.

Convergence Analysis

We first study the data efficiency of MB^{2}C and the other three baselines. For this study, we do not limit ourselves to a sliding window of two months for MB^{2}C, since the MFRL method requires copious amount of training data. Figure 3.6 shows that the accumulated reward of four control methods in each episode during a training process. One episode contains the data collected in one month, corresponding to 2976 time-step. We calculate the reward function every timestep. The reward in Figure 3.6 is the accumulated reward of one episode, i.e., the sum of the rewards of 2976 time-steps. From the results in Figure 3.6, we see that the episode reward increases and tends to be stable as the number of training episodes increases. When the episode

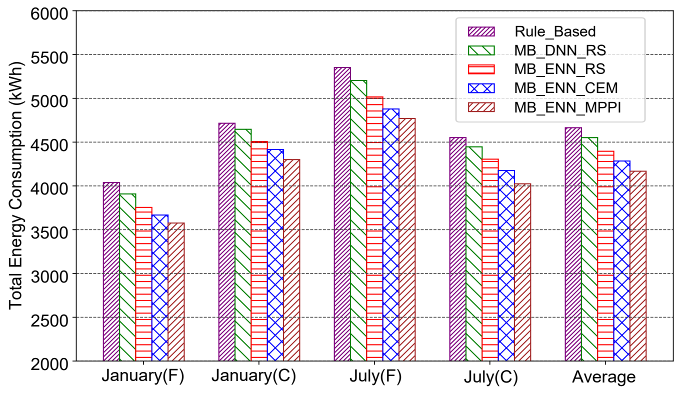


Figure 3.9: Energy Decomposition.

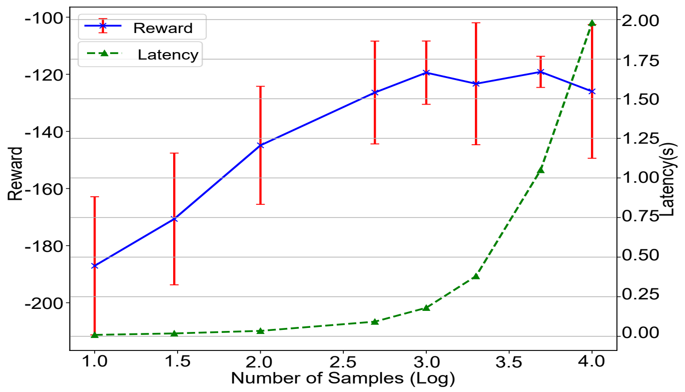


Figure 3.10: Samples of MPPI Controller.

reward does not change much, it means that we cannot do further to improve the learned control policy and thus the training process converges.

As indicated in Figure 3.6, MB^{2}C behaves better than rule-based method after the 1.75×10^4 time-steps. In this stage, the ENN model is first learned from off-line historical data. Then it can be deployed into real buildings and leverages the MPPI controller for exploration to further improve its performance. The model-based DRL and model-free DRL need 7.5×10^4 and 23.75×10^4 time-steps to behave better than rule-based method. MB^{2}C achieves 4.28× and 13.57× more data-efficient than model-based DRL and model-free DRL.

For convergence time, MB^{2}C converges faster than both model-based DRL and model-free DRL. MB^{2}C needs 4.75×10^4 and model-based DRL needs 11.5×10^4 time-steps. The model-free DRL needs 50x10^4 timesteps. MB^{2}C is 2.4× and 10.52× data-efficient than model-based DRL and model-free DRL with the same performance as model-free DRL.

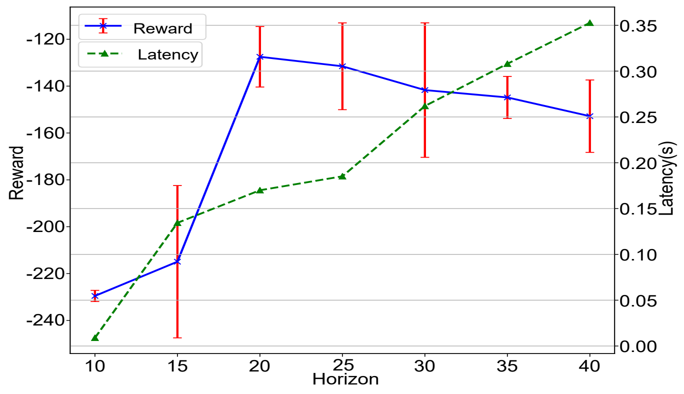


Figure 3.11: Horizon of MPPI Controller.

Table 3.1: Thermal Comfort Statistical Results for Rule-based, Model-based, Model-free and MB^{2}C Schemes

Location	Comfort	Metric	Rule-based method		Model-based method		Model-free based method		MB^{2}C	
			January	July	January	July	January	July	January	July
Fresno	PMV	Mean	-0.36	-0.20	-0.32	-0.19	-0.11	-0.03	-0.04	0.13
		Std	0.26	0.36	0.31	0.34	0.15	0.18	0.11	0.14
		Violation rate	1.22%	1.51%	2.12%	1.71%	0	0.14%	0.40%	0.58%
Chicago	PMV	Mean	-0.17	-0.30	-0.26	-0.18	-0.25	0.07	-0.23	0.05
		Std	0.23	0.33	0.24	0.31	0.17	0.19	0.07	0.20
		Violation rate	1.20%	2.04%	1.9%	2.13%	0.95%	0	0.46%	1.23%

Energy Efficiency

Figure 3.7 depicts the energy consumption results of four control methods. The results reveal that MB^{2}C saves 10.65% and 8.23% energy on average, compared with the rule-based method and model-based DRL. Compared with model-free DRL, MB^{2}C achieves comparable performance. MB^{2}C reduces the energy consumption of HVAC by modeling the complex building dynamics accurately and finding better heating and cooling setpoints.

We can also find that for different seasons and cities, the energy consumption is different. In Fresno, the building consumes 4770.04 kWh in July which is 33.39% more energy than that in January which consumes 3576.07 kWh. The reason is that in July, the outdoor air temperature range at Fresno is 15°C $\sim$ 42°C. We have to keep cooling in daylight. However, in January, the outdoor air temperature range at Fresno -1°C $\sim$ 18°C. This means that we can use outside air that is already in best range of thermal comfort to save energy.

In Chicago, the building consumes 4300.47 kWh in January that is 6.86% more

Table 3.2: Effect of Different Network Architecture

Number of hidden layers	Energy Consumption (kWh)	Thermal Comfort (PMV)
1	4911	0.11
2	4820	0.15
3	4988	0.09
4	5032	0.08

energy than in July, because the weather is cold and the outdoor air temperature range in Chicago is -20°C ∼ 15°C. In July, the outdoor air temperature range at Merced and Chicago is similar, 15°C ∼ 42°C and 15°C ∼ 40°C respectively. But the energy consumption in Fresno is 18.53% higher than the energy in Chicago. The reason is that the average day and night temperature difference for each day is larger than Chicago.

Thermal Comfort

Table 3.1 presents the average PMV value for all five zones in January and July under Fresno and Chicago weather data. All four control methods can maintain the PMV value in the desired range (-0.7∼0.7) for most of the time. The average violation rate of model-based method is 1.97%, which is a little higher than the other three methods, because the controller tries random actions and some of the actions may lead to bad thermal comfort. MB^{2}C achieves a low average violation rate by leveraging more accurate ENN model and more effective MPPI controller.

Neural Network Architecture

To investigate the impact of different neural network architectures on energy consumption and thermal comfort, we conduct experiments using July weather data from Fresno. Four neural networks were tested, each with a different number of hidden layers: 1, 2, 3, and 4. The results of these experiments, presented in Table 3.2, demonstrate the energy consumption and thermal comfort achieved by *DRLIC* with each neural network configuration.

Analyzing the experimental outcomes, we observe that neural networks with more hidden layers generally provided better thermal comfort, as indicated by results closer to 0 on the thermal comfort scale. However, this improvement in comfort comes at the cost of higher energy consumption. The underlying reason is that *DRLIC* aims to strike a balance between energy consumption and thermal comfort, recognizing that increased energy usage can enhance people's perceived comfort.

Considering these findings, we select a neural network with 2 hidden layers for *DRLIC*. This choice is motivated by its ability to minimize energy consumption while still meeting the requirement for thermal comfort. By striking a suitable compromise between energy efficiency and comfort, *DRLIC* demonstrates optimal performance with this neural network configuration.

Daily Energy Consumption for Five Zones

We analyze the daily energy consumption of MB^{2}C for five zones in July at Fresno. As shown in Figure 3.8, we record the heating energy and cooling energy for each zone per day. The top five hollow line symbols record the trend of cooling energy for five zones respectively. The bottom five solid line shows the trend of heating energy for five zones respectively. The energy spent by the third zone is higher than the other zones, because the third zone is south-oriented and the sunlight hits into that zone most of the time.

We also see that both heating and cooling occurs in some days, because the day and night temperature difference is large. In the daylight, the average outdoor temperature is 38°C, and thus we need more energy for cooling. However, at night, the average outdoor temperature is 15°C, and thus we need some heating air to meet the minimum requirement of thermal comfort (in our simulations we assume an office-like environment with students working at night sometimes).

Performance Decomposition

We implement three versions of MB^{2}C with different control methods, i.e., RS (MB_ENN_RS), CEM (MB_ENN_CEM) and MPPI control method (MB_ENN_MPPI). We also compare with the rule-based method and the existing model-based DRL method (MB_DNN_RS).

For MB_ENN_CEM, we implement (Cross-entropy method) CEM [71] controller that begins as the RS method and does this sampling for multiple iterations $m \in$

$\{0...M\}$ at each time step. The top J highest-scoring action sequences from each iteration are used to update and refine the mean and variance of the sampling distribution for the next iteration. After M iterations, the optimal heating and cooling actions are selected to be the resulting mean of the action distribution.

Figure 3.9 demonstrates the energy consumption of these four methods in two different months and at two different places (Fresno and Chicago). Compared with the rule-based method, MB_DNN_RS can only save 2.42% energy. When the building dynamics model in MB_DNN_RS changed to proposed model (MB_ENN_RS), 3.34% more energy can be saved, which illustrates the efficiency of proposed model. When we change the RS method to CEM method and MPPI method with the proposed model, 2.39% and 4.89 % more energy can be saved that illustrates efficiency of the MPPI controller.

Parameter Setting

MB^{2}C has two important parameters that may influence its performance.

The Number of Samples in the MPPI Algorithm. Figure 3.10 illustrates the performance of the MPPI controller as the number of sample trajectories is changed. We run MPPI controller with ground truth model to investigate the effect of different number of trajectories (10, 30, 100, 500, 1000, 2000, 5000, 10000). We ran 10 times to calculate the mean and standard reward for each number of trajectories. From Figure 3.10, we can see that the reward increases quickly as we increase the number of trajectories before 1000 trajectories (power of 3 in the figure). Then it increases slowly after 1000 trajectories, indicating that it is enough for the MPPI algorithm to converge. We also calculate the latency for making one action selection under different number of trajectories. we can see that the latency increases exponentially when trajectories increase. Thus we choose 1000 as the number of trajectories by considering the best reward and lower latency trade-off.

The Length of Horizon in the MPC Process. The horizon refers to the number of steps to look ahead in the MPC process. We investigate the effect of different length of Horizon H in Algorithm 2 to the performance of MPPI Controller. From Figure 3.11, we can see that the reward increases as the length of H increases and achieves the highest reward when H is 20. Then the reward decreases when we continue increasing the length of H. The reason is that small horizon results in more greedy actions that may not consider future dynamics. Large horizon produces worse actions since the prediction errors aggregate as the horizon becomes larger. We choose 20 for the horizon, which balances the prediction errors and action performance with short latency.

3.6 Discussion

Building Model Calibration. Currently, we are leveraging the existing five-zone building model in EnergyPlus to evaluate all the existing control methods. We have not done the calibration for this building model since we have no historical operation data of that building. The buildings implemented in EnergyPlus are based on first principles thermodynamical models, so we expect this model to be similar in performance to a real building. Moreover, it is reasonable to compare all the control methods based on the same building model implemented in EnergyPlus as ground truth. So, for the evaluation done in the paper, we believe this is a fair comparison to test the relative performance of different schemes for “a particular building”. If the proposed MB^{2}C was to be deployed in a real building, we would first need to learn the dynamics model from the existing historical data from a real building. Then we deploy the model in the real building for control. If we were to do simulations to test MB^{2}C before real deployment, we need to develop a calibrated EnergyPlus model that matches the target building [13, 47].

Occupancy and Weather Model. In MB^{2}C, we provide the ground-truth value of weather and occupancy for ENN dynamics model. MB^{2}C might be a bit more optimistic since we assume perfect prediction for the weather and occupancy. The errors in prediction may impact controller performance. However, we believe the performance will not significantly deviate from actual results considering model prediction errors. First is that the existing occupancy and weather prediction model [11, 45, 72, 73] show very small prediction error. Second is that MPPI controller outputs the optimal trajectory over the planning horizon. MPPI only takes the first optimal action and re-plans at the next time step based on new observations. This efficiently avoids compounding model error over time.

3.7 Conclusions

This paper proposes MB^{2}C, a novel model-based DRL HVAC control system for multi-zone buildings. We develop a new building dynamics model as an ensemble of multiple environment-conditioned neural network models. We also adopt a model predictive path integral control method to perform HVAC control. We compare the performance of MB^{2}C with the rule-based, and state-of-the-art model-based and model-free DRL schemes. The results show that MB^{2}C can achieve 10.65%, 8.23% energy savings on the former and comparable performance with the later, while maintaining (and sometimes even improving) thermal comfort of occupants. Perhaps more importantly, we can achieve this by significantly reducing the training set required by an order of magnitude (10.52$\times$ less).

Chapter 4

Deep Reinforcement Learning for Irrigation Control

Agricultural irrigation is a major consumer of freshwater. Current irrigation systems used in the field are not efficient, since they are mainly based on soil moisture sensors' measurement and growers' experience, but not future soil moisture loss. It is hard to predict soil moisture loss, as it depends on a variety of factors, such as soil texture, weather and plants' characteristics. To improve irrigation efficiency, this paper presents *DRLIC*, a deep reinforcement learning (DRL)-based irrigation system. *DRLIC* uses a neural network (DRL control agent) to learn an optimal control policy that takes both current soil moisture measurement and future soil moisture loss into account. We define an irrigation reward function that facilitates the control agent to learn from past experience. Sometimes, our DRL control agent may output an unsafe action (e.g., irrigating too much water or too little). To prevent any possible damage to plants' health, we adopt a safe mechanism that leverages a soil moisture predictor to estimate each action's performance. If it is unsafe, we will perform a relatively-conservative action instead. Finally, we develop a real-world irrigation system that is composed of sprinklers, sensing and control nodes, and a wireless network. We deploy *DRLIC* in our testbed composed of six almond trees. Through a 15-day in-field experiment, we find that *DRLIC* can save up to 9.52% of water over a widely-used irrigation scheme.

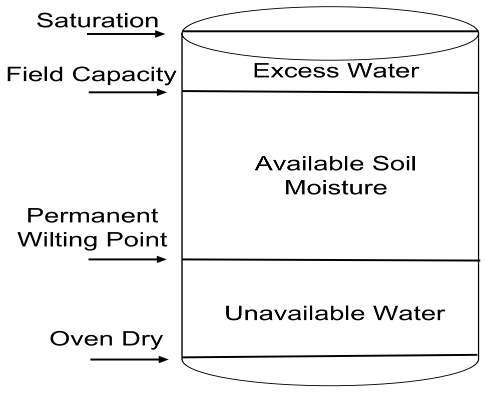


Figure 4.1: The various levels of the soil water content [1].

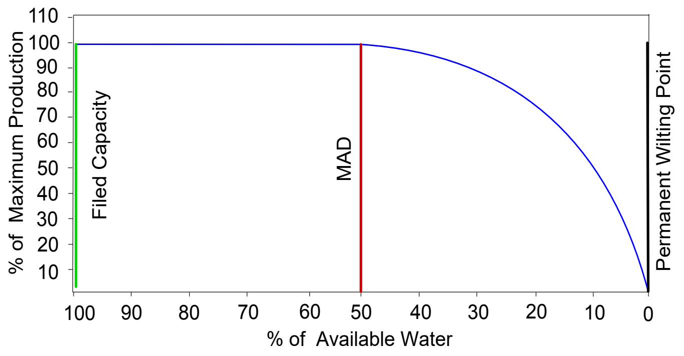


Figure 4.2: How plant production (growth) is affected by soil water content [2].

4.1 Introduction

Agriculture is a major consumer of ground and surface water in the United States, accounting for approximately 80% of the Nation's consumptive water use [74, 75] and over 90% in many Western states[1]. California's 2019 almond acreage is estimated at 1,530,000 acres, and almond irrigation is estimated to consume roughly 195.26 billion gallons per year [76, 77]. With a historic drought afflicting the Western states, it is imperative to improve the irrigation efficiency for saving our limited freshwater reserve. This work is focused on the irrigation efficiency of almond orchards.

The primary goal of agricultural irrigation is to guarantee the trees' health and maximize production. To do so, the trees' soil moisture should be maintained with a range between the Field Capacity (FC) level and the Management Allowable Depletion (MAD) level. If the soil moisture is lower than the MAD level, the almond trees will turn brown or even die. If the soil moisture is higher than the FC level, excess

[1]Irrigation Water Use: https://www.ers.usda.gov/

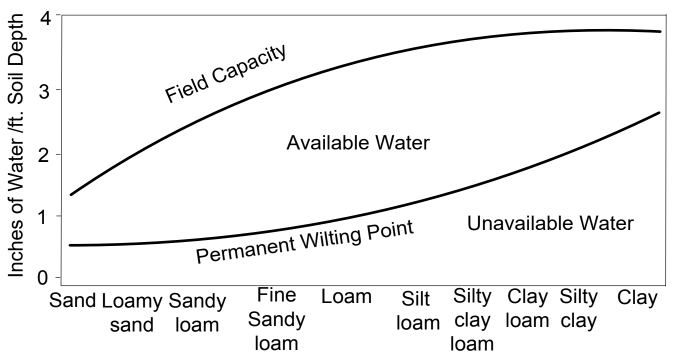


Figure 4.3: Relationship between available water capacity and soil texture [3].

water in the soil will reduce the movement of oxygen, impacting the ability of the tree to take in water and nutrients. Both FC and MAD levels can be determined by the type of plants and soil. For a specific orchard, we need to know the soil type. We can then find the FC and MAD levels for a specific soil type by referring to a manual [78].

To maintain the soil moisture between the MAD and FC range, the sprinklers need to be opened every day or several days, depending on the soil moisture change. Due to the high evaporation loss in California, daily irrigation is recommended by the Almond Board of California [78] and used in some existing irrigation systems [79, 80]. Current micro-sprinkler irrigation systems normally irrigate plants at night, since irrigating during the day causes higher evaporative water loss (14-19%) [81]. Therefore, the irrigation scheduling problem is to decide the irrigation water volume for each sprinkler to guarantee that the soil moisture will be still within the MAD and FC range at next irrigation time. The decision is based on the current soil moisture level and the predicted soil moisture loss of next day. The latter is determined by soil type, local weather, and plants' properties (e.g., the root's length and the number of leaves). The irrigation's goal is to irrigate the trees with a proper amount of water, so that the soil moisture will be still above the MAD level at the next irrigation time.

Optimal irrigation control strategies should model the soil moisture loss that will be experienced before the next irrigation time. If we have such a soil moisture prediction model, conventional Model Predictive Control (MPC) methods can be used to decide the optimal amount of water to irrigate. However, the performance of these methods relies highly on the accuracy of the soil moisture prediction model [82, 83]. It is hard to obtain an accurate model for an almond orchard, because the soil moisture is affected by many factors, including soil type, topography and surrounding environment (e.g., ambient temperature, humidity, and solar radiation intensity), and internal transpiration from plants [84]. In addition, customized soil moisture models are required for different orchards, limiting the scalability of MPC-based methods. Due to the above two limitations, MPC-based methods have not been used in orchards.

The irrigation systems currently used in orchards are ET-based or sensor-based control methods. Evapotranspiration (ET) is an estimate of moisture lost from soil, subject to weather factors such as wind, temperature, humidity, and solar irradiance. All these weather factors are being measured by weather stations. Local ET value is also publicly available [85] and updated every hour. Based on the ET values since the last irrigation time, ET-based irrigation controllers start the sprinklers to compensate for the soil moisture loss. However, they do not consider the soil moisture loss of next day before the next irrigation time. If the soil moisture loss in the last day does not equal the soil moisture loss that will happen in the next day, ET-based irrigation may under-irrigate or over-irrigate. In addition, a safe margin of water [86] is normally added, making ET-based methods over-irrigate in most cases [79].

With accurate soil moisture sensors, irrigation controllers can react directly to the soil moisture level [79]. The commonly-used controllers are "rule-based", in which a certain amount of water will be supplied once soil moisture deficiency is detected. However, parameters for the time and the amount to irrigate are generally tuned by growers by their experience. Without predicting how much water will be lost, sensor-based irrigation normally does not systematically take into account future weather information, such as rain and wind in next day.

To solve the limitations of the above existing irrigation schemes, we develop *DRLIC*, a practical Deep Reinforcement Learning (DRL)-based irrigation system, which automatically learns an optimal irrigation control policy by exploring different control actions. In *DRLIC*, a control agent observes the *state* of the environment, and chooses an *action* based on a control policy. After applying the action, the environment transits to a next state and the agent receives a *reward* to its *action*. The goal of learning is to maximize the expected cumulative discounted reward. *DRLIC*'s control agent uses a neural network to learn its control policy. The neural network maps "raw" observations to the irrigation decision for the next day. The state includes the weather information (e.g., ET and Precipitation) of today and next day.

To minimize the irrigation water consumption while not impacting the trees' health, we design a reward function that considers three specific situations. If the soil moisture result is higher than the FC level or lower than the MAD level, we will give the control agent a negative reward. If the soil moisture result is within the MAD and FC range, we will give the control agent a positive reward inversely proportional to the water consumption.

Ideally, *DRLIC*'s control agent should be trained in a real orchard of almond trees. However, due to the long irrigation interval (one day in our case), the control agent can only explore 365 control actions per year. It will take 384 years to train a converged control agent. Therefore, to speed up the training process, we train our control agent in a customized soil-water simulator. The simulator is calibrated by the 2-month soil moisture data of six almond trees and can generate sufficient training data for *DRLIC* using 10-year weather data.

Working as an irrigation controller in the field, the control agent may meet some states that it has not seen during training, especially for the control agent trained in a simulated environment. In this situation, the control agent may make a poor decision that violates plants' health, i.e., making the soil moisture level lower than the MAD level or higher than the FC level. To handle the gap between the simulated environment and the real orchard, we design a safe irrigation mechanism. If *DRLIC*'s control agent outputs an unwise action, instead of executing that action, we use the ET-based method to generate another action. We use the soil moisture model of our soil-water simulator to verify whether an action is safe or not.

To evaluate the performance of *DRLIC*, we build an irrigation testbed with micro-sprinklers currently used in almond orchards. Six almond trees are planted in two raise-beds. Each tree has a sensing and control node, composed of an independently-controllable micro-sprinkler and a soil moisture measurement set (two sensors deployed at different depths in the soil). Each node can send its sensing data to our server via IEEE 802.15.4 wireless transmission, and receive irrigation commands from the server.

We have deployed our testbed in the field and collected soil moisture data from six sensing and control nodes for more than three months. We use 2-month data to train our soil moisture simulator and 0.5-month data to validate its accuracy. After training *DRLIC*'s control agent, we have deployed the controller in our testbed for 15 days. Experiment results demonstrate that *DRLIC* can reduce the water usage by 9.52% over the ET-based control method, without damaging the almond tree health.

We summarize the main contributions of this paper as follows:

- We design *DRLIC*, a DRL-based irrigation method for agricultural water usage saving.
- A set of techniques have been proposed to transform *DRLIC* into a practical irrigation system, including our customized design of DRL states and reward for optimal irrigation, a validated soil moisture simulator for fast DRL training, and a safe irrigation module.
- We build an irrigation testbed with customized sensing and actuation nodes, and six almond trees.
- Extensive experiments in our testbed show the effectiveness of *DRLIC*.

4.2 Irrigation Problem

Soil Water Content Parameters. Soil is a plant's water reservoir. Water can fill up to 35% of the space in soil. Soil water content is the amount of water in the soil, which is often measured as a percentage of water by volume (%) or by inches of water per foot of root (in/ft). Soil moisture sensors are used to measure the soil water content (%) at one location in the soil. For a tree with a root of several feet, multiple soil moisture sensors may be deployed in different depths along with the root. The root is divided into a certain number of pieces. A soil moisture sensor is deployed at the middle point of each piece. The soil water content of the tree can be calculated as $V = \sum_{j=1}^{M} \varphi_j * d_j$, where M is the number of moisture sensors installed at different depths (M is 2 in our experiments); φ_j is the reading measured by the jth soil moisture sensor; and d_j is the depth that the jth moisture sensor covers. If such a set of soil moisture sensors are used to measure the soil water content of a region, they will be deployed under a typical tree that has similar soil water content with most of the trees in the region.

A healthy plant's root must be within a sufficient supply of water. Figure 4.1 shows two critical levels of soil water content for plants' health [1]. 1) If the soil water content is below the Permanent Wilting Point (PWP), plants cannot suck necessary moisture out of the soil. Keeping soil below the PWP level for an extended period of time will cause plants to wilt or eventually die. 2) If the soil water content of a tree is above Field Capacity (FC), the soil has an over-abundance of water, which will cause water waste and rotting of the root over time (impacting the trees' health). Therefore, *the goal of irrigation systems is to maintain soil water content between the PWP level and the FC level.*

For fruit trees like almond, production is the major goal of irrigation. To maximize the production, we need to maintain the soil water content above the Management Allowable Depletion (MAD) level, instead of the PWP level. Figure 4.2 depicts the relationship between soil water content and plant production for almond trees [2]. The curve and the MAD level may be different for different fruits. From Figure 4.2, we can see that the MAD level for almond trees is the median value (50%) between the FC level and the PWP level. Therefore, *almond trees can achieve their maximum production, as long as we maintain the soil water content above the MAD level.*

How to Determine these Parameters in an Orchard? The soil water content range between the FC level and the PWP level is the Available Water holding Capacity (AWC) of the soil. As shown in Figure 4.3, different soil types have different AWCs [3]. The soil's AWC may be affected by its texture, presence and abundance of rock fragments, and its depth and layers. The soil's AWC increases as it becomes finer-textured from sands to loam [3], and the soil's AWC decreases as it contains more clay from loam to clay [3].

The AWC of a tree, V_{awc}, can be calculated as $V_{awc} = \sigma_{awc} * D_{foot}$, where σ_{awc} is the soil's AWC and D_{foot} is the tree's root depth in the unit of feet. The AWC for different soil types, σ_{awc}, can be found in [3].

The PWP level for a soil type, V_{pwp}, can also be calculated as $V_{pwp} = \varphi_{pwp} * D_{inch}$, where φ_{pwp} is the soil moisture content at the wilting point of that soil type and D_{inch} are the root depth of the plant in the unit of inches. φ_{pwp} for a specific soil type can be found in [3].

Based on the above two parameter (V_{awc} and V_{pwp}), we can also obtain the FC level as $V_{fc} = V_{awc} + V_{pwp}$, and the MAD level as $V_{mad} = \alpha * V_{awc} + V_{pwp}$, where α is set to 50% for almond trees.

How to Use these Parameters for Irrigation? The goal of irrigation is to maintain the soil water content of plants between the FC level and the MAD level. To correctly set an irrigation system, we need to know the soil' AWC in the orchard and the PWP level (V_{awc} and V_{pwp}). We can determine these two parameters based on the above method, as long as we know the soil type. If the orchard is large, the soil type varies in space and these two parameters change too. We need to adapt the setting of these two parameters in the irrigation system accordingly.

How Many Valves to Control in an Orchard? Ideally, the sprinkler for each tree should be individually controlled, since the ET of each tree in an orchard varies from 0.12 to 0.20 inches [87]. Moreover, the soil type also varies spatially in an orchard [78], e.g., there are 10 soil type differences with soil clay loam accounting for from 45.6% to 54.7% and 0 to 8 percent slopes in a 60-acre orchard of California [2]. However, there are around 75-125 almond trees in one acre, it is costly to deploy a soil moisture sensor under each tree. Thus, an orchard is normally divided into several irrigation regions based on the similarity of soil texture. A valve is used in each irrigation region to control all the sprinklers. The irrigation problem of a large orchard is to control a number of valves. This paper is focused on irrigation scheduling, but not field partitioning. A simple way to partition an orchard into several irrigation regions is to survey the soil samples across the orchard using an auger. Growers normally conduct the survey for other purposes too, such as planning the density of trees and fertilizing the trees.

4.3 *DRLIC* System Design

In this section, we first give an overview of *DRLIC* [80, 88, 89]. We model the irrigation problem as a Markov decision process. We design a DRL-based irrigation scheme and a safe irrigation module.

[2]Soil Map: https://casoilresource.lawr.ucdavis.edu/gmap/

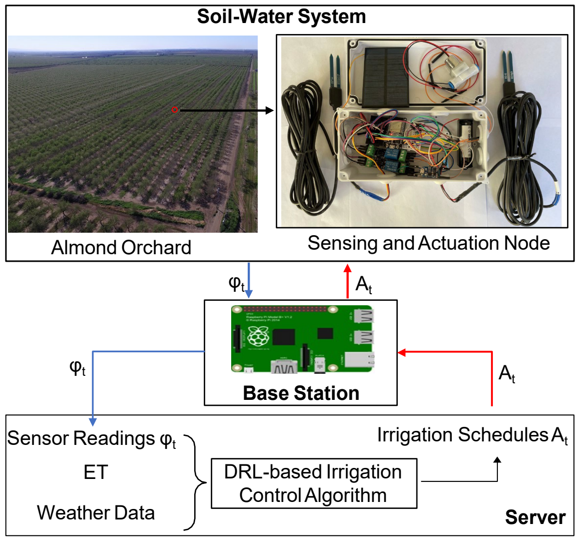


Figure 4.4: *DRLIC* System Architecture.

4.3.1 Overview

Figure 4.4 shows the system architecture of *DRLIC*, which is composed of two key components, i.e., a wireless network of sensing and actuation sprinkler nodes, and a DRL-based control algorithm.

For an almond orchard, we install the sensing and actuation node for each irrigation region. One sensing and actuation node is equipped with a set of soil moisture sensors that are deployed at different depths in the soil. Sensing data is transmitted to the base station via an IEEE 802.15.4 network. The *Base Station* collects the data from *DRLIC* nodes and sends them to a local server using Wi-Fi. These sensing data collected from all *DRLIC* nodes creates a "snapshot" of the soil moisture readings φ_t across the entire orchard.

On the server, the DRL-based irrigation control agent makes irrigation decisions based on the soil moisture sensors' readings, ET and weather data from local weather stations. It provides the optimal irrigation schedule for all *DRLIC* nodes. The objective of *DRLIC* is to minimize the total irrigation water consumption while meeting the requirement of almond health. The server will send the generated irrigation schedules A_t to all *DRLIC* nodes. By receiving a command, a node may open its sprinkler by a latching solenoid with two relays. The implementation details of the nodes will be introduced in Section 4.4.

4.3.2 MDP and DRL for Irrigation

We adopt the daily irrigation scheme, i.e., the irrigation starts at 11 PM every day. Each time, the controller decides how long to open each sprinkler to guarantee that the soil water content will be still within the MAD and FC range tomorrow night. The future soil water content is determined by the current soil water content, the irrigated water volume, the trees' water absorption, and soil water loss (caused by runoff, percolation and ET). Such a sequential decision-making problem can be formulated as a Markov Decision Process (MDP), modeled as ¡S, A, T, R¿, where

- S is a finite set of states, which includes sensed moisture level from orchard and weather data from local station.
- A is a finite set of irrigation actions for all control valves.
- T is the state transition function defined as $T : S \times A \rightarrow S$. The soil water content at next time step is determined by current soil water content and the irrigation action.
- R is the reward function defined as $S \times A \rightarrow \mathrm{R}$, which qualifies the performance of a control action.

Based on the above MDP-based irrigation problem formulation, we will find an optimal control policy $\pi(s)^* : S \rightarrow A$, which maximizes the accumulative reward R. We cannot apply conventional tools (e.g., dynamic programming) to search for the optimal control policy, because the state transition function is hard to analytically characterize. In this paper, we consider an RL-based approach to generating irrigation control algorithms. Unlike previous approaches that use pre-defined rules in heuristic algorithms, our approach will learn an irrigation policy from observations.

DRL is a data-driven learning method. It has been widely applied in many control applications [90, 57, 59, 91, 47]. DRL learns an optimal control policy through interacting with the environment. At each time step t, the control agent selects an action $A_t = a$, given the current state $S_t = s$, based on its policy π_θ.

$$a \sim \pi_\theta(a|s) = \mathbb{P}(A_t|S_t = s; \theta) \tag{4.1}$$

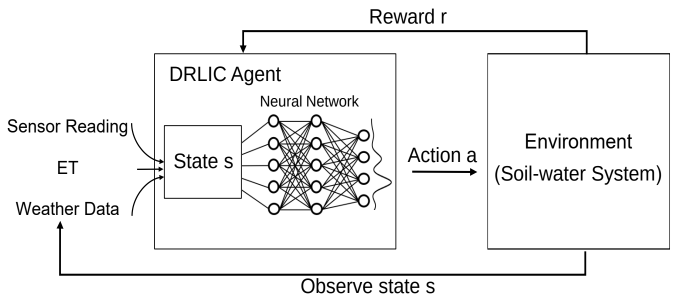


Figure 4.5: Deep Reinforcement Learning in *DRLIC*.

In DRL, the control policy is approximated by a neural network parameterized by θ [92]. When the control agent takes the action a, a state transition $S_{t+1} = s'$ occurs based the system dynamics f_θ (Equation 4.2), and the control agent receives a reward $R_{t+1} = r$.

$$s' \sim f_\theta(s, a) = \mathbb{P}(S_{t+1} | S_t = s, A_t = a) \tag{4.2}$$

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \, \mathbb{E}_{\pi_\theta}[r] \tag{4.3}$$

Due to the Markov property, both reward and state transition depend only on the previous state. DRL then finds a policy π_θ that maximizes the expected reward (Equation 4.3).

4.3.3 Deep Reinforcement Learning in *DRLIC*

Figure 4.5 summarizes the DRL architecture of *DRLIC*. The irrigation control policy (*DRLIC* Agent) is derived from training a neural network. The agent takes a set of information as input, including current soil water content, today's weather data (e.g., ET and precipitation), and the predicted weather data for tomorrow. Based on the input, the agent outputs the best action, i.e., the amount of water to irrigate. Until the next day at 11 PM, the resulting soil water content is observed and passed back to the agent to calculate a reward. The agent uses the reward to update the parameters of the neural network for better irrigation control performance. Next, we introduce the design of each DRLIC component.

State in ***DRLIC***

The state in our irrigation MDP model contains the information of three parts. (a) Sensed state, which is the soil water content measured by *DRLIC* nodes. (b) Weather-related state, which includes the current and predicted state variables from weather station. (c) Time-related state, which is about date information.

Sensed State. The soil water content of each irrigation region, calculated by Equation 4.6 using sensor reading φ from *DRLIC* node.

Weather-related State. It is a vector containing the weather information of current day and next day: ET (inch), Precipitation (inch), maximum, average, minimum Temperature (°F), maximum, average, minimum Humidity (%), average Solar Radiation (Ly/day), average Wind Speed (mph), Predicted ET by Equation 4.16 (inch), and forecasted Precipitation (inch) from local weather station.

Time-related State. Date including the month. The soil moisture may vary in different months.

Action in ***DRLIC***

Based on the current state outlined above, our irrigation scheduling is to find the best amount of water to irrigate (inch), which can maintain plant health (or maximize production) with minimum water consumption. The action is a vector that contains the water amount to irrigate for each irrigation region in an orchard. When the agent outputs an action, we will convert the amount of irrigation water to the open time duration (td) td_i for ith micro-sprinkler. It is calculated as $td_i = a_i/I$, where I is the irrigation rate. We set I to 0.018 inch/min according to the specifications of the micro-sprinklers used in our testbed.

Reward in DRLIC

We define the reward function to express our objective of achieving good plant health with minimum water consumption. Both plant health and water consumption should be incorporated in the reward function. As we know from Section 4.2, to achieve the maximum production of almond trees, we need to maintain the soil water content between the MAD level and FC level. We use the soil water content deviation from these two levels as a proxy for plant health.

To minimize water consumption while not affecting the plant health, we consider three situations in the design of the reward, as shown in Equation 4.5. First, when the soil water content (V_i) for ith irrigation region is higher than the FC (V_{fc}) level, the irrigated water is more than the plants' need. In this case, the plants' health is affected by over-irrigated water, and water consumption is too high. Second, when V_i is between V_{fc} and V_{mad}, the plants are in good health. In this case, we strive to maintain the V_i close to V_{mad} to save water, so we give a reward inversely proportional to the water consumption. Third, when V_i is lower than V_{mad}, the plants are under water stress. The plants' health is significantly impacted, proportional to the distance between V_i and V_{mad}.

By considering the above three situations, our reward function is defined as follows:

$$R = -\sum_{i=1}^{N} R_i \tag{4.4}$$

$$R_i = \begin{cases} \lambda_1 * (V_i - V_{fc}) + \mu_1 * a_i, & V_i > V_{fc} \\ \mu_2 * a_i, & V_{fc} > V_i > V_{mad} \\ \lambda_3 * (V_{mad} - V_i) + \mu_3 * a_i, & V_i < V_{mad} \end{cases} \tag{4.5}$$

$$V = \textstyle\sum_{j=1}^{M} \varphi_j * d_j \tag{4.6}$$

$$V_{mad} = \alpha * V_{awc} + V_{pwp} \tag{4.7}$$

$$V_{fc} = V_{awc} + V_{pwp} \tag{4.8}$$

$$V_{pwp} = \varphi_{pwp} * D_{inch} \tag{4.9}$$

$$V_{awc} = \sigma_{awc} * D_{foot} \tag{4.10}$$

where N is the number of irrigation regions in one orchard. a is the amount of water from the RL agent. σ_{awc} and φ_{pwp} are set by referring to the manual of California Almond Board [78] based on our specific soil type in our testbed. Equations 4.6, 4.7, 4.8, 4.9 and 4.10 have been introduced in Section 4.2.

In our current implementation, the parameters of our reward function are set to the values shown in Table 4.1, based on the specifications of our testbed. The parameters in Equation 4.5 (i.e., λ_1, μ_1, μ_2, λ_3 and μ_3) are set to the best values that

Table 4.1: Parameter Setting in Reward.

Parameter	Value	Parameter	Value
λ_1	3	α	50 (%)
μ_1	8	D_{inch}, D_{foot}	23.62 inches, 1.97 (feet)
μ_2	3	d	11.81 (inches)
λ_3	10	φ_{pwp}	10 (%)
μ_3	1	σ_{awc}	2.4 (in./ft.)

provide the best rewards during training. Their values are set by grid search, which will be introduced in detail in Section 4.5. The values of these parameters in Table 4.1 confirm with our design goal of the reward function. First, when V_i is larger than V_{fc}, we give penalties due to both plants' health and water consumption ($\lambda_1 = 3$, but $\mu_1 = 8$). Second, when V_i is lower than V_{mad}, we give a higher penalty due to plants' health ($\lambda_3 = 10$, but $\mu_3 = 1$).

4.3.4 DRLIC Training

Policy Gradient Optimization

In the above DRL framework, a variety of policy gradient algorithms can be used to train the irrigation control agent. Policy gradient algorithms achieve the objective in Equation 4.3 by computing an estimate of the policy gradient and optimizing the objective through stochastic gradient ascent (Equation 4.11).

$$\theta \leftarrow \theta + \alpha \bigtriangledown_\theta \mathbb{E}_{\pi_\theta}[r] \tag{4.11}$$

In this work, we use proximal policy optimization (PPO) [62], which has been successfully applied in many applications such as navigation [93] and games [94]. PPO is known to be stable and robust to hyperparameters and network architectures [62].

PPO minimizes the loss function in Equation 4.12, which is equivalent to maximizing the Monte Carlo estimate of rewards with regularization. The advantage function $\hat{A}_t$ given by Equation 4.13 is used to estimate the relative benefit of taking an action from a given state.

$$L_{PPO}(\theta) = -\hat{\mathbb{E}}_t[min(w_t(\theta)\hat{A}_t, clip(w_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4.12}$$

$$\hat{A}_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{4.13}$$

$$w_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4.14}$$

In Equation 4.14, $\pi_\theta(a_t|s_t)$ is the policy being updated with the loss function and $\pi_{\theta_{old}}(a_t|s_t)$ is the policy that was used to collect data with environment interaction. As the data collection policy differs from the policy being updated, it introduces a distribution shift. The ratio $w_t(\theta)$ corrects for this drift using importance sampling. The ratio of two probabilities can blow up to large numbers and destabilize training, so the ratio is clipped with ϵ.

Data Collection and Preprocessing

On day t, the *DRLIC* agent observes a state s (e.g., moisture level), and then chooses an action (water amount). After applying the action, the soil-water environment's state transits to s_{t+1} next day and the agent receives a reward r. After that, a data pair (s_t, a_t, r_t, s_{t+1}) can be collected. We conduct data normalization by subtracting the mean of the states/action and dividing by the standard deviation. We use 10-year weather data (2010-2020) to generate the data pairs in our dataset, which will be used to train our *DRLIC* agent.

Training Process

Ideally, *DRLIC*'s control agent should be trained in an orchard of almond trees. A well-trained DRL agent needs 384 years to converge due to the long control interval of irrigation systems. It is impossible to train *DRLIC* agent in an orchard. A feasible solution is to refer to a high-fidelity simulator. However, there are no such simulators available in the soil-water domain. Then we decide to leverage a data-driven simulator to speed up the training process. We employ the soil water content predictor introduced in Section 4.3.5 as our soil-water simulator. The simulator allows *DRLIC* to "experience" the weather of 10 years in several minutes.

The training procedure of *DRLIC* is outlined in Algorithm 3. We train *DRLIC* using 1000 episodes and length of an episode as 30 days. For each episode, we can collect 30 training data pair (s_t, a_t, r_t, s_{t+1}) under different weather data and leverage

Algorithm 3 *DRLIC* Training Algorithm

```
Input: State s, Action a, Reward r, an initialized policy, $\pi_\theta$;
Output: A trained irrigation control agent
for i=0,..., # Episodes do
    State ← Soil-water environment
    $\theta_{old} \leftarrow \theta$
    for t = 0, ..., Steps do
        $\hat{a}_t = \pi_\theta(s_t)$  $s_{t+1}, r_{t+1} = env.step(\hat{a}_t)$
    end
    Compute $\hat{A}_t$  With minibatch of size M  $\theta \leftarrow \theta - \alpha \nabla_\theta L_{PPO}(\theta)$
end
```

Equation 4.12 to optimize the objective in Equation 4.3 through stochastic gradient ascent. The training ends once Algorithm 3 converges: at the end of each episode the total reward obtained is compared with the previous total. If the current episode reward does not change by $\pm$ 3%, we consider the policy has converged. If the policy does not converge, the training will continue up to a maximum of 100 training iterations (# episodes = 100). After the training, we will deploy the trained *DRLIC* agent into the real almond orchard.

When we are given a new environment (e.g., a new orchard), we first need to collect the real-world irrigation data of new environment by existing controller (e.g., ET-based control) to build a soil water content predictor to describe the water balance in the root zone soil. Then we leverage the soil water content predictor to speed up the training process, after that, we deploy the well-trained *DRLIC* agent for this new orchard.

4.3.5 Safe Mechanism for Irrigation

We design a safe mechanism that integrates the RL and ET controller in a coupled close-loop. Figure 4.6 illustrates the workflow of safe mechanism, with the following key elements. (i) Different from the pure RL framework, we introduce a safety moisture condition detector to evaluate whether the RL algorithm outputs a safe action. (ii) If so, the action goes to the RL agent, who will be in charge of irrigation control. (iii) Otherwise, we will use an ET-based controller to generate an action for that control cycle. (iv) *DRLIC* will the RL agent for the future control cycles. We now introduce the soil water content predictor and safety condition detector.

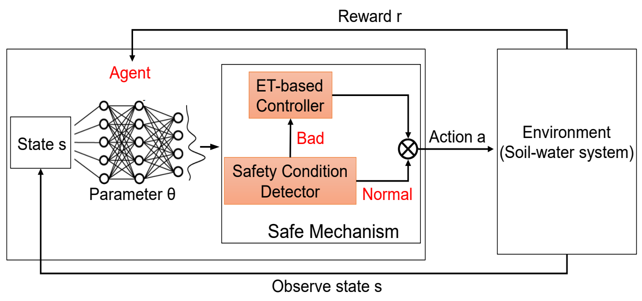


Figure 4.6: Deep Reinforcement Learning with Safe Mechanism.

Soil Water Content Predictor. To enable early detection of an unsafe action, we design a soil water content predictor to predict the moisture trend after taking an action. Then we design a safe condition detector to detect almond health penalty $p(t)$. The idea is to detect whether the damage metric for an almond tree is higher than a threshold. If so, the detector will command *DRLIC* to switch from RL to ET-based controller.

We design a soil water content predictor to describe the water balance in the root zone soil. The variations of water storage in the soil are caused by both inflows (irrigation and precipitation) and outflows (evapotranspiration). This leads to the following mathematical expression:

$$V_{i,t+1} = c_1 * V_{i,t} + c_2 * (A_{i,t} + P_t) + c_3 * E_t + b \tag{4.15}$$

$$E_t = \Gamma_c * RA * TD^{(1/2)} * (T_t + 17.8°C) \tag{4.16}$$

where $V_{i,t+1}$ denotes the predicted moisture level in the root zone for ith irrigation region after taking the action from RL, E_t and P_t are the plants' ET and the measured rainfall. In time period t, and $A_{i,t}$ is the irrigation amount for ith irrigation region. c_1, c_2, and c_3 are coefficients. It is assumed in this work that runoff and water percolation are proportional to soil moisture level [95, 96, 97] in Equation 4.15. All the coefficients can be determined by means of system identification techniques [98]. All variables are normally expressed in inches.

The weather data can be get from local weather station. For ET, we adopt the simple calculation model established in [99]. As shown in Equation 4.16, where Γ_c is a crop-specific parameter. RA stands for extraterrestrial radiation, which is in the same unit as E_t. TD denotes the annual average daily temperature difference,

Table 4.2: Coefficients of Predictor for Each Tree.

	c1	**c2**	**c3**	**b**	**R^2**	**NRMSE**
Tree1	0.973	0.288	-0.103	0.003	0.982	0.062
Tree2	0.937	0.325	-0.121	0.013	0.985	0.071

which can be derived from local meteorological data, and T_t is the average outdoor temperature during the tth time period.

Safety Condition Detector. We employ the difference between predicted moisture level and lower bound as a detector to estimate the almond tree damage. As explained in Section 4.2, MAD is the lower bound. Then we use $\sum_{i=1}^{N}(V_{mad} - V_{i,t+1})$ as a safety condition detector, $V_{i,t+1}$ denotes the predicted moisture level from ith irrigation region for t timestep. V_{mad} is the water content lower bound. *DRLIC* will evoke ET-based controller once safety condition detector detects the dangerous irrigation action.

Parameter Learning of our Soil Water Content Predictor. We leverage the designed testbed to collect the irrigation amount of almond trees for 2 months. The ET value for each day is collected from a local weather station [85] and the moisture level for each tree is collected by the designed *DRLIC* node. Then the linear least square method was applied to estimate the coefficients. R^2 is used to explain the strength of the relationship between the moisture level and related factors. Normalized root-mean-square error (NRMSE) is used as a goodness-of-fit measure for predictors. The results are shown in Table 4.2, we can see that the R^2 is close to 1 indicating that the irrigation, ET and precipitation have a strong relationship with soil water content for the tree. The NRMSE is less than 0.1 which means that the predictor can achieve accurate prediction for soil water content.

4.4 Testbed and Hardware

4.4.1 Testbed and Microsprinkler Description

Figure 4.7 shows our micro-sprinkler irrigation testbed. The micro-sprinkler irrigation system is installed and designed to be identical in hardware, micro-sprinkler coverage, etc. This irrigation system measures 290 cm x 160 cm, with micro-sprinklers arranged in a 3x2 grid, each 97cm from the next. The micro-sprinklers chosen were

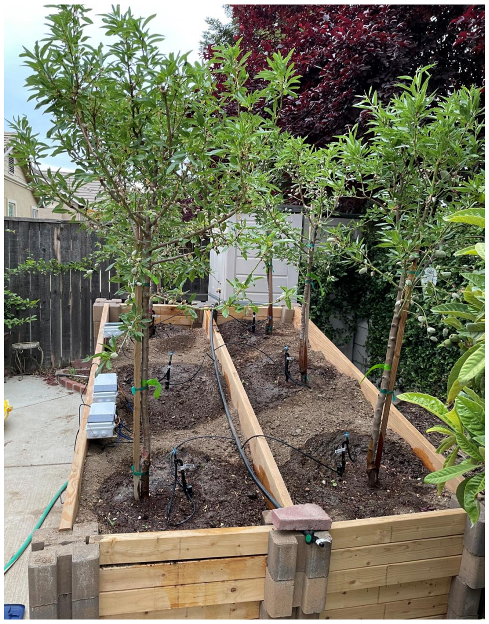


Figure 4.7: Testbed and Microsprinkler Irrigation System.

1/4 ’, 360 ° pattern by Rainbird, which are currently considered state-of-the-art in micro-sprinkler technology. Six all-in-one young almond trees were planted into the testbed (three for each). The average height is 2 meters. The soil with 2.7 m ^{3} volume is collected from a local orchard that is a typical loam soil and the plant-available water-holding capacity is 2.4 inches of water per foot.

4.4.2 *DRLIC* Node Development.

The designed DRLIC node in Figure 4.4 consists of four main parts: sensors, actuator, power supply and transmission module.

Sensors: It consists of several moisture sensors for different depths. The moisture sensors vary in their sensitivity and their volume of soil measured. Each moisture sensor for 12-inch depth provides accurate quantitative soil moisture assessment following the Almond Board Irrigation Improvement Continuum [78]. We assign 2 moisture sensors for each *DRLIC* node since the depth of root zone of the almonds in our testbed is 24 inches.

A key feature of the *DRLIC* node is the ability to measure the volumetric water content in the surrounding soil. We opted to purchase research-quality Decagon EC-

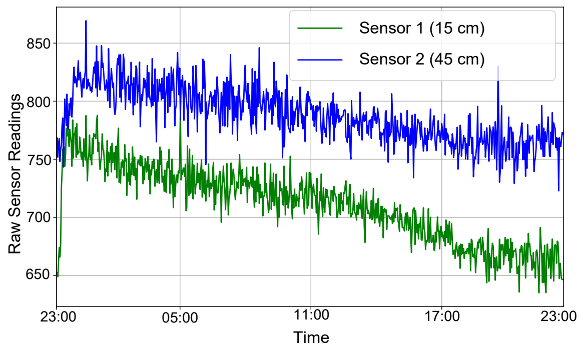
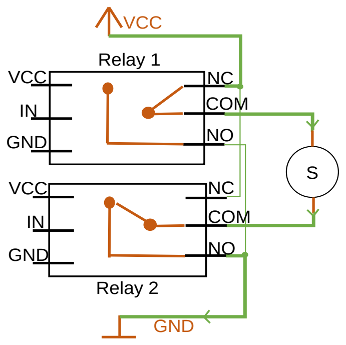
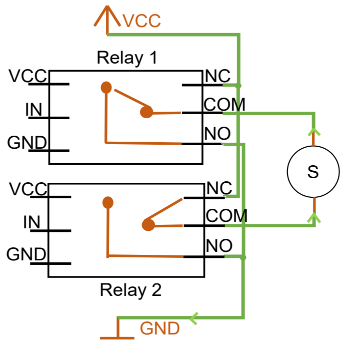


Figure 4.8: Daily Soil Moisture Readings.



(a) Positive Current Pulse.



(b) Negative Current Pulse.

Figure 4.9: On and off Circuit Diagram for Latching Solenoid.

5 sensors [3], with a reported accuracy of ±3%. Raw sensor readings collected over a period of one day with a high sampling frequency can be seen in Figure 4.8. The sensors report the dielectric constant of the soil, which is an electrical property highly dependent on the volumetric water content (VWC).

$$\varphi(m^3/m^3) = 9.92 * 10^{-4} * raw\_reading - 0.45 \tag{4.17}$$

A linear calibration Function 4.17 above provided by the sensor manufacturer is used to convert the raw readings to VWC. The range of φ is between 0% and 100%. φ of saturated soils is generally 40% to 60% depending on the soil type.

Actuator: It consists of a latching solenoid with two relays. A standard solenoid requires constant power to allow water to flow, making it a poor choice for a battery-powered system. The nine-volt performance all-purpose alkaline batteries from Amazon can only continue to power the standard 12V DC solenoid for 8 hours. To extend

[3]Decagon devices. http://www.decagon.com/products/soils/

DRLIC node lifetime, we chose to use a latching solenoid for micro-sprinkler actuation, requiring only a 25ms pulse of positive (to open) or negative (to close) voltage. The h-bridge is usually used to produce a bi-directional current to control the latching solenoid [100]. However, it needs a special design to meet different voltages requirements for the ESP32 and latching solenoid.

In order to control the latching solenoid, we design a circuit diagram using two relays to operate with a very little connection overhead. A relay is an electrically operated switch. Figure 4.9 shows the turn-on and off circuit diagram for latching solenoid. When both the relays are off, there is no current going through the solenoid (S). Initially, both the relays are in a normally closed (NC) position. To turn the solenoid on, Relay 1 is switched from NC to normally open(NO) for 25ms, providing the positive current pulse through the solenoid. The current path shown in Figure 4.9(a) is: $VCC -> NC_1 -> COM_1 -> S -> COM_2 -> NO_2 -> GND$. To turn the solenoid off, Relay 2 is switched from NC to NO for 25ms, de-latching the solenoid to the closed position. The current path shown in Figure 4.9(b) is: $VCC -> NC_2 -> COM_2 -> S -> COM_1 -> NO_1 -> GND$. To prevent over-irrigation in the event of a power failure, we have the power supply module to continuously provide the power.

Power Supply: Power supply consisted of a 5v, 1.2W solar panel for energy-harvesting and a 18650 Lithium Li-ion battery with a capacity of 3.7V 3000 MAH for energy storage. The TP4056 lithium battery charger module comes with circuit protection and prevents battery over-voltage and reverse polarity connection. All sensors (1 ESP32, 2 moisture sensors, 2 relays and 1 latching solenoid) are powered with this power supply module. It can provide continuous power to prevent over-irrigation in the event of a power failure for the actuator module.

Transmission Module: Transmission includes uplink and downlink. In the uplink path, the moisture sensor readings from the field are sampled by the ESP32, a low-cost, low-power system on a chip (SoC) series with Wi-Fi capability. The readings are then sent from ESP32 to the base station as input for the optimal control. In downlink path, the control command calculated by the DRL agent will be routed to all ESP32 to turn on or off the solenoids.

4.5 Implementation

In this section, we illustrate in detail the implementation of *DRLIC* and tuning hyper-parameters.

***DRLIC* Implementation Details** We implement *DRLIC* in Python using widely available open-source frameworks, including Pandas, Scikit-learn and Numpy. The control scheme - *DRLIC* is implemented using the scalable deep reinforcement learning framework, RLlib [101]. RLlib supports TensorFlow, TensorFlow Eager, and PyTorch. RLlib provides multi-ways for us to customize the training process of the target environment modeling, neural network modeling, action set building and distribution, and optimal policy learning. The 10-year weather data (2010-2020) are collected for *DRLIC*, with 9 years used for training and the remaining 1 year used for testing. In our implementation of *DRLIC*, we use the Adam optimizer for gradient-based optimization with a learning rate of 0.01. The discount factor is 0.99. The neural network model has 2 hidden layers with 256 neurons for each. The local server for training and running *DRLIC* is a 64-bit quad-core Intel Core i5-7400 CPU at 3.00 GHz that runs Ubuntu 18.04.

Training Details and Tuning Hyper-parameters. The performance of *DRLIC* agent is sensitive to the hyperparameter values chosen. Unfortunately, there is no simple approach that allows *DRLIC* agent to understand whether a specific value for a given parameter would improve total reward. To address this issue and further increase *DRLIC* 's performance, we leverage a tuning approach to optimize the *DRLIC* 's hyperparameters, such as λ, μ associated with rewards and penalties, and the learning rates. In particular, we employ grid search which allows us to specify the range of values to be considered for each hyper-parameter. The grid search process constructs and evaluates our model using every combination of the hyper-parameters. Finally, we employ cross-validation to evaluate each learned model.

4.6 Evaluation

In this section, we evaluate the performance of *DRLIC* in the field. We evaluate *DRLIC* system for 15 days in the real world.

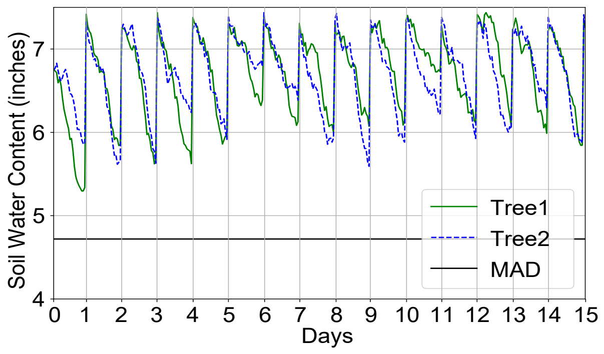


Figure 4.10: ET-based Method.

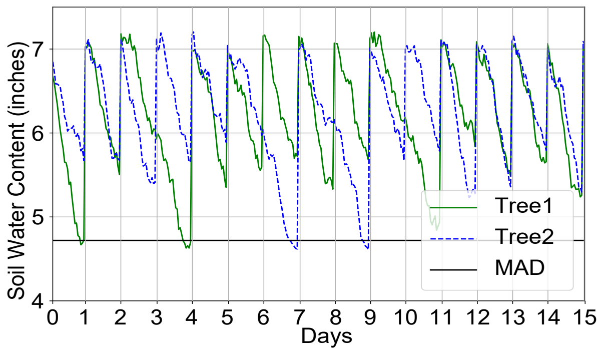


Figure 4.11: Sensor-based Method.

4.6.1 Experiment Setting

Baseline Strategy:

We compare *DRLIC* to two state-of-the-art irrigation control schemes introduced in Section 4.7.

ET-Based Irrigation Control [78]. To implement an ET-based controller, we query a local weather station for the previous day's ET loss. To compensate for the loss, we use the sprinkler's irrigation rate provided by its dataset to calculate how long the system should be activated for irrigation.

Sensor-based Irrigation Control [79]. The sensor-based controller has two thresholds, the lower and upper soil water content levels. The first is set at 4.96 inches, 10% higher than MAD to avoid the under irrigation occurring prior to the wetting front arriving at the sensor depth. The latter is set to 6.97 inches, 5% below FC to allow for some rainfall storage. We carefully set these two thresholds based on the soil environment of our testbed.

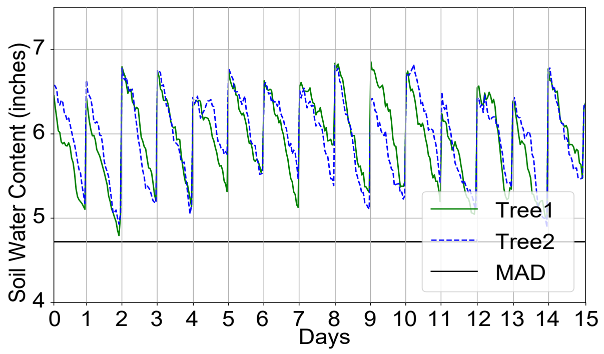


Figure 4.12: DRLIC.

Performance Metrics

We evaluate the performance of *DRLIC* and two baseline systems in terms of two performance metrics.

Quality of Service. Although the irrigation system has no control over solar exposure and soil nutrients, it has direct control over the moisture levels in the soil. For this reason, our primary metric for irrigation quality is the system's ability to maintain soil moisture above this MAD threshold at all times at all of our measured locations. By doing so, we are guaranteeing that the plant has sufficient moisture to be healthy and no production loss. In this paper, we call this the quality of service of the irrigation system.

Water Consumption. As each sprinkler uses a water supply and we directly control the times at which each micro-sprinkler is active, we can monitor the amount of water consumed by these three systems at all times to determine the efficiency of each system. Thus another metric is the water consumption, which we would like to minimize subject to the quality of service constraints.

Experiments in our Testbed

We validate the *DRLIC* system with baselines in real-world deployment in terms of plant health and water consumption for 15 days. In the case study, we have six almond trees in our testbed as shown in Figure 4.7. *DRLIC*, sensor-based control and ET-based control are used to irrigate the upper, middle and lower two trees separately since there is no runoff between trees in our testbed. To allow three irrigation systems to operate independently, Every micro-sprinkler is controlled by a *DRLIC* node. In

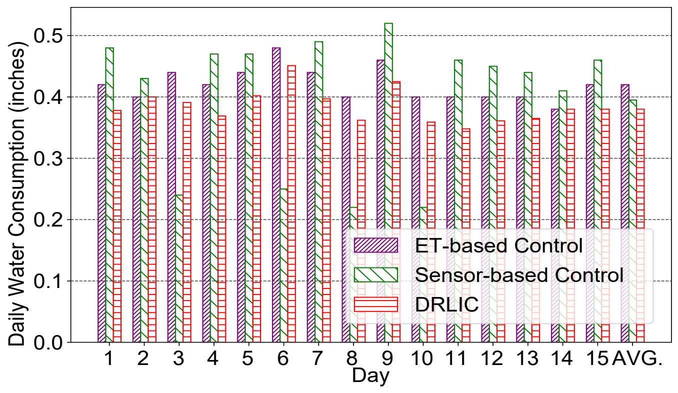


Figure 4.13: Daily Water Consumption.

this way, the only difference among the three systems is the schedules sent to the nodes.

4.6.2 Experiment Results

Quality of Service

Irrigation systems are installed to maintain almond health with no production loss. Figure 4.10, 4.11, and 4.12 shows the daily soil water content in the field for ET-based control, Sensor-based control and *DRLIC*. The black horizontal line shows the MAD level. If soil water content is below this line, tree health will be impacted. We can see that *DRLIC* and ET system can maintain the soil water content above MAD threshold during the 15 days deployment and thus meet the requirement of almond health. However, the trees irrigated by Sensor-based method are in an under-irrigation period of 18 hours for four days (day 1, 4, 7 and 9) since the soil water content of Sensor-based method is lower than the MAD. The reason is that the moisture level of previous day is close but not reached to MAD, so the sensor-based method will not irrigate even though the moisture level is in an under-irrigation trend. *DRLIC* system can irrigate what the trees need based on the learned model about the water changes in the soil and maintain the soil water content close to MAD level.

All three underlying irrigation systems begin with enough water content on the first day. We see that the soil water contents of the two trees in ET control system are much above the FC threshold. In our deployment of *DRLIC* against the ET control strategy in Figure 4.10, we see that soil water content for these two trees is different and much higher than the MAD level. This emphasizes the limitations of ET and the core of our work. The irrigated regions don't receive moisture the same way, and most of the time, the ET-based controller irrigates more water than the plant needs.

Water Consumption

When a decision must be made to switch to a new almond irrigation control system, a primary concern is the efficiency of the proposed system. The system's ability to return its investment based on increased efficiency will often dictate the acceptance of the technology. In addition, the environmental benefits of reduced freshwater consumption are clear and help promote system adoption.

In our experimental setup, the water source provided by each micro-sprinkler is pressure-regulated to the industry standard, 30 psi. Each micro-sprinkler head distributing water uses a clearly-defined amount of water per unit time, as described in the almond irrigation manual [78]. By tracking exactly when each micro-sprinkler is actuated by the system, we can determine very accurately how much water has been consumed.

Figure 4.13 shows the daily irrigation amount of two trees actuated by ET-based control, sensor-based control and *DRLIC* in a 15 days' deployment experiment. From this figure, we can see that *DRLIC* can save an average 9.52% and 3.79% of the water compared with ET-based and Sensor-based control during 15 days deployment experiment. ET-based control is a centralized control method to irrigate all almond trees without considering their specific need. Sensor-based control is water-efficient by monitoring the moisture and irrigating when the moisture level is lower than the MAD level. However, the thresholds are site-specific and not optimal. *DRLIC* can learn optimal irrigation control by interacting with the local weather and soil water dynamic environment.

4.6.3 Effect of our Safe Irrigation Mechanism.

In the 15 days' deployment, we find that there are two days (Day 2 and 14 in Figure 4.12) *DRLIC* triggers the ET-control method. This can also be validated from Figure 4.13, we can find the water consumption of ET method and *DRLIC* on days 2 and 14 are the same. We check the weather data to understand the reason and

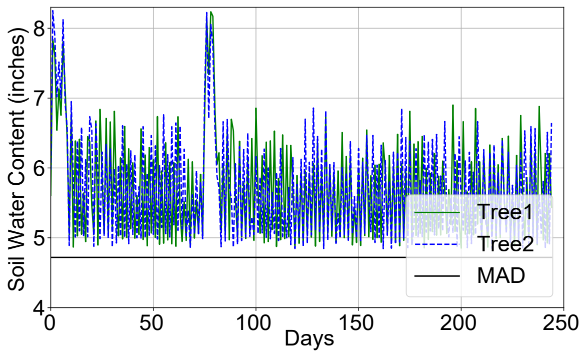


Figure 4.14: Daily Soil Water Content with Safe Mechanism.

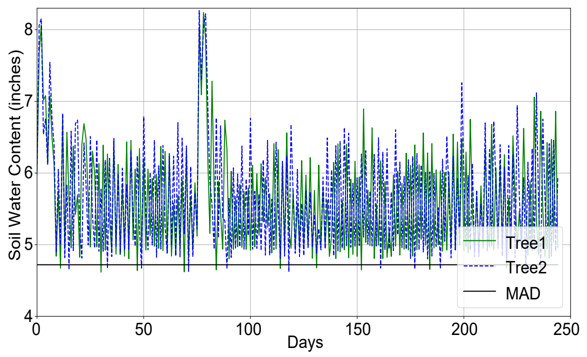


Figure 4.15: Daily Soil Water Content (w/o Safe Mechanism).

find that the wind speeds of days 2 and 14 are 7.2 and 11.9 mph receptively which is much higher than the average 2.8 mph of the other 13 days.

We now run *DRLIC* with and without safe mechanism for a whole growing season in simulation, labeled as Robust-RL and RL-only, respectively. Figure 4.14 and 4.15 show the daily soil water content of Robust-RL and RL-only for a same growing season 2020, respectively. From the almond's perspective, Robust-RL maintains health with 0 days below the MAD level. The RL-only irrigation method has 21 days below the MAD level. The reason is that the RL models trained from past weather data "misbehave" on the test weather data. while it may be possible to train on changing weather to obtain a robust policy, no offline training can ever cover all possible weather changes. The RL agent with safe mechanism from *DRLIC*, however, is robust to weather changes because the safety condition detector will detect the dangerous actions from RL agent and the ET system will take control.

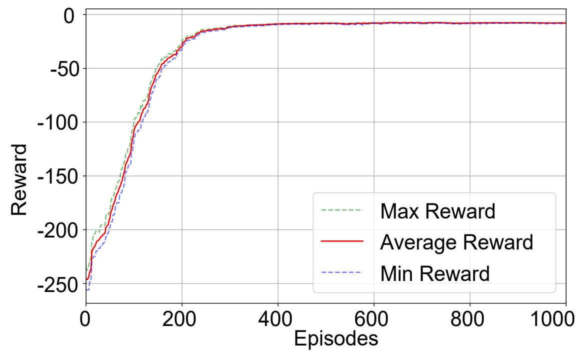


Figure 4.16: Policy Convergence of DRL.

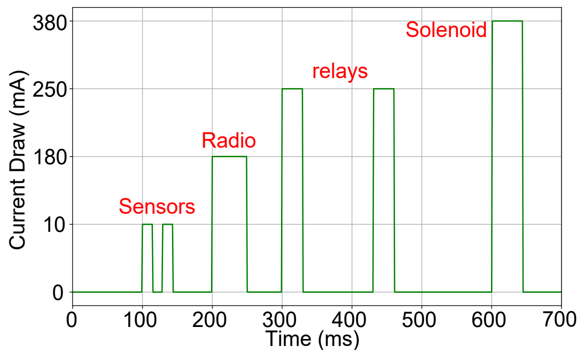


Figure 4.17: Energy Profile for Different Kinds of Sensors.

4.6.4 Effect of proposed Reward.

In this section, we discuss the simulation results of *DRLIC* with different rewards for a whole growing season (March 1st to October 31st, 246 days).

In order to minimize the water consumption while not affecting the plant health, we consider three situations in the reward. 1) The soil water content (V_i) is higher than the FC (V_{fc}) level. 2) V_i is between V_{fc} and V_{mad}. 3) V_i is lower than V_{mad}. Only in the second situation, the plants are in good health. To evaluate our reward function, we compare it with a simple reward ($DRLIC_{MAD}$) that only maintains V_i above V_{mad}. It is commonly used in the sensor-based method [79]. The reward is defined as: $R = -\sum_{i=1}^{N} \lambda_3 * (V_{mad} - V_i) + \mu_3 * a_i, V_i < V_{mad}$. This function gives more penalty to plant health when V_i lower than V_{mad} since plants' health is significantly impacted. All the parameters are the same in Section 4.3.3.

Figure 4.19 shows the water consumption of *DRLIC* with our proposed reward (*DRLIC*) and the simple reward (*DRLIC* _MAD). *DRLIC* can save 2.04% more water than *DRLIC* _MAD, as the latter does not consider the case when V_i is higher

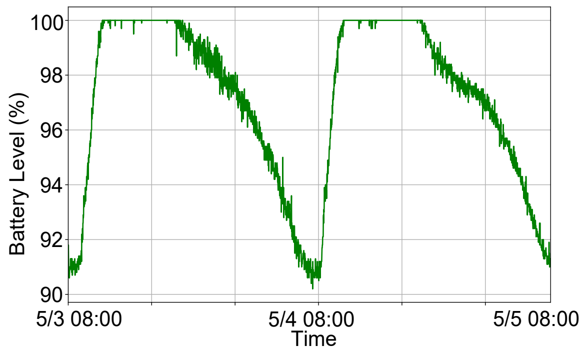


Figure 4.18: Battery Charging and Discharging Cycle.

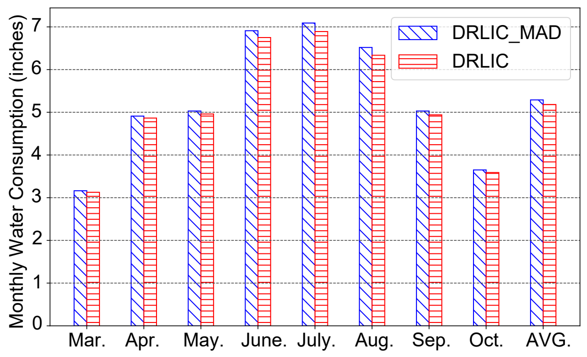


Figure 4.19: Water Consumption for *DRLIC* with Different Reward

than V_{mad}. *DRLIC* considers two more situations by giving different penalties to plants' health and water consumption. The first case is over-irrigation. The water consumption is too high. Therefore, the penalty for water consumption is higher than plant health. In the second case, the plants are in good health. *DRLIC* strives to maintain the V_i close to V_{mad} to save more water.

4.6.5 *DRLIC* Policy Convergence.

Figure 4.16 shows the RL training process and the policy converges around the 500th training iteration. We define the length of an episode as 30 days. We randomly vary the soil water content for each tree between the FC (7.08 inches) and MAD (4.72 inches) at the beginning of each episode. By doing so, the policy is exposed to different soil water content conditions and learns to avoid water depletion than the MAD level during training. At the beginning of the experiment, the RL policy receives a larger negative reward as it does not know a valid sequence of actions that maximize the reward. The policy converges at the 500th training iteration. The

Table 4.3: Micro-sprinkler Node Manufacture Cost.

Component	Price	Component	Price
Moisture Sensor x 2	\$250	ESP32	\$6.5
18650 Li-ion battery	\$3	Solar Panel	\$4.3
Latching Solenoid	\$4	Switch Relay x 2	\$5
Waterproof Enclosure	\$12	Maintenance Fee	\$10
		Total	\$294.8

whole training (i.e. 1000 training iterations) takes $\sim$ 4 hours using a 64-bit quad-core Intel Core i5-7400 CPU at 3.00 GHz.

4.6.6 Energy Consumption of Sensor Nodes

From a wireless sensor network standpoint, the ability of a system to operate for a long period of time without user intervention is fundamental. *DRLIC* nodes are no different, especially if they are meant to be put on the ground. For this reason, our hardware and software were designed to consume as little energy as possible. *DRLIC* nodes were fitted with a latching solenoid, allowing the flow of water to be turned on or off with a short pulse of power, rather than a constant supply. For additional energy savings, the radio in each node is duty-cycled, activating for only a 10 second period every 1 minute. We need this high data frequency, the reason is that the base station can send an off command to *DRLIC* with a minute granularity. In our devices, the four peripherals that consume significant energy are the two moisture sensors, solenoid, two relays and radio. To meet this energy, we design an energy harvesting mechanism by leveraging one 5/6 V 1.2 W solar panel.

Figure 4.17 shows the energy consumption for different sensors. Each moisture sensor sample requires 10 mA of power for 10 ms, and each flip of the latching solenoid requires 380 mA of power for 30ms. The ESP32 radio requires 180 mA of power for 50ms when in transmitting mode. The relay requires 250 mA for 20 ms for switching on or off. In our system, to ensure we don't cut power too early, we add a safety band of 50% on the timing on both of these devices, triggering for 15 ms and 45 ms for the sensor and solenoid, respectively. Overall, the solar-harvest mechanism can meet the daily requirement of all the sensors in *DRLIC* node.

Figure 4.18 shows the two days' energy charging and discharging process. After a night discharge, the 18650 battery level is increasing at 9:15 am on May 3rd. It usually takes 2 hours to fully charge the battery (9:15 - 11:35 am). The battery level will keep 100% from 11:35 am to 18:45 pm, the energy harvested from solar can meet the energy requirement of all sensors in *DRLIC* node. The battery will discharge from 100% at 18:45 pm of May 3rd to the 90.7% at 8:45 am of May 4th. Then the whole energy charging and discharging process repeat. The lowest battery level is an average of 90%. In the 2 week's deployment, we find that even on a cloudy day, the battery can also be charged and will take one more hour to be fully charged.

4.6.7 Return on Investment

A primary concern to purchasing or upgrading an irrigation control system is the return on investment, i.e., how long does it take to save enough money from water consumption to cover the cost of the new irrigation system. To calculate the return on investment of *DRLIC*, we take into account the initial investment cost of the *DRLIC* system and the money saved from the less water consumption provided by our increased irrigation efficiency.

We first calculate the cost to develop a single *DRLIC* node. All the components of a *DRLIC* node can be found in a consumer electronics store and a home improvement store. Table 4.3 lists the cost of all components. In total, a *DRLIC* sensing and actuation node costs $294.8. A large portion of the budget is the cost of two soil moisture sensors. We use two expensive soil moisture sensors that provide accurate measurement and a long lifetime.

The factors that mostly influence the payback of our system are water price and water volume saved by *DRLIC*. Water price varies considerably in different irrigation district and over time. This study assumed 100% groundwater usage and availability. Each tree costs $11.3 for irrigation water per month. Based on our experiment results, *DRLIC* can save 9.52% of water expense per month, corresponding to $1.08. Normally, almond orchards have 100 trees per acre. As a result, *DRLIC* can save $108 per month. Take a 60-acre almond orchard with 10 irrigation regions as an example. Each irrigation region is six acres. *DRLIC* can save $648 in each irrigation region per month.

In each irrigation region, we need to deploy one *DRLIC* node, which costs $294.8. The other irrigation components will use the existing infrastructure, such as the pipelines and micro-sprinklers under each tree. The cost of upgrading the existing irrigation system with our irrigation control system is \$294.8 for one irrigation region in an orchard. Every month, our system can save \$648. Therefore, it only needs half a month for our irrigation system to return the investment.

4.7 Related work

ET-Based Irrigation Control. As the weather is a primary water source or sinks in an irrigated space, systems have been developed to use weather as input for control. The simplest of these systems use standard fixed-schedule irrigation, but allow a precipitation sensor to override control to save water during rain. The more complicated systems, now the industry standard, use evapo-transpiration (ET), an estimate of the amount of water lost to evaporation and plant transpiration to do efficient water-loss replacement [102]. Some providers boast an average 30% reduction in water consumption, but as with all industry irrigation systems, ET-based systems are limited by centralized control, and can not provide site-specific irrigation, reducing potential system efficiency and quality of control.

Sensor-based Irrigation Control. With the introduction of more accurate and efficient soil moisture sensors [103], work has been done to create irrigation controllers that react directly to moisture levels in the soil [79]. Moisture sensors buried in the root zone of trees accurately measure the moisture level in the soil and transmit this data to the controller. The controller then adjusts the pre-programmed watering schedule as needed. There are two types of soil moisture sensor-based systems: 1) Suspended cycle irrigation systems. Suspended cycle irrigation systems use traditional timed controllers and automated watering schedules, with start times and duration. The difference is that the system will stop the next scheduled irrigation cycle when there is enough moisture in the soil. 2) Water on-demand irrigation requires no programming of irrigation duration (only start times to water). This type maintains two soil moisture thresholds. The lower one to initiate watering, and the upper one to terminate watering [79]. However, without a model of the way water is lost, these thresholds are usually set based on experience and are not optimal.

Model-based Irrigation Control. In [100], a mechanistic PDE model of moisture movement within irrigated space is built. Using this model, an optimal watering schedule can be found to maintain a proper moisture level. However, the PDE model is not updated over time and future weather prediction is not taken into account. To tackle these two limitations, the same authors further improve the control system in [86]. The PDE model is eschewed in favor of an adaptive approach that involves models trained from sensor data. Long-term and short-term models are developed to describe the relationship of runoff between sprinklers in the movement of water through the soil.

As indicated by the authors [86], their system is designed for turf irrigation, and it is unlikely to provide benefit in shrubbery or tree irrigation. First, the turf soil moisture is affected by water runoff on soil surface and the overlapping coverage of sprinklers. The models in [86] are focused on capturing the relationship of runoff between sprinklers. For tree irrigation, however, there is little runoff due to the tree space. The soil moisture model for tree irrigation needs to consider the soil-water relationship under different depths. Second, as shown in [104], the decay of volumetric water content derived from the long-term model of [86] was shown to be much quicker than the real-world scenarios. It is bound to irrigate lightly and frequently, which has been found to be inefficient [105].

DRL-based Control. DRL has been applied in many applications, such as network planning [90], large language model training [106, 107], sensor energy management [108], mobile app prediction [109, 110] and building energy optimization [47, 111]. However, this paper tackles some unique challenges for irrigation control. First, we define an irrigation reward function that considers three cases for tree irrigation, as introduced in Section 4.3.3. Second, to prevent any possible damage to plants' health, we adopt a safe mechanism that replaces some unwise actions generated by DRL agent. Third, due to the data inefficiency, we leverage a data-driven simulator to speed up the training process.

4.8 Conclusions

We present *DRLIC*, a DRL-based irrigation system that generates optimal irrigation control commands according to current soil water content, current weather data and forecasted weather information. A set of techniques have been developed, including our customized design of DRL states and reward for optimal irrigation, a validated soil moisture simulator for fast DRL training, and a safe irrigation module. We design *DRLIC* irrigation node and build a testbed of six almond trees. Extensive experiments in real-world and simulation show the efficiency of *DRLIC* system.

Chapter 5

Virtual Machines Rescheduling Using Deep Reinforcement Learning in Data Centers

Modern industry-scale data centers receive thousands of virtual machine (VM) requests per minute. Due to the continual creation and release of VMs, many small resource fragments are scattered across physical machines (PMs). To handle these fragments, data centers periodically reschedule some VMs to alternative PMs. Despite the increasing importance of VM rescheduling as data centers grow in size, the problem remains understudied. We first show that, unlike most combinatorial optimization tasks, the inference time of VM rescheduling algorithms significantly influences their performance, causing many existing methods to scale poorly. Therefore, we develop a deep reinforcement learning system for VM rescheduling, VMR^{2}L, which incorporates a set of customized techniques, such as a two-stage framework that accommodates diverse constraints and workload conditions, as well as an effective feature extraction module. Our experiments on an industry-scale data center show that VMR^{2}L can achieve a performance comparable to the optimal solution but with a running time of seconds.

5.1 Introduction

Cloud service providers allow end-users [112, 113] to access computing resources, such as CPU and memory . They adopt resource virtualization to maximize hardware utilization, allocating Virtual Machines (VMs) [114, 115] with the requested resources to end-users. An industry-scale data center typically has hundreds to thousands of Physical Machines (PMs), where each PM can host multiple VMs that run independently [116]. A central server manages all VM requests on PMs by performing two tasks, scheduling and rescheduling. When a new VM request arrives, the scheduling algorithm assigns it to a proper PM for different resource utilization goals, such as minimizing the overall fragment rate (FR) or maximizing the number of available PMs. For example, when the remaining resources on a PM cannot fulfill a VM request of X CPU cores, $X \in \{2,4,8,16,32\}$, a fragment occurs. FR is the ratio between the remaining CPU resources that cannot be further utilized by a X-Core VM and the total free CPU resources across all PMs.

An optimal scheduling algorithm needs to consider all the VM requests in an optimization problem. However, as the demand for computing resources increases, hundreds of VMs can enter and exit a data center per minute. The execution time to solve the resource allocation problem is too long to process all the requests in real time. Simple heuristic algorithms [117, 118, 119, 120] are normally used, but they are not optimal and may result in many fragments scattered across PMs. Therefore, the VM manager needs to periodically migrate some VMs to alternative PMs to reduce FR. As a result, rescheduling is critical to optimize resource usage. Taking a datacenter of two PMs as an example, each PM has 44 CPUs. PM1 has 24 allocated CPUs and 20 free CPUs. PM2 has 32 allocated CPUs and 12 free CPUs. The FR is calculated as ((20%16)+(12%16))/32=50%. X is set to 16, as VMs requesting 16 or less cores are the most popular specifications sought by end-users. In our dataset from a commercial data center, these VMs account for 88.87% of all VM requests. The cloud service providers want to save the available PM resources for future 16-core VMs. Although the total number of free CPUs on two PMs is 32, they can only serve one more 16-core VM on PM2, resulting a FR of 50%. Assuming there is a 4-core VM on PM2, we can reschedule this VM from PM2 to PM1, reducing FR to (16%16+16%16)/32 = 0%. All free CPUs can now be utilized for up to two 16-Core

VM requests. By simply rescheduling one VM, we can double the number of new 16-core VM requests that can be served.

We can formulate VM rescheduling as a Mixed Integer Programming (MIP) problem that finds the best VMs and migrates them to proper PMs for optimizing resource utilization. The problem may have some constraints from the service expectations (e.g., the maximum number of migrated VMs) and the available hardware resources (e.g., current number of allocated VMs and occupied PMs). An off-the-shelf MIP solver, such as Gurobi [121] and CPLEX [122], can be used to solve the optimization problem. However, solving a large-scale MIP problem costs minutes or even hours, especially for large data centers beyond thousands of VMs and PMs. Such a long running time is intolerable, since new VM requests will have arrived during this delay, causing the generated solution to be no longer optimal or even feasible.

To accelerate the running speed of the MIP approach, hand-tuned heuristics can be integrated into the process, e.g., adding constraints to limit the solver's search space. These heuristics must trade-off between the optimality of the solution and the tractability of the problem. Unfortunately, even highly-skilled experts need many iterations to find a proper trade-off manually. Moreover, no universal heuristics that can achieve a good trade-off for all VM rescheduling scenarios. Operators have to manually examine and repeat the iterative design process for every scenario.

In this work, we develop VMR^{2}L, a deep Reinforcement Learning (RL) system for VM rescheduling. VMR^{2}L trains a Deep Neural Network (DNN) as the rescheduling agent that learns an optimal rescheduling algorithm by interacting with a data center. The DNN agent takes the current state of the data center as input, and outputs a pair of VM and PM as action (i.e., migrating the VM to the PM). Such a DNN inference can be done within 10 ms. We can run the inference multiple times to move a certain number of VMs to new PMs. To avoid the cost of training a VM rescheduling agent in a real data center, we use a simulator that generates the next state (the VM mapping over all PMs) according to the current state and the VM migration action. As a result, VMR^{2}L can provide comparable rescheduling performance as the MIP solver, but only within 1 second. To transform VMR^{2}L into a practical system, we need to tackle the following three challenges.

First, a commercial data center needs to handle thousands of VMs over hundreds of PMs. An action of VMR^{2}L is to find a VM and migrate it to a proper PM. The action space is large, e.g., almost millions of combinations. It's hard for RL agent to explore such a large action space and learn a good policy [123, 124]. To handle this problem, we design a two-stage VM rescheduling pipeline. The first stage selects a VM to be migrated and masks out all the PMs that cannot host the selected VM due to some constraints, such as insufficient CPU resources or affinity between VMs. The second stage chooses the appropriate destination PM to host the selected VM. Additionally, we also inject domain knowledge from conventional methods by pretraining the first stage via knowledge distillation.

Second, VMs continually enter and exit data centers, leading to a dynamic number of VMs deployed on PMs; however, VMR^{2}L's rescheduling agent takes the mapping of all VMs over PMs as the input. Due to a dynamic number of VMs, the state's dimension to the RL agent changes too. To handle this problem, we leverage a transformer-based neural network to extract effective features from raw data. Notably, we recognize the VM rescheduling problem as reassigning edges on a bipartite graph composed of two-level trees. To incorporate the graph information into our model, we introduce a sparse attention within each local tree and redesign the transformer network accordingly.

Third, we implement a prototype of VMR^{2}L including the above three key components. Extensive evaluation on two real datasets demonstrates that The results show that under a typical workload, VMR^{2}L can generate a solution within one second and its solution is only 2.67% worse than the optimal solution. We also verify that VMR^{2}L can generalize to different workloads and additional objectives beyond FR.

We summarize the contributions of this paper as follows:

- **RL for VM rescheduling.** We formulate the VM rescheduling problem as a Markov Decision Process (MDP), and develop a RL-based system, VMR^{2}L, to solve this problem.

- **Customized RL techniques for VM rescheduling.** We tackle three challenges in designing a RL framework for VM rescheduling and tackle them by three customized RL techniques, including two-stage VM rescheduling pipeline, effective feature extraction, and risk-seeking evaluation of the VM rescheduling agent.

- **A VMR^{2}L prototype and extensive evaluation.** We develop a prototype of VMR^{2}L and conduct extensive experiments over two real datasets. We release the datasets and a custom Gym environment for RL training.

5.2 Related Work

Connections to Bin Packing. The use of optimized placement mechanisms proved to be successful in a broad set of use cases, including production quality scenarios [125, 60] as well as transportation logistics [126, 127, 128, 129]. A typical solution exploits heuristics based on bin packing [130]. In fact, VM placement can be modeled as a bin-packing problem, where VMs and PMs are objects and bins, respectively. Bin packing typically involves packing a set of items into fixed-sized bins such that the number of bins required [126] or the total surface area is minimized [127, 128]. However, there are two notable differences. First, the problem of VM rescheduling concerns adjusting an initial assignment of VMs to PMs and has practical value in the industry, since some VMs might terminate causing the problem state to change. On the other hand, rebalancing items that are already packed in bins has received little attention in the context of other traditional bin-packing applications. Second, the total number of VMs and PMs in a data center can easily go into the range of several thousands or more [129], and is far more than the typical scale of bin packing problems, which typically involve no more than a few hundred items [131, 132, 133, 134].

RL for Optimization Problems. RL has been recently introduced to solve optimization problems, e.g., building ML compilers and optimizing neural network architectures [135]. In particular, RL is used to select branching variables or find cutting planes in the Branch-and-cut method [136, 137, 138, 139, 140, 141, 142]. Besides, RL can also be applied to existing heuristics for MIPs to further increase the quality of solutions [143, 144, 145]. However, the above approaches are not directly appropriate for the VM rescheduling task due to their poor computation complexity. Although they are designed to accelerate MIPs, even a state-of-the-art technique as POP [146] fails to deliver a satisfying solution within the second-level inference time limit required for the VM rescheduling task.

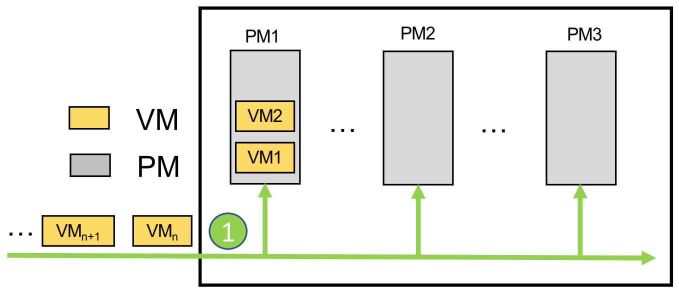


Figure 5.1: VM scheduling procedure.

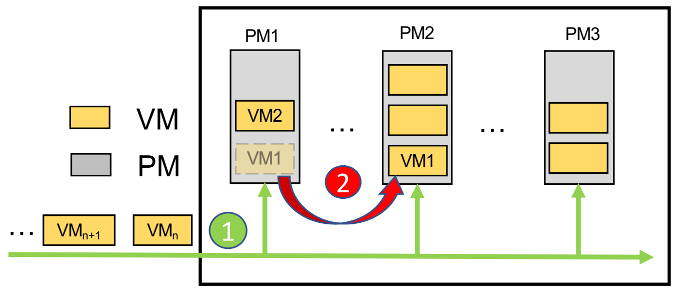


Figure 5.2: VM rescheduling procedure.

5.3 Background

5.3.1 VM Scheduling and Rescheduling

Cloud service providers offer VMs to users for their computation requests. This process involves 1 VM Scheduling and 2 Rescheduling stages shown in Figure 5.1 and Figure 5.2.

VM Scheduling (VMS). When new VM requests arrive, there are multiple available PMs that can host them. A typical solution exploits heuristics based on bin-packing, where VMs and PMs are objects and bins respectively. For instance, the first-fit algorithm [117, 118] traverses all the PMs and places the VM on the first PM that can meet the CPU and memory requirement of the VM request. The best-fit algorithm [119, 120] sorts all PMs that can meet the requirements of the current VM according to the amount of the FR reduction before and after this VM is added, and schedule the PM with the largest FR reduction. The best-fit method is currently used in **anonymous company**'s data centers to reduce FR.

VM Rescheduling (VMR). The VM rescheduling algorithms migrate VMs from their source PMs to destination PMs. Due to the overhead of VM migrations, a

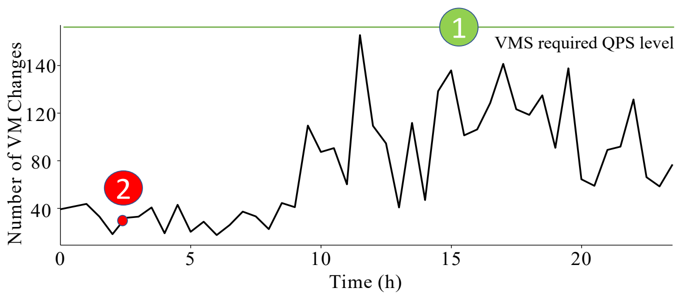


Figure 5.3: Number of VM changes.

number limit is set to control the number of VMs to migrate. At **anonymous company**, whenever a high FR is observed that could potentially lead to insufficient resources for upcoming VM requests, a round of VM rescheduling is initiated.

The Necessity of Rescheduling. Commercial data centers need to handle hundreds of VMs (queries) arriving and exiting per second (QPS) with a high processing throughput. We collect a 30-day dataset from an **anonymous company** data center including the number of VMs changes (VMs arriving and exiting) per minute. We also do experiments with the data from a large-scale data center in Section 5.5. Figure 5.3 depicts the maximum number of VM changes traced per minute over 24 hours. To ensure scheduling is robust, the VM rescheduler needs to meet the maximum number of VMs changes rather than the average, as indicated by the green line in Figure 5.3. During the running time of a VM rescheduling algorithm, the VM scheduling algorithm is still processing incoming VM requests and some accomplished VMs may also be deleted. To mitigate the impact of VM allocation and deallocation, the VM rescheduling process is typically performed periodically (e.g., every day, mostly during off-peak hours). Figure 5.3 highlights the time (2:00 AM) to execute the VM rescheduling algorithm with a red point.

5.3.2 Problem Formulation and Motivation

Problem Formulation and Two Algorithms

In a data center, let $\mathcal{V}, \mathcal{P}$ be the set of VMs and PMs, respectively. On the supply side, a PM $i \in \mathcal{P}$ has two NUMAs[1], where NUMA j can provide $U_{i,j}$ CPU resources and $V_{i,j}$ memory resources. On the demand side, a VM $k \in \mathcal{V}$ requires

[1]Non-uniform memory access.

Table 5.1: The VM types we consider in our experiments.

VM Types	large	xlarge	2xlarge	4xlarge	8xlarge	16xlarge	22xlarge
Requested CPU	2	4	8	16	32	64	88
Requested Memory (GB)	4	8	16	32	64	128	256
Deploy NUMA	Single	Single	Single	Single	Double	Double	Double
Percentage	11.01%	43.51%	16.13%	17.62%	6.22%	5.31%	0.20%

u_k CPU resources and v_k memory resources and should be deployed on a single PM using $w_k \in \{1, 2\}$ NUMAs. w_k is the number of NUMAs required by VM k (1 for single-NUMA deployment, 2 for double-NUMA). After deploying several VMs on PM $i \in \mathcal{P}$, it remains $\tilde{U}_{i,j}$ spare CPU resources on NUMA j. We define **X-core fragment** of PM i as $\sum_j (\tilde{U}_{i,j} \% X)$, i.e., the remaining CPU resources cannot be further utilized by any additional X-core VMs.

Given an initial assignment of M VMs each onto one of the N PMs, the VM rescheduling task is to reassign a subset of deployed VMs and migrate them each onto a new PM. The maximum number of VMs that we can migrate for a given task is called *Migration Number Limit (MNL)*. We formulate the VM rescheduling as an optimization problem that searches for a reassignment of MNL-selected VMs, in order to minimize the total X-core fragments across all PMs:

$$\text{Minimize:} \quad \sum_{i,j} \left(U_{i,j} - \sum_k \frac{x_{k,i,j} \cdot u_k}{w_k} - X y_{i,j} \right) \tag{5.1}$$

$$\text{Subject to:} \quad \sum_k \frac{x_{k,i,j} \cdot u_k}{w_k} + X y_{i,j} \leq U_{i,j}, \tag{5.2}$$

$$\sum_k \frac{x_{k,i,j} \cdot v_k}{w_k} \leq V_{i,j}, \tag{5.3}$$

$$\sum_{i,j} x_{k,i,j} = w_k, \tag{5.4}$$

$$\sum_k (1 - x_{k,i_k,j_k}) \leq MNL, \tag{5.5}$$

$$x_{k,i,0} = x_{k,i,1}, \quad \forall k \in \{k | w_k = 2\}, \tag{5.6}$$

$$x_{k,i,j} \in \{0, 1\} \text{ and } y_{i,j} \in \mathbb{Z}. \tag{5.7}$$

Here, $\{x, y\}$ are the decision variables, where $x_{k,i,j}$ represents whether VM k is deployed to the NUMA j of PM i in the new assignment (0 for No, 1 for Yes), and $y_{i,j}$ represents the maximum number of X-core VMs can be deployed on NUMA j of PM i using the remaining CPU resources. The objective in Equation 5.1 is to minimize the total X-core fragments.

Equation 5.2 and 5.3 enforce that the resource usage by VMs cannot exceed the total capacity of a PM. Equation 5.4 indicates that each VM must be deployed on exactly one PM. Equation 5.5, in which i_k and j_k are the initial PM id and NUMA id (0 for double-NUMA VMs) of VM k, means the total migration number should not exceed the limit. Lastly, Equation 5.6 restricts VMs with double NUMAs from deploying both of its NUMAs on the same PM.

Note (1) each PM has two NUMAs; (2) w_k is a constant for each VM as determined by their types (Table 5.1). Thus, $\sum_{i,j} x_{k,i,j} = w_k$ (Equation 5.4) enforces that the actual NUMA allocation number of VM k matches the desired configuration. When $w_k = 1$, Equation 5.4 constraints VM k to be deployed on one NUMA of a PM; when $w_k = 2$, Equation 5.4 constraints VM k to be deployed on both NUMAs of a PM. Note that deploying VM k on two NUMAs of two different PMs (each PM hosting a NUMA) violates $x_{k,i,0} = x_{k,i,1}, \forall k \in \{k|w_k = 2\}$ (Equation 5.6). Because $w_k \neq 0$, it guarantees each VM is deployed.

Mixed Integer Programming (MIP) Solvers. The formulation qualifies as a Mixed Integer Programming (MIP) problem because it includes integer decision variables $x_{k,i,j}$ and $y_{i,j}$, as well as linear constraints and an objective function that intertwines these integers with continuous resource parameters, embodying a classic MIP structure with a blend of discrete and continuous elements. The above optimization problem can be solved by an off-the-shelf MIP solver such as Gurobi [121] and CPLEX [122], which can find an optimal solution through algorithms of branch & bound, cutting planes, etc. In our experiments, we use the former. The primary issue of the MIP approach is that its poor time complexity prevents it from scaling to industry-scale data centers with thousands of PMs and VMs.

Greedy Algorithm (GA). To obtain a feasible solution within a short time frame, greedy algorithms are often used in industry data centers [147]. They normally include two stages: filtering and scoring. In the filtering stage, we calculate the change in FR for each VM if it is removed from its source PM, and only select the VM candidate that corresponds to the most significant drop in FR. In the scoring

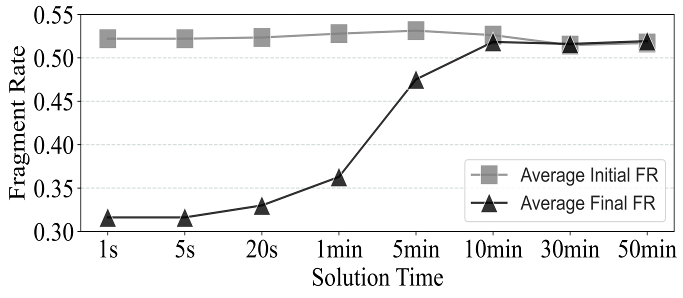


Figure 5.5: Effect of the Running Time of MIP at MNL 50.

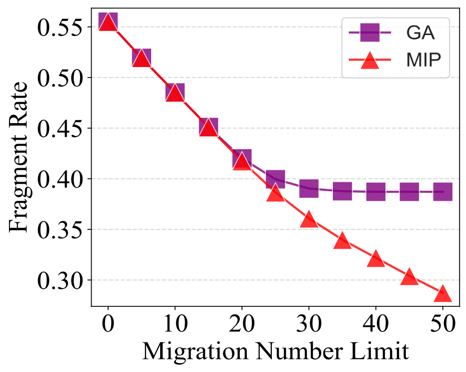


Figure 5.6: Fragment Rate Performance.

stage, we calculate the change in FR if the selected VM is migrated to each of the eligible PMs. We then greedily assign the selected VM to the PM that leads to the largest drop in FR. The above two stages are repeated until the MNL is reached. Although GA can significantly reduce the search space of the rescheduling problem, we show that it often fails to yield a good solution.

Experiment Results

We conduct experiments to quantify the performance of the above MIP and GA algorithms in terms of FR and running time. We use a real dataset collected from a data center with 280 PMs and 2089 VMs. The detailed experiment settings will be found in Section 5.6.

Figure 5.6 depicts the FR performance of MIP and GA under different MNLs. MIP's FR is higher than the GA since it guarantees a near-optimal solution. Moreover, the FR gap between MIP and GA becomes larger as MNL increases, because

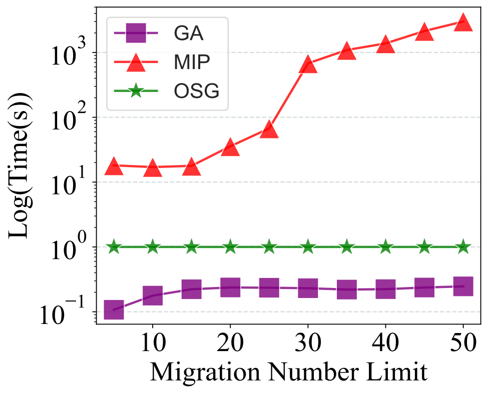


Figure 5.7: Running time.

GA only exploits the action that would lead to the most significant drop in FR when it migrates one VM.

In Figure 5.7, the time required for MIP and GA to generate a solution is displayed, and the green dashed line indicates the one-second guarantee (OSG). If the solution time is less than 1 second, the FR performance is guaranteed and the impact of VM changes on FR is negligible. However, as the MNL increases from 25 to 50, the computation time of MIP grows exponentially, taking 1.78 minutes for 25 migrations and 50.55 minutes for 50 migrations. One contributing factor is the NP-hard nature of VM rescheduling, which leads to an exponential increase in complexity as the problem size grows. This is evident when the number of MNLs rises, significantly expanding the possible solutions and solution space. Additionally, Gurobi uses a range of heuristics and cutting strategies to accelerate solving these problems. However, the efficiency of these strategies varies based on the specific characteristics of the problem. Notably, when the number of MNLs reaches around 25, these heuristics may lose their effectiveness, resulting in a marked increase in the time required to find a solution. This poor running time performance is unacceptable in data centers where new VM requests continually come in, and the problem state is constantly varying. In contrast, the running time of the GA algorithm is lower than the OSG and remains constant when the MNL exceeds 25. After migrating 25 VMs, the GA algorithm can no longer find any more VMs that can lower the FR.

The dynamic nature of VM allocation and deallocation can affect FR performance if the migration plan is based on the current VM-PM mapping and solved after 50 minutes. We quantify the performance reduction from two aspects. First, we measure the percentage of failed VMs during the VM Rescheduling stage, where a VM is considered a failure if it cannot be found, or there are not enough available resources to migrate it from the source to the destination PM. Second, we measure the reduction in FR percentage, which is calculated by comparing the initial FR and final FR after migrating all 50 VMs using MIP's solution. The top subfigure in Figure 5.5 shows the failed VM percentage, and the bottom subfigure shows the FR reduction percentage.

We can see from the figures that the VM Rescheduling can successfully schedule all 50 VMs, and the FR performance is 93.14% between the initial and final FR after migrating all the 50 VMs at VM generation speed 0/min, meaning no VM changes. However, the percentage of failed VMs increases, and the FR performance decreases as the VM changes speed rises. If the VM changes speed exceeds 40/min, there is no successful VM migration, and no performance gain, rendering the migration plan useless. The green dash lines show the initial and final FR if the MIP can run in 1s, we can see that the dynamic of VM allocation and deallocation has no effect on the FR performance. So, the running time of VM rescheduling algorithm should meet the one-second guarantee.

Limitations of the Current Methods. From the above experiment results, we see that the MIP solver has the scalability challenge due to its intensive computation overhead. Therefore, the greedy algorithm is currently used in data centers to achieve fast FR decrease. However, its FR performance can be stuck into a local optimal since it is focused on one VM at each step, without considering future steps. To reduce the execution time of MIP solvers, some current methods rely on estimating feasible solutions using proprietary heuristic methods and then using branch-and-cut techniques [148, 149] to identify optimal solutions. The hand-tuned heuristics are based on human expertise to overcome the scalability challenge. The heuristics aim to prune the search space to make the problem tractable. Yet, they all rely on human expertise and operational experience. It is a tedious, time-consuming iterative process even for experts. Even worse, such a process has to be done and tuned for every VM rescheduling problem as there are no universal heuristics for all scenarios.

5.4 Design of VMR^{2}L

5.4.1 VM Rescheduling as an RL Problem

Under the Markov Decision Process (MDP) framework, at each time step t, the scheduling agent selects an action $A_t = a$, given the current state $S_t = s$, based on its policy π_θ. More specifically, the agent's action is to select a VM and a PM, by which it reschedules the selected VM from its source PM to the chosen destination PM. T is the state transition function defined as $T : S \times A \rightarrow S$, which maps the current state to the next state given an action. The episode terminates if the selected VM cannot find a suitable PM or the number of actions taken reaches a pre-defined threshold. The latter is consistent with the requirement upon deployment that each rescheduling comes at a cost and we may only move the VMs around for a limited number of steps. Notably, in this problem, T is deterministic as the environment has no aleatoric uncertainties. In other words, we can directly simulate the next state and the change in FR given the current state and action taken by the agent. Thus, we build a cheap simulator so that our agent learns by interacting with the simulator instead of with the real environment. Given the above design, we now formalize the state, action, and reward of VMR^{2}L 's RL framework.

State Representation. The state is what the DRL agent takes as input at each rescheduling step. The state representation includes **1) PM features** - 8 in total: the first four features are the remaining CPU resources on NUMA 0 and NUMA 1, as well as the remaining memory resources on NUMA 0 and NUMA 1. The other features are manually designed, which are the FR and the fragment size of each NUMA. The last four features allow the agent to directly access the internal fragment information. **2) VM features** - 14 in total: the first four features are the requested CPU and memory resources of NUMA 0 and NUMA 1, and the last two are designed to be the fragment sizes of NUMA 0 and NUMA 1. We also include the 8 features from its source PM to better incorporate the locality information of each VM. Note that if a single NUMA is requested, we use zeros as placeholders for the other NUMA. For training efficiency, we scale each feature dimension with min-max normalization.

Action Representation. The DRL agent interacts with the environment via actions. The action at each step can be represented as a 2-tuple (k, i), where $k \in \mathcal{V}$

and $i \in \mathcal{P}$. Specifically, the action is to reschedule a VM k from its source PM to a destination PM i. Note that the source PM can be retrieved once we select k, so we do not include it in the action space.

Reward Representation. Reward represents the immediate evaluation of the rescheduling effect for each action under a certain state. The goal of VM rescheduling is to minimize the FR across all PMs. While in principle we could return the FR of all PMs as a single final reward to the agent after finishing an entire episode, it corresponds to a form of sparse reward and it is known to be difficult for training [150], as the action at each step only contributes marginally to the change in the overall FR. Instead, we propose to generate dense rewards and use the change in FR on the source PM and the destination PM as an intermediate reward at each step. As such, the range of the reward is naturally scaled down to a $[-2, 2]$ range, which we further normalize by dividing with a constant c (c=4 in our case)[2]. Equation 5.8 calculates the fragment size on each NUMA, and Equation 5.9 shows fragment changes for the source PM and destination PM.

$$S_i = \sum_{j=0}^{1} \left(\tilde{U}_{i,j} \% X\right) \div c, \tag{5.8}$$

$$R = (S_{\text{before, src}} - S_{\text{after, src}}) + (S_{\text{before, dest}} - S_{\text{after, dest}}), \tag{5.9}$$

where $S_{\text{before},\cdot}$ and $S_{\text{after},\cdot}$ show the fragment changes before and after this selected VM leaves (enters) the source (destination) PM. Recall that we focus on $X = 16$ in this paper.

5.4.2 A Two-Stage Framework

When the RL agent takes action (k, i), meaning to reschedule a VM k from its source PM to a destination PM i, PM i might not have enough available CPU or memory to host VM k. In addition to resource constraints, in practical scenarios, we must consider additional constraints to ensure service stability. For example, a critical application might request for several VMs to be hosted across different PMs, which can be enforced in the form of a hard anti-affinity constraint.

Off-the-shelf RL models impose such hard constraints typically by invoking a

[2]It is a common practice to use reward scaling to get better results for deep RL [151].

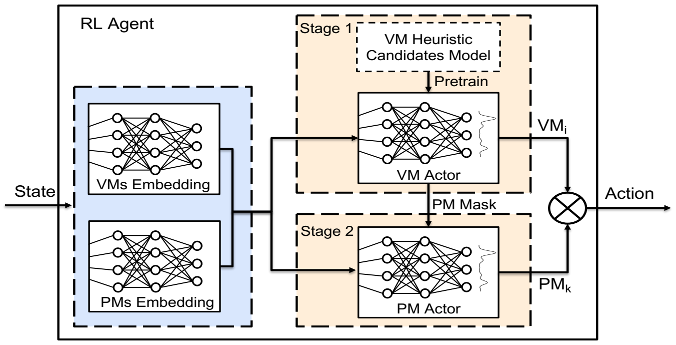


Figure 5.8: The two-stage RL agent.

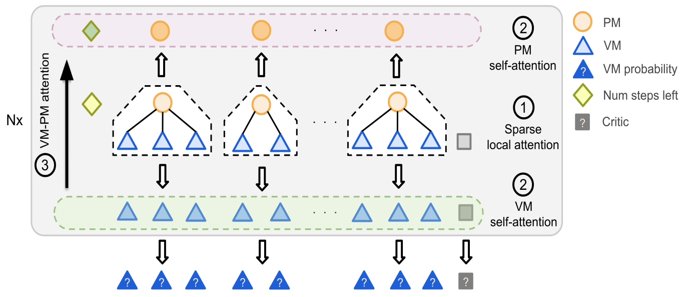


Figure 5.9: VM actor with sparse local-attention capturing tree-level features.

heavy penalty if an illegal action is chosen, or by directly setting the probabilities for all illegal actions to be zero. Heavy penalties can result in gradient instability and thus lead to an inferior convergence rate. On the other hand, as the size of the action space is $O(\mathcal{V} \cdot \mathcal{P})$, masking all illegal actions is overly time-consuming and fails to meet the latency requirement.

To better accommodate a variety of constraints, we leverage the characteristics of the VM rescheduling problem and design a two-stage framework that allows the action tuple to be generated sequentially. As shown in Figure 5.8, in Stage 1, the VM actor selects the VM candidate to be rescheduled. Once a candidate VM is selected, we can efficiently mask out all the PMs that cannot host the candidate VM and then proceed to Stage 2, where the PM actor selects an appropriate destination PM from the remaining PMs. Additionally, when we select a VM to reschedule, a considerable portion of the PMs cannot meet its resource requirements. The exploration of RL agent on these PMs inevitably hinders the learning process. The proposed framework can effectively reduce the large action space and thus mitigate the exploration challenge.

5.4.3 Feature Extraction with Sparse Attention

Scalability. To make effective rescheduling decisions, *DRLIC* must extract meaningful representations of the state observation, which include features of each individual PM and VM as well as their affiliations. As Figure 5.3 implies, the number of VMs can vary drastically even in the same data center. This implies that the size of the features at each time step is also highly dynamic. To encode these features, one option is to concatenate features from all VMs and PMs into a long vector. However, this approach cannot handle an arbitrary number of VMs as neural networks usually require fixed-sized inputs, and it also requires a model with a large number of parameters that would be difficult to train.

Instead, we propose to share two small embedding networks across all VMs and PMs — one to process each PM's features and another one to process each VM's features. As such, the number of weight parameters is *independent* of the number of machines in the system. This is achieved via an attention-based transformer models [152] but tailored for rescheduling. Transformers have demonstrated strong performances in Natural Language Processing [153, 154, 155, 156], Computer Vision [157, 158, 159], as well as combinatorial optimization, such as in vector bin-packing [160, 161]. However, compared to bin-packing, there is a notable difference in VM rescheduling: we must choose from a set of VMs that have already been assigned to PMs.

Tree-level Features. Consider a PM with 2 CPUs left. It contains a VM with 4 CPUs and another VM with 2 CPUs. Suppose a second PM has a fragment size of 8 while hosting a VM with 8 CPUs. In order to minimize the total 16-core fragments, an ideal approach would be to first remove the two VMs with 2 and 4 CPUs from the first PM, and then reassign the VM with 8 CPUs from the second PM to the first PM. However, if we merely include the source PM's features in each of the VM's features and feed them into the vanilla transformer model, there will not be sufficient information for the two actors to take the above actions. Instead, each VM must also be aware of the other VMs that are hosted on the same PM, which is not possible in the vanilla transformer model. In fact, each PM can be viewed as a tree of depth one, where the PM acts as the root node and the VMs it hosts act as the leaf nodes.

In order to allow every VM to recognize which other VMs are hosted on the same PM, we propose to include an additional stage of *sparse local-attention* within each PM tree, i.e., we only allow PMs and VMs to attend to each other if and only if they belong to the same tree.

5.5 Implementation

RL algorithm and Neural Network Architecture We implement VMR^{2}L based on the CleanRL framework [162] using PPO as the backbone [62]. VMR^{2}L contains about 8.5K lines of Python code. The overall framework is implemented using PyTorch [163]. As the model can be trained end-to-end, it does not require manual feature engineering and can learn statistically what is important from the features. On the other hand, domain experts can inject prior knowledge into VMR^{2}L in the form of heuristic models through knowledge distillation. Additionally, the number of model parameters is independent of the number of VMs or PMs, allowing it to scale to large data centers.

VM Rescheduling Simulator. While DRL can be very powerful, its main drawback is the amount of training data required [164]. In light of this, we design a simulator for the VM rescheduling task. The simulator follows the OpenAI Gym environments [165] including specific file hierarchy and function abstractions. Given an existing VM-PM mapping and a rescheduling action, we can directly simulate the change in FR caused by the action. Thus, during training, VMR^{2}L only needs to interact with the simulator instead of with the real environment, which drastically lowers the amount of real-world data required to train the agent.

Experiment Setup. Experiments are done on a Linux server with 8 CPUs (Intel Xeon E5) and 1 GPU (NVIDIA RTX 3090). The training time for knowledge distillation, VMR^{2}L with sparse attention, and VMR^{2}L without sparse attention is 2h, 92h, and 48h, respectively. We sample 8 trajectories for VMR^{2}L and report the best result. We report the average over 3-5 runs with different random seeds.

5.6 Evaluation

5.6.1 Simulator and Datasets

While DRL can be very powerful, its main drawback is the amount of training data required [164, 111]. In light of this, we design a simulator for the VM rescheduling task. The simulator follows the OpenAI Gym environments [165] including specific file hierarchy and function abstractions. Given an existing VM-PM mapping and a rescheduling action, we can directly calculate the change in FR caused by the action. Thus, during training, VMR^{2}L only needs to interact with the simulator instead of with the real environment, which drastically lowers the amount of real-world data required to train the agent.

As for the datasets, we have two seed initial mappings from an industry-scaled real cloud data center – one medium dataset with 2089 VMs and 280 PMs, and one large dataset with 4546 VMs and 1176 PMs. Note that the RL agent must be able to generalize to VM-PM mappings unseen during training and a dynamic number of VMs at deployment time. To better evaluate the agent's performance under various initial mappings, we generate 4400 initial mappings with different numbers of VMs for both the medium and the large datasets. Each mapping is generated by removing the existing VMs on each PM and randomly scheduling some of them to any PM that can fit them. We generate three versions of the middle dataset with low, middle, and high workloads (different remaining CPU resources). We leverage 4000 datasets for training, 200 datasets for both validation and test. Both the simulator and datasets are available to the research community.

5.6.2 Existing Baseline Algorithms

Existing baselines can be summarized into six categories: heuristics (e.g., greedy, α-VBPP), optimization algorithms (e.g., MIP), approximate algorithms (e.g., POP), search-based algorithms (e.g., MCTS), deep learning-based (e.g., Decima), and hybrid methods (e.g., NeuPlan). We compare with at least one representative algorithm from each category.

MIP Algorithm: introduced in Section 5.3.2.

Greedy Algorithm: introduced in Section 5.3.2.

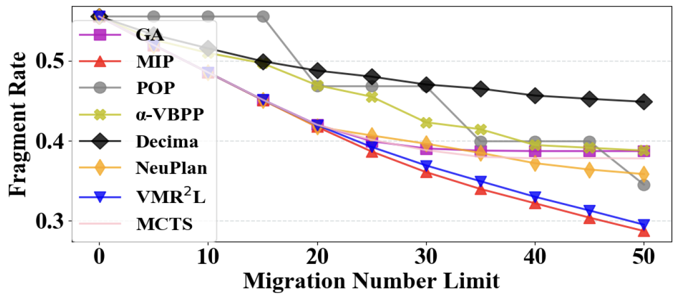


Figure 5.10: Fragment Rate Performance.

Vector Bin Packing Problem (α-VBPP): We generalize the VBPP [130] algorithm for initial scheduling to rescheduling. We first divide the entire episode into MNL/α stages. During each stage, we greedily remove α number of VMs that lead to the most fragments, and then apply VBPP to treat them as incoming VMs. We carefully tune α (10 in our case) to achieve the best FR reduction.

Partitioned Optimization Problems (POP) [146]: It solves the optimization problem formulated in Section 5.3.2 by splitting the problem into subproblems (each containing a subset of VMs and PMs) and coalescing the resulting sub-rescheduling solution into a global rescheduling solution for all VMs.

Monte-Carlo Tree Search (MCTS) [134]: As traditional search based methods need to perform multiple rollouts during inference time to achieve a good performance, we use DDTS [134] to prune the search space.

Decima [124]: Decima uses a graph neural network to encode the VM and PM information and trains using deep RL. Decima balances the size of the action space and the number of actions required by decomposing VM rescheduling decisions into a two-dimensional action, which outputs i) the VM that needs to migrate, and ii) an upper number of PM subsets to choose as the destination.

NeuPlan [90]: In the first stage, an RL agent takes in the problem as a graph and generates the first few VM migrations to prune the search space. In the second stage, it uses a MIP solver for the remaining MNLs. A relax factor β (30 in our case) is used to control the size of the MNL space to be explored by MIP.

5.6.3 Overall Performance

Figure 5.10 and Figure 5.11 shows the FR and running time of all methods. The absolute numbers of Figure 5.10 and Figure 5.11 are listed in Table 5.2.

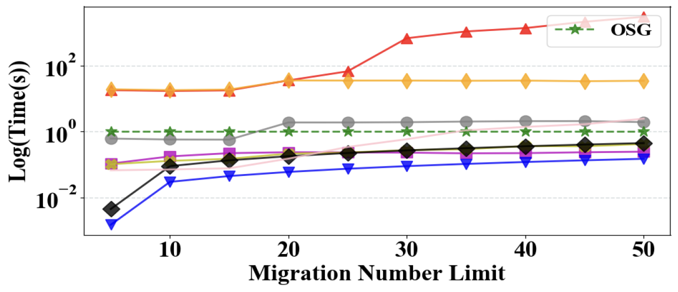


Figure 5.11: Inference Time

Table 5.2: Fragment Rate & Solution Time(s)

Method	**MNL=10**		**MNL=20**		**MNL=30**		**MNL=40**		**MNL=50**	
	FR	**Time**	**FR**	**Time**	**FR**	**Time**	**FR**	**Time**	**FR**	**Time**
GA	**0.485**	0.18	0.420	0.24	0.390	0.23	0.387	0.22	0.387	0.24
POP	0.555	0.57	0.468	1.90	0.468	1.93	0.399	2.07	0.345	1.94
Decima	0.515	0.12	0.487	0.24	0.470	0.36	0.456	0.48	0.448	0.60
NeuPlan	**0.485**	18.06	**0.417**	35.70	0.396	35.31	0.372	35.20	0.358	34.80
RLVMR	**0.485**	**0.04**	0.419	**0.08**	**0.369**	**0.12**	**0.332**	**0.16**	**0.295**	**0.20**
MIP	0.485	17.06	0.417	35.70	0.361	680	0.322	1380	0.287	3033

Fragment Rate. The results in Figure 5.10 reveal that VMR^{2}L can achieve +14.48%, +17.60%, +22.37%, +23.71%, 23.84%, and +34.17% performance gain when compared to POP, NeuPlan, MCTS, GA, α-VBPP and Decima respectively, when the task is to perform 50 migrations on the medium dataset. Notably, VMR^{2}L is merely 2.67% behind the optimal MIP solution (0.2953 vs. 0.2859). It is worth noting that the performance gap between VMR^{2}L and the near-optimal MIP does not increase with more MNLs.

Although α-VBPP can reduce FR with more MNL, its performance is inferior to that of GA. This is because α-VBPP only removes the α worst VMs for each stage based on a single timestep, failing to consider future opportunities to replace them back and achieve even higher FR reduction. POP fails to achieve a good performance since it still relies on MIPs to solve each subproblem. To meet the second-level latency requirement, we must divide the problem into many subproblems, causing its solutions to be only locally optimal. On the other hand, Decima reduces the huge action space by limiting the PM actor to only select from a subset of PMs, but the subsampling of PMs is completely random, as opposed to our solution. While MCTS with DDTS uses neural networks to prune the search space, it still requires a significant number of rollouts to achieve stable performance. Lastly, NeuPlan is able to achieve a low FR, since it solves a large subproblem entirely with MIP. We implement NeuPlan as follows: if MNL is less than 20 steps, MIP is used to solve the optimization problem. If MNL is larger than 20, the first 20 VMs are migrated by MIP and the remaining VMs will be handled by RL. Notice that NeuPlan's FR is higher than VMR^{2}L after MNL = 20, since after this step NeuPlan relies entirely on its RL agent, which fails to learn a good policy given such a huge state and action space.

Inference Time. From Figure 5.11, we can see that the solution time of GA, α-VBPP, Decima and VMR^{2}L is less than OSG. VMR^{2}L can generate one trajectory within 0.15s when MNL = 50. In comparison, MIP requires 50.55 minutes to provide the optimal solution. The running time of POP can be adjusted by setting how many subproblems to divide into. To meet the latency requirement of the VM rescheduling task, we set this number to 16 so POP can deliver a solution within 1.94s. Both GA and α-VBPP are greedy algorithms and can provide the solution within 1s. Decima requires 0.45s, which is at a roughly same scale as VMR^{2}L, since both use end-to-end deep RL. Lastly, NeuPlan takes 34.8s to yield the final solution since it still needs MIP to solve 20 steps.

5.6.4 Scalability to the Large Dataset

We conduct experiments on a large dataset with 4546 VMs and 1176 PMs to analyze the scalability of VMR^{2}L. Fig. 5.12 shows the FR and time performance of different baseline methods with the MNL from 50 to 200. The MIP is not included in this experiment since we cannot get a solution within 50 minutes. As it turns out, VMR^{2}L again achieves better performance than the baselines on both FR and solution time.

From the left subfigure in Fig. 5.12, we can see that VMR^{2}L can achieve average 2.01%, 2.08%, 6.14% and 7.87% and max 2.34%, 5.15%, 8.6%, and 10.5% performance gain compared with POP, GA, NeuPlan, and Decima respectively. Decima and NeuPlan both cannot achieve good FR performance on such a large dataset. NeuPlan mostly relies on RL's solution since MIP cannot work with MNL larger than 20. GA

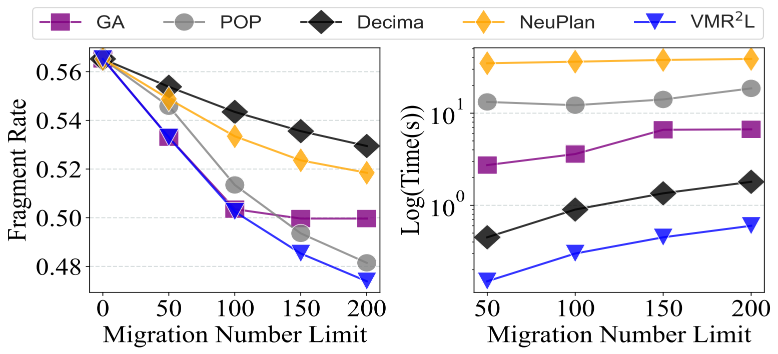


Figure 5.12: Fragment rate and inference time between different methods on the large dataset.

gradually stops reducing FR after 100 steps. POP and VMR^{2}L continue to reduce FR even at MNL 200. POP achieves worse FR than GA before 100 and better FRs for larger MNLs.

The right subfigure in Fig. 5.12 shows the running time to generate a migration solution with MNL set from 50 to 200. GA, POP, Decima, NeuPlan, and VMR^{2}L can generate a solution within 4.91s, 14.53s, 1.125s, 37.23s, and 0.375s on average. GA increases the time when MNL is less than 150. GA calculates the score of all the VMs and PMs as a metric to evaluate their migration value. The calculation time increases with the number of VMs and PMs. POP costs more time with a bigger MNL. VMR^{2}L increases time linearly from MNL 50 to 200. For VMR^{2}L, the neural network inference time will only increase minimally with the number of PMs and VMs. Decima needs three times than VMR^{2}L since the GNN needs to encode the VM-PM information. NeuPlan needs MIP to solve 20 MNL.

5.6.5 Different Service Objectives

VMR^{2}L's flexibility enables it to learn different policies depending on the high-level objective. We now consider a new objective: given FR goals, we would like to minimize the number of migrations needed to reach the FR goals. As seen in Fig. 5.13, the top subfigure displays the used MNL, while the bottom subfigure shows the FR, both sharing the x-axis with the FR goals. In general, as the FR goal decreases, GA requires significantly more migration steps than both MIP and VMR^{2}L. Note that none of the three methods can achieve the FR goal of 0.25, since the optimal

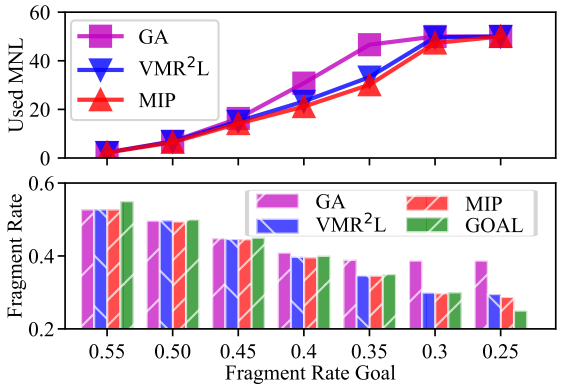


Figure 5.13: MNL Performance under Different FR Goals.

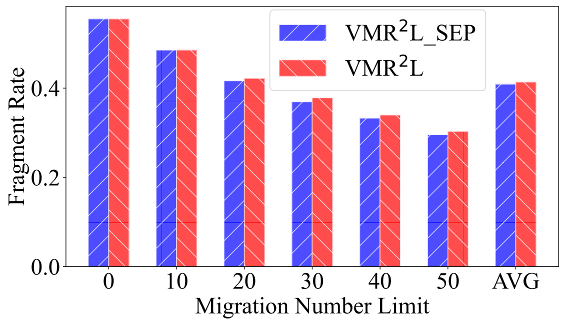


Figure 5.14: Fragment Rate Gap between VMR^{2}L and VMR^{2}L $_{SEP}$.

FR is 0.2859 at MNL 50. To summarize, the used MNLs of GA, VMR^{2}L, and MIP increase with lower FR goals. Across all FR goals, MIP and VMR^{2}L achieve 14.77% and 11.11% less MNL than GA, respectively. VMR^{2}L performs similarly to MIP, with a difference of only 3.66%, but with a millisecond-level inference time. On the other hand, GA is stuck at a FR goal of 0.4 since it only optimizes for the next-step performance, instead of the long-term performance.

5.6.6 Generalizing to Different MNLs

In practice, the required migration number limit (MNL) can constantly vary due to changing business requirements such as target fragment rates. We show that we can readily achieve good performance by only training one VMR^{2}L agent with MNL = 50, and deploying it for MNLs = $\{10, 20, 30, 40, 50\}$. For comparison, we train a separate VMR^{2}L agent for each MNL, which we denote as VMR^{2}L $_{\text{SEP}}$. As shown in Fig. 5.14, VMR^{2}L performs only marginally worse than VMR^{2}L $_{\text{SEP}}$ with an average FR performance gap of 1.16%. This suggests that the VMR^{2}L agent trained with a large MNL can be readily applied to the tasks with smaller MNLs. It avoids the overhead of maintaining a separate VMR^{2}L agent for each MNL.

5.7 Conclusion

Compared to conventional bin-packing applications, VM rescheduling presents unique challenges due to the expanding size of data centers. It needs to handle many

VMs while meeting a strict inference speed requirement. To this end, we introduce VMR^{2}L, a tailored deep RL solution featuring a two-stage framework for diverse service constraints and a sparse attention module for better feature extractions. We hope that our released datasets and RL environment will support future research in this area.

Chapter 6

Conclusions

In this body of work, we have made significant advancements in the optimization of cyber-physical systems, a realm that presents complex and evolving challenges. Our approach, grounded in deep reinforcement learning, has enabled us to design innovative systems across various applications. For instance, in HVAC control, we have prioritized minimizing energy consumption while ensuring human comfort. Similarly, our irrigation control systems are tailored to reduce water usage without compromising plant health. Another notable contribution is our two-stage DRL agent framework in data centers, which efficiently balances rescheduling performance with solution speed in complex environments.

Addressing the substantial data training requirements that are inherent in deep reinforcement learning, our research introduces a groundbreaking model-based reinforcement learning method. This approach has significantly reduced the training duration, cutting it down from an overwhelming five years to a mere 183 days. This reduction in training time is not just a quantitative improvement but a qualitative leap in the field of machine learning, enabling more agile and responsive system development. Furthermore, we have innovatively tackled the challenge of managing big action state space in VM rescheduling. Our two-stage VM rescheduling pipeline is a demonstration to this effort. The first stage of this pipeline involves selecting a VM for migration and masking out unsuitable physical machines (PMs) due to constraints such as insufficient CPU resources or VM affinity. The second stage is focused on choosing the optimal PM to host the selected VM. This method effectively narrows down the action space and enhances solution time performance significantly.

Finally, the effectiveness of these innovations has been rigorously validated through extensive testing against competitive baseline strategies in both real-life deployments and simulated scenarios. The results from these tests have been unequivocal. They demonstrate that our techniques not only yield substantial reductions in system energy and water consumption but also improve the solution times for VM rescheduling in data centers. More importantly, these advancements have significantly enhanced the quality of service. By elevating human comfort and plant health, and achieving near-optimal levels of efficiency comparable to mixed integer programming in data centers, our work sets a new benchmark in the field.

Overall, the contributions of this research are not only diverse in their application but also profound in their impact, setting new precedents in the optimization of cyber-physical systems.

Bibliography

[1] Field capacity. https://nrcca.cals.cornell.edu/soil/CA2/CA0212.1-3.php.

[2] R Troy Peters, Kefyalew G Desta, and Leigh Nelson. Practical use of soil moisture sensors and their data for irrigation scheduling. 2013.

[3] Soil quality indicators. https://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/nrcs142p2_053288.pdf.

[4] Michaela Killian and Martin Kozek. Ten questions concerning model predictive control for energy efficient buildings. *Building and Environment*, 2016.

[5] Anil Aswani, Neal Master, Jay Taneja, David Culler, and Claire Tomlin. Reducing transient and steady state electricity consumption in hvac using learning-based model-predictive control. *Proceedings of the IEEE*, 100(1):240–253, 2011.

[6] Mehdi Maasoumy, M Razmara, M Shahbakhti, and A Sangiovanni Vincentelli. Handling model uncertainty in model predictive control for energy efficient buildings. *Energy and Buildings*, 77:377–392, 2014.

[7] Samuel Privara, Jiří Cigler, Zdeněk Váňa, Frauke Oldewurtel, Carina Sagerschnig, and Eva Žáčeková. Building modeling as a crucial part for building predictive control. *Energy and Buildings*, 56:8–22, 2013.

[8] Xiaoshu Lü, Tao Lu, Charles J Kibert, and Martti Viljanen. Modeling and forecasting energy consumption for heterogeneous buildings using a physical–statistical approach. *Applied Energy*, 144:261–275, 2015.

[9] Oswaldo Lucon, Diana Ürge-Vorsatz, A Zain Ahmed, Hashem Akbari, Paolo Bertoldi, Luisa F Cabeza, Nicholas Eyre, Ashok Gadgil, LD Harvey, Yi Jiang, et al. Buildings. 2014.

[10] Wai Wai Shein, Yasuo Tan, and Azman Osman Lim. Pid controller for temperature control with multiple actuators in cyber-physical home system. In *IEEE NBiS*, 2012.

[11] Alex Beltran and Alberto E Cerpa. Optimal hvac building control with occupancy prediction. In *ACM BuildSys*, 2014.

[12] Bocheng Li and Li Xia. A multi-grid reinforcement learning method for energy conservation and comfort of hvac in buildings. In *IEEE CASE*, 2015.

[13] Zhiang Zhang and Khee Poh Lam. Practical implementation and evaluation of deep reinforcement learning control for a radiant heating system. In *ACM BuildSys*, 2018.

[14] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, and Jianhua Zou. Deepapp: A deep reinforcement learning framework for mobile application usage prediction. In *ACM SenSys*, 2019.

[15] Tianshu Wei, Yanzhi Wang, and Qi Zhu. Deep reinforcement learning for building hvac control. In *ACM DAC*, 2017.

[16] Athanassios Tzempelikos and Andreas K Athienitis. The impact of shading design and control on building cooling and lighting demand. *Solar energy*, 2007.

[17] Zhijin Cheng, Qianchuan Zhao, Fulin Wang, Yi Jiang, Li Xia, and Jinlei Ding. Satisfaction based q-learning for integrated lighting and blind control. *Energy and Buildings*, 2016.

[18] Liping Wang and Steve Greenberg. Window operation and impacts on building energy consumption. *Energy and Buildings*, 92:313–321, 2015.

[19] Kang Yang, Yuning Chen, Xuanren Chen, and Wan Du. Link quality modeling for lora networks in orchards. In *22nd ACM/IEEE IPSN*, pages 27–39, 2023.

[20] Roel De Coninck and Lieve Helsen. Practical implementation and evaluation of model predictive control for an office building in brussels. *Energy and Buildings*, 2016.

[21] D Kolokotsa, GS Stavrakakis, K Kalaitzakis, and D Agoris. Genetic algorithms optimized fuzzy controller for the indoor environmental management in buildings implemented using plc and local operating networks. *Engineering Applications of Artificial Intelligence*, 2002.

[22] Konstantinos Dalamagkidis, Denia Kolokotsa, Konstantinos Kalaitzakis, and George S Stavrakakis. Reinforcement learning for energy conservation and comfort in buildings. *Building and environment*, 2007.

[23] Zhiang Zhang, Adrian Chong, Yuqi Pan, Chenlu Zhang, Siliang Lu, and Khee Poh Lam. A deep reinforcement learning approach to using whole building energy model for hvac optimal control. In *2018 Building Performance Analysis Conference and SimBuild*, 2018.

[24] Xianzhong Ding, Alberto Cerpa, and Wan Du. Exploring deep reinforcement learning for holistic smart building control. *arXiv preprint arXiv:2301.11510*, 2023.

[25] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI 2016*, 2016.

[26] Kazufumi Ito and Karl Kunisch. *Lagrange multiplier approach to variational problems and applications.* Siam, 2008.

[27] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *AAAI*, 2018.

[28] Poul O Fanger et al. Thermal comfort. analysis and applications in environmental engineering. *Thermal comfort. Analysis and applications in environmental engineering.*, 1970.

[29] PO Fanger. Moderate thermal environments determination of the pmv and ppd indices and specification of the conditions for thermal comfort. *ISO 7730*, 1984.

[30] David Christopher Pritchard. *Lighting.* Routledge, 2014.

[31] Steven J Emmerich and Andrew K Persily. *State-of-the-art review of CO2 demand controlled ventilation technology and application.* Citeseer, 2001.

[32] ANSI/ASHRAE Standard 62.1. Ventilation for acceptable indoor air quality. 2016.

[33] sketchup, 2018.

[34] Michael Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *Journal of Building Performance Simulation*, 2011.

[35] U.S. Department of Energy. Energyplus 8.6.0, 2016.

[36] Alex Beltran, Varick L Erickson, and Alberto E Cerpa. Thermosense: Occupancy thermal based sensing for hvac control. In *ACM BuildSys Workshop*, 2013.

[37] ASHRAE Guideline. Guideline 14-2002, measurement of energy and demand savings. *American Society of Heating, Ventilating, and Air Conditioning Engineers, Atlanta, Georgia*, 2002.

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

[39] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

[40] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[41] Refrigerating American Society of Heating and Air-Conditioning Engineers. Standard 55. Thermal environmental conditions for human occupancy. 2017.

[42] Daniel A Winkler, Alex Beltran, Niloufar P Esfahani, Paul P Maglio, and Alberto E Cerpa. Forces: feedback and control for occupants to refine comfort and energy savings. In *ACM UbiComp*, 2016.

[43] Ltd. DR International.2012. 2011 building energy data book. `https://openei.org/doe-opendata/dataset/buildings-energy-data-book`.

[44] Jyri Salpakari and Peter Lund. Optimal and rule-based control strategies for energy flexibility in buildings with pv. *Applied Energy*, 2016.

[45] Daniel A Winkler, Ashish Yadav, Claudia Chitu, and Alberto E Cerpa. Office: Optimization framework for improved comfort & efficiency. In *ACM/IEEE IPSN*, 2020.

[46] Narendra N Kota, John M House, Jasbir S Arora, and Theodore F Smith. Optimal control of hvac systems using ddp and nlp techniques. *Optimal Control Applications and Methods*, 17(1):71–78, 1996.

[47] Xianzhong Ding, Wan Du, and Alberto Cerpa. Octopus: Deep reinforcement learning for holistic smart building control. In *ACM BuildSys*, 2019.

[48] Zoltan Nagy, June Y Park, and J Vazquez-Canteli. Reinforcement learning for intelligent environments: A tutorial. *Handbook of Sustainable and Resilient Infrastructure*, 2018.

[49] June Young Park and Zoltan Nagy. Hvaclearn: A reinforcement learning based occupant-centric control for thermostat set-points. In *ACM e-Energy*, 2020.

[50] Chi Zhang, Sanmukh R Kuppannagari, Rajgopal Kannan, and Viktor K Prasanna. Building hvac scheduling using reinforcement learning via neural network based model approximation. In *ACM BuildSys*, 2019.

[51] Siddharth Goyal and Prabir Barooah. A method for model-reduction of non-linear thermal dynamics of multi-zone buildings. *Energy and Buildings*, 2012.

[52] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE ICRA*, 2018.

[53] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *IEEE ICRA*, 2017.

[54] Tong Wu and Jorge Ortiz. Towards adaptive anomaly detection in buildings with deep reinforcement learning. In *ACM BuildSys*, 2019.

[55] Bharathan Balaji, Sunil Mallya, et al. Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In *IEEE ICRA*, 2020.

[56] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K Gupta. Aces: Automatic configuration of energy harvesting sensors with reinforcement learning. *ACM TOSN*, 2020.

[57] Zhihao Shen, Wan Du, Xi Zhao, and Jianhua Zou. Dmm: fast map matching for cellular data. In *ACM MobiCom*, 2020.

[58] Zhi Cao, Honggang Zhang, Yu Cao, and Benyuan Liu. A deep reinforcement learning approach to multi-component job scheduling in edge computing. In *IEEE MSN*, 2019.

[59] Miaomiao Liu, Xianzhong Ding, and Wan Du. Continuous, real-time object detection on mobiledevices without offloading. In *IEEE ICDCS*, 2020.

[60] Xianzhong Ding, Zhiyong Zhang, Zhiping Jia, Lei Ju, Mengying Zhao, and Huawei Huang. Unified nvtcam and stcam architecture for improving packet matching performance. *ACM SIGPLAN Notices*, 52(5):91–100, 2017.

[61] Bingqing Chen, Zicheng Cai, and Mario Bergés. Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy. In *ACM BuildSys*, 2019.

[62] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[63] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, 2017.

[64] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

[65] 2019 Sergey Levine. Model-based reinforcement learning. `http://rail.eecs.berkeley.edu/deeprlcourse/`.

[66] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *CoRL*, 2020.

[67] Xianzhong Ding, Alberto Cerpa, and Wan Du. Multi-zone hvac control with model-based deep reinforcement learning. *arXiv preprint arXiv:2302.00725*, 2023.

[68] A. Standard. Standard 55-2004-thermal environmental conditions for human occupancy. *ASHRAE Inc*, 2004.

[69] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[70] Mesut Avci, Murat Erkoc, Amir Rahmani, and Shihab Asfour. Model predictive hvac load control in buildings using real-time electricity pricing. *Energy and Buildings*, 2013.

[71] Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*. Elsevier, 2013.

[72] Claudia Chiţu, Grigore Stamatescu, and Alberto Cerpa. Building occupancy estimation using supervised learning techniques. In *IEEE ICSTCC*, 2019.

[73] Grigore Stamatescu, Alex Beltran, and Alberto Cerpa. Data-driven comfort models for user-centric predictive control in smart buildings. In *ACM BuildSys*, 2016.

[74] Thomas C Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Multirobot routing algorithms for robots operating in vineyards. *IEEE Transactions on Automation Science and Engineering*, 17(3):1184–1194, 2020.

[75] David V Gealy, Stephen McKinley, Menglong Guo, Lauren Miller, Stavros Vougioukas, Joshua Viers, Stefano Carpin, and Ken Goldberg. Date: A handheld co-robotic device for automated tuning of emitters to enable precision irrigation. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 922–927. IEEE, 2016.

[76] Julian Fulton, Michael Norton, and Fraser Shilling. Water-indexed benefits and impacts of california almonds. *Ecological indicators*, 96:711–717, 2019.

[77] 2018 Almond Board of California. Water footprint for almonds. `https://almonds.com/sites/default/files/2020-05/Water_footprint_plus_almonds.pdf`.

[78] Almond Board of California. Almond irrigation improvement continuum. `https://www.almonds.com/sites/default/files/2020-02/Almond-Irrigation-Improvement-Continuum.pdf`.

[79] GL Grabow, IE Ghali, RL Huffman, et al. Water application efficiency and adequacy of et-based and soil moisture–based irrigation controllers for turfgrass irrigation. *Journal of irrigation and drainage engineering*, 2013.

[80] Xianzhong Ding and Wan Du. Drlic: Deep reinforcement learning for irrigation control. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 41–53. IEEE, 2022.

[81] Yenny Fernanda Urrego-Pereira, Antonio Martínez-Cob, and Jose Cavero. Relevance of sprinkler irrigation time and water losses on maize yield. *Agronomy Journal*, 2013.

[82] Dilini Delgoda, Hector Malano, Syed K Saleem, and Malka N Halgamuge. Irrigation control based on model predictive control (mpc): Formulation of theory and validation using weather forecast data and aquacrop model. *Environmental Modelling & Software*, 2016.

[83] Camilo Lozoya, Carlos Mendoza, Leonardo Mejía, Jesús Quintana, Gilberto Mendoza, Manuel Bustillos, Octavio Arras, and Luis Solís. Model predictive control for closed-loop irrigation. *IFAC Proceedings Volumes*, 2014.

[84] Bruno Silva Ursulino, Suzana Maria Gico Lima Montenegro, Artur Paiva Coutinho, et al. Modelling soil water dynamics from soil hydraulic parameters estimated by an alternative method in a tropical experimental basin. *Water*, 2019.

[85] California department of water resources. `https://www.cimis.water.ca.gov/`.

[86] Daniel A Winkler, Miguel Á Carreira-Perpiñán, and Alberto E Cerpa. Plug-and-play irrigation control at scale. In *ACM/IEEE IPSN*, 2018.

[87] Haoyu Niu, Dong Wang, and YangQuan Chen. Estimating actual crop evapotranspiration using deep stochastic configuration networks model and uav-based crop coefficients in a pomegranate orchard. In *Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping V*. International Society for Optics and Photonics, 2020.

[88] Xianzhong Ding and Wan Du. Smart irrigation control using deep reinforcement learning. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 539–540. IEEE, 2022.

[89] Xianzhong Ding and Wan Du. Optimizing irrigation efficiency using deep reinforcement learning in the field. *arXiv preprint arXiv:2304.01435*, 2023.

[90] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021.

[91] Devanshu Kumar, Xianzhong Ding, Wan Du, and Alberto Cerpa. Building sensor fault detection and diagnostic system. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 357–360, 2021.

[92] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[93] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A Preiss, Nora Ayanian, and Gaurav S Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *IEEE IROS*, 2019.

[94] Christopher Berner, Greg Brockman, Brooke Chan, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[95] Su Ki Ooi, Iven Mareels, Nicola Cooley, Greg Dunn, and Gavin Thoms. A systems engineering approach to viticulture on-farm irrigation. *IFAC Proceedings Volumes*, 2008.

[96] Guotao Cui and Jianting Zhu. Infiltration model based on traveling characteristics of wetting front. *Soil Science Society of America Journal*, 82(1):45–55, 2018.

[97] Guotao Cui and Jianting Zhu. Prediction of unsaturated flow and water backfill during infiltration in layered soils. *Journal of Hydrology*, 557:509–521, 2018.

[98] Dilini Delgoda, Syed K Saleem, Hector Malano, and Malka N Halgamuge. Root zone soil moisture prediction models based on system identification: Formulation of the theory and validation using field and aquacrop data. *Agricultural Water Management*, 2016.

[99] George H Hargreaves and Zohrab A Samani. Reference crop evapotranspiration from temperature. *Applied engineering in agriculture*, 1(2):96–99, 1985.

[100] Daniel A Winkler, Robert Wang, Francois Blanchette, et al. Magic: Model-based actuation for ground irrigation control. In *ACM/IEEE IPSN*, 2016.

[101] Eric Liang, Richard Liaw, Robert Nishihara, et al. Rllib: Abstractions for distributed reinforcement learning. In *ICML*. PMLR, 2018.

[102] Richard G Allen, Luis S Pereira, Dirk Raes, Martin Smith, et al. Crop evapotranspiration-guidelines for computing crop water requirements-fao irrigation and drainage paper 56. *Fao, Rome*, 1998.

[103] Hang Yu, Zhiping Jia, Lei Ju, Chunguang Liu, and Xianzhong Ding. Energy efficient routing algorithm using software defining network for wsns via unequal clustering. In *Geo-Spatial Knowledge and Intelligence: 4th International Conference on Geo-Informatics in Resource Management and Sustainable Ecosystem, GRMSE 2016, Hong Kong, China, November 18-20, 2016, Revised Selected Papers, Part II 4*, pages 154–163. Springer, 2017.

[104] Akshay Murthy, Curtis Green, Radu Stoleru, Suman Bhunia, Charles Swanson, and Theodora Chaspari. Machine learning-based irrigation control optimization. In *ACM BuildSys*, 2019.

[105] Light and frequent irrigation. `https://www.usga.org/course-care/water-resource-center/our-experts-explain--water/is-it-better-to-irrigate-light-and-frequent-or-deep-and-infreque.html`.

[106] Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto Cerpa, and Wan Du. Hpc-gpt: Integrating large language model for high-performance computing. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 951–960, 2023.

[107] Le Chen, Xianzhong Ding, Murali Emani, Tristan Vanderbruggen, Pei-Hung Lin, and Chuanhua Liao. Data race detection using large language models. *arXiv preprint arXiv:2308.07505*, 2023.

[108] Francesco Fraternali, Bharathan Balaji, Dhiman Sengupta, Dezhi Hong, and Rajesh K Gupta. Ember: energy management of batteryless event detection sensors with deep reinforcement learning. In *ACM SenSys*, 2020.

[109] Zhihao Shen, Kang Yang, Zhao Xi, Jianhua Zou, and Wan Du. Deepapp: a deep reinforcement learning framework for mobile application usage prediction. *IEEE Transactions on Mobile Computing*, 2021.

[110] Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. Atpp: A mobile app prediction system based on deep marked temporal point processes. In *IEEE DCOSS*, 2021.

[111] Xianzhong Ding, Wan Du, and Alberto E Cerpa. Mb2c: Model-based deep reinforcement learning for multi-zone building control. In *Proceedings of the 7th ACM international conference on systems for energy-efficient buildings, cities, and transportation*, 2020.

[112] Zelin Du, Qianling Zhang, Mao Lin, Shiqing Li, Xin Li, and Lei Ju. A comprehensive memory management framework for cpu-fpga heterogenous socs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(4):1058–1071, 2022.

[113] Mao Lin, Keren Zhou, and Pengfei Su. Drgpum: Guiding memory optimization for gpu-accelerated applications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 164–178, 2023.

[114] Jörg Thalheim, Peter Okelmann, Harshavardhan Unnibhavi, Redha Gouicem, and Pramod Bhatotia. Vmsh: hypervisor-agnostic guest overlays for vms. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 678–696, 2022.

[115] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual

machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[116] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean: Vm allocation service at scale. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 845–861, 2020.

[117] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman. Implementation and performance analysis of various vm placement strategies in cloudsim. *Journal of Cloud Computing*, 4:1–21, 2015.

[118] György Dósa and Jirí Sgall. First fit bin packing: A tight analysis. In *30th International symposium on theoretical aspects of computer science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[119] Chi Trung Ha, Trung Thanh Nguyen, Lam Thu Bui, and Ran Wang. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In *Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part II 20*, pages 140–155. Springer, 2017.

[120] Francisco Parreño, Ramón Alvarez-Valdés, Jose Manuel Tamarit, and Jose Fernando Oliveira. A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, 20(3):412–422, 2008.

[121] Gurobi solver. `https://www.gurobi.com/`.

[122] Cplex optimizer. `https://www.ibm.com/analytics/cplex-optimizer`.

[123] 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.

[124] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288. 2019.

[125] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of network and computer applications*, 52:11–25, 2015.

[126] Qingpeng Cai, Will Hang, Azalia Mirhoseini, George Tucker, Jingtao Wang, and Wei Wei. Reinforcement learning driven heuristic optimization. *Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD)*, abs/1906.06639, 2019.

[127] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method, 2017.

[128] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 1386–1394, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[129] Ye Xia, Mauricio Tsugawa, Jose AB Fortes, and Shigang Chen. Large-scale vm placement with disk anti-colocation constraints using hierarchical decomposition and mixed integer programming. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1361–1374, 2016.

[130] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

[131] A. Paul Davies and Eberhard E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operational Research*, 114(3):509–527, May 1999.

[132] Xijun Li, Mingxuan Yuan, Di Chen, Jianguo Yao, and Jia Zeng. A data-driven three-layer algorithm for split delivery vehicle routing problem with 3d container loading constraint. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, page 528–536, New York, NY, USA, 2018. Association for Computing Machinery.

[133] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 741–749. AAAI Press, 2021.

[134] Qianwen Zhu, Xihan Li, Zihan Zhang, Zhixing Luo, Xialiang Tong, Mingxuan Yuan, and Jia Zeng. Learning to pack: A data-driven tree search algorithm for large-scale 3d bin packing problem. In *Proceedings of the 30th ACM International Conference on Information Knowledge Management*, CIKM '21, page 4393–4402, New York, NY, USA, 2021. Association for Computing Machinery.

[135] Ameer Haj-Ali, Qijing Jenny Huang, John Xiang, William Moses, Krste Asanovic, John Wawrzynek, and Ion Stoica. Autophase: Juggling hls phase orderings in random forests with deep reinforcement learning. *Proceedings of Machine Learning and Systems*, 2:70–81, 2020.

[136] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[137] Marc Etheve, Zacharie Alès, Côme Bissuel, Olivier Juan, and Safia Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 176–185. Springer, 2020.

[138] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020.

[139] Haoran Sun, Wenbo Chen, Hui Li, and Le Song. Improving learning to branch via reinforcement learning. 2020.

[140] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.

[141] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.

[142] Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. *arXiv preprint arXiv:2205.11107*, 2022.

[143] Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33:20012–20023, 2020.

[144] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3243–3250, 2020.

[145] Meng Qi, Mengxin Wang, and Zuo-Jun Shen. Smart feasibility pump: Reinforcement learning for (mixed) integer programming. *arXiv preprint arXiv:2102.09663*, 2021.

[146] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 521–537, 2021.

[147] Kubernetes scheduler. `https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/`.

[148] Matthias Elf, Carsten Gutwenger, Michael Jünger, and Giovanni Rinaldi. Branch-and-cut algorithms for combinatorial optimization and their implementation in abacus. In *Computational Combinatorial Optimization*, pages 157–222. Springer, 2001.

[149] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[150] Desik Rengarajan, Gargi Vaidya, Akshay Sarvesh, Dileep Kalathil, and Srinivas Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration. *arXiv preprint arXiv:2202.04628*, 2022.

[151] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[152] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[153] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[154] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805v2*, 2018.

[155] Xumiao Zhang Xianshang Lin Pan Hu Yunfei Ma Songwu Lu Wan Du Z. Morley Mao Ennan Zhai Dennis Cai Yifei Xu, Yuning Chen. A practical benchmark for cloud native yaml configuration generation. 2023.

[156] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[157] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg

Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[158] Miaomiao Liu, Sikai Yang, Wyssanie Chomsin, and Wan Du. Real-time tracking of smartwatch orientation and location by multitask learning. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 120–133, 2022.

[159] Miaomiao Liu, Kang Yang, Yanjie Fu, Dapeng Wu, and Wan Du. Driving maneuver anomaly detection based on deep auto-encoder and geographical partitioning. *ACM Transactions on Sensor Networks*, 19(2):1–22, 2023.

[160] Jingwei Zhang, Bin Zi, and Xiaoyu Ge. Attend2pack: Bin packing through deep reinforcement learning with attention. *ArXiv*, abs/2107.04333, 2021.

[161] Dongda Li, Changwei Ren, Zhaoquan Gu, Yuexuan Wang, and Francis Lau. Solving packing problems by conditional query learning, 2020.

[162] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[163] Facebook AI Research. Pytorch: An open source machine learning framework. `https://pytorch.org/`, 2019. Accessed: April 23, 2023.

[164] Vincent Mai, Kaustubh Mani, and Liam Paull. Sample efficient deep reinforcement learning via uncertainty estimation. In *International Conference on Learning Representations*, 2022.

[165] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.