

Active Reinforcement Learning for Robust Building Control

Austin Jang



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-101

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-101.html>

May 11, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Thank you to my advisor Costas Spanos for guiding me through this whole process. Thank you to all the friends who supported me in this really rough year, especially Jennifer Zhou, who made sure I didn't skip too many meals. And finally, I would never have gotten here without my friend Dr. Lucas Spangher, who introduced me to RAISE Lab and research in RL and energy in general.

Active Reinforcement Learning for Robust Building Control

by

Doseok Jang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Costas Spanos, Chair

Professor Avidoh Zakhor

Spring 2023

Active Reinforcement Learning for Robust Building Control

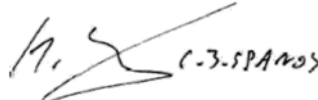
Doseok Jang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee



Professor Costas Spanos
Research Advisor

5/11/2023

(Date)

* * * * *



Professor Avidoh Zakhor
Second Reader

5/11/23

(Date)

Active Reinforcement Learning for Robust Building Control

Copyright 2023
by
Doseok Jang

Abstract

Active Reinforcement Learning for Robust Building Control

by

Doseok Jang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Costas Spanos, Chair

Reinforcement learning (RL) is a powerful tool for optimal control that has found great success in Atari games, the game of Go, robotic control, and building optimization. RL is also very brittle; RL agents often overfit to their training environment and fail to generalize to new settings. Unsupervised environment design (UED) has been proposed as a solution to this problem, in which the agent trains in environments that have been specially selected to help it learn. However, previous UED algorithms focus on trying to train an RL agent that generalizes across a large distribution of environments. This is not desirable when we wish to prioritize performance in one environment over others. For example, we will be examining the setting of robust RL building control, where we wish to train an RL agent that prioritizes performing well in normal weather while still being robust to extreme weather conditions. We demonstrate a novel UED algorithm, ActiveRL, that uses uncertainty-aware neural network architectures to generate new training environments at the edge of the RL agent's ability while being able to prioritize performance in a desired base environment. We show that ActiveRL is able to outperform state-of-the-art UED algorithms in minimizing energy usage while maximizing occupant comfort in the setting of building control.

To my parents.

Contents

Contents	ii
List of Figures	iv
List of Tables	vi
1 Introduction	1
1.1 Overview of HVAC Setpoint Control	1
1.2 Overview of Uncertainty Estimation in Deep Learning	3
1.3 Related Work and Gaps	3
1.4 Contributions	5
1.5 Roadmap	5
2 An Overview of Deep Reinforcement Learning	7
2.1 Overview	7
2.2 Markov Decision Processes	8
2.3 Classical Reinforcement Learning	8
2.4 Deep Reinforcement Learning	9
3 Active Reinforcement Learning	11
3.1 Overview	11
3.2 Proximal Policy Optimization	11
3.3 Uncertainty Estimation	13
3.4 Generating Environments	14
3.5 Prioritized Level Replay	16
4 Experiments to Assess Robustness of ActiveRL to Extreme Weather Conditions	20
4.1 Overview	20
4.2 Sinergym Environment for Simulating HVAC Control in Buildings . .	21
4.3 Evaluating ActiveRL’s Robustness to Extreme Weather Events	27

4.4	Evaluating ActiveRL’s Generalization to US Weather Conditions . . .	28
4.5	Evaluating ActiveRL’s Generalization from Sim2Real	30
4.6	Ablations Exploring Components of ActiveRL	30
5	ActiveRL Experimental Results	32
5.1	Overview	32
5.2	ActiveRL is Robust to Extreme Weather Events	32
5.3	ActiveRL Generalizes to US Weather Conditions	33
5.4	ActiveRL Generalizes from Simulation to Reality (Sim2Real)	36
5.5	Ablations Exploring Components of ActiveRL	39
6	Conclusion and Future Work	41
6.1	Conclusion	41
6.2	Limitations and Future Work	41
6.3	Roadmap to Deployment	42
	Bibliography	44

List of Figures

2.1	Flow of information in a typical RL setup, taken from [63]	7
3.1	An illustration of ActiveRL and ActivePLR. A. The overall flow of data in the ActiveRL training procedure. B. The ActiveRL environment selection process. C. How ActiveRL can be integrated into the PLR framework to select environments through ActivePLR.	12
4.1	The Sinergym environment. A. A breakdown of the Sinergym environment. A simulated building sends sensor data as observations to an RL agent, which responds with HVAC setpoints as actions, and is rewarded according to energy use and thermal comfort. B. We conduct three experiments using the Sinergym environment: evaluating our agents in hand-crafted extreme weather conditions, in a Sinergym simulation with higher fidelity than during training to simulate the Sim2Real jump, and in realistic weather conditions sampled from across the US.	24
5.1	Performance of each algorithm on training an RL HVAC agent, tested on various weather patterns. ActiveRL and ActivePLR outperform all baselines. We report the standard error of the mean over 5 trials for each result. A. Average reward achieved by each algorithm on the base environment ϕ_0 throughout training. B. Average reward achieved by each algorithm averaged over all 6 environments throughout training. C. Average reward achieved by each algorithm in each of the 5 extreme weather environments throughout training. Note that none of the algorithms were trained on these specific extreme weather environments.	34
5.2	Improvement in reward achieved by using ActiveRL instead of each baseline, on 120 randomly sampled weather patterns from across the US. A higher number here indicates that ActiveRL performs well in comparison to that baseline. On the other hand, a higher number indicates the baseline performs poorly.	35

- 5.3 Results of Sim2Real experiments. All error bars represent the standard error of the mean over 5 trials. **A.** Each controller is trained on a building simulation with $dt = 1$ hour and evaluated on a simulation with higher fidelity $dt = 0.25$ hours for the base environment configuration ϕ_0 and five handcrafted extreme weather scenarios. The rightmost bars show the average performance of each algorithm over all 6 scenarios. **B.** The average drop in reward over all environments when evaluating in the higher fidelity simulation compared to evaluation in the lower fidelity simulation. Lower is better here. 38
- 5.4 **Ablation A.** We explore how the γ regularization parameter affects the performance of ActiveRL. Higher values of γ mean that ActiveRL is forced to propose training environment configurations ϕ that are close to the base environment ϕ_0 . The left graph shows the average reward obtained throughout training on the base environment ϕ_0 . The right graph shows the reward averaged over ϕ_0 and the 5 extreme weather environments. ActiveRL seems quite sensitive to γ . **B.** Similarly, we explore how the η learning rate parameter affects ActiveRL. η is the learning rate used by ActiveRL to conduct gradient descent on ϕ . Higher values of η means a coarser-grained search for the ϕ that maximizes the agent’s uncertainty. ActiveRL seems insensitive to η once it gets small enough (< 0.1). 40

List of Tables

4.1	Bounds for Sinergym environment configuration variables	26
4.2	Relevant hyperparameters	27
4.3	Hyperparameter sweep ranges	28

Acknowledgments

Thank you to my advisor Costas Spanos for guiding me through this whole process. Thank you to all the friends who supported me in this really rough year, especially Jennifer Zhou, who made sure I didn't skip too many meals. And finally, I would never have gotten here without my friend Dr. Lucas Spangher, who introduced me to RAISE Lab and research in RL and energy in general.

Chapter 1

Introduction

Reinforcement learning (RL) is a powerful tool that has found great success in solving complex, sequential decision-making tasks like the game of Go [61], Atari games [49], StarCraft [58], and many others. However, RL agents often overfit to their training environment and fail to generalize to new environments. This is a serious issue in tasks where we expect the environment not to stay static; when there is some distribution shift between the training environment and the test environment. For example, we will be exploring the use of RL in building control, where the agent is often trained to optimize performance in normal weather conditions, but fails when tested in extreme weather conditions. **The overall goal of this project is to use uncertainty to select training environments such that the resulting trained RL agent is robust to uncommon but dangerous scenarios in the test environment without sacrificing overall performance, through applications to RL HVAC control.**

1.1 Overview of HVAC Setpoint Control

One application of robust RL that we will focus on is the use of RL for building control in response to weather conditions and occupant schedules. With the advent of the Internet of Things and 21st century modernization, buildings now have a glut of sensory information that can be tapped into to maximize occupant comfort and increase energy efficiency. Smart buildings can be equipped with sensors to detect things like temperature, airflow, humidity, occupancy, light, and energy usage [28]. Because buildings account for 40% of primary energy consumption, 73% of electricity usage, and 40% of greenhouse gas emissions in the US [1], all this new sensor data represents a valuable opportunity to reduce human energy consumption in the transition to a sustainable future.

At the same time, smart buildings also offer the opportunity to provide more comfortable conditions for occupants. Although humans spend over 90% of their time in buildings, studies have shown that only 40% of commercial building occupants are satisfied with their thermal conditions [25]. Smarter building control can directly impact the comfort and productivity of building occupants.

There are several avenues through which buildings can be automatically controlled, such as heating, ventilation, and air conditioning (HVAC) units, energy storage systems, plug-in electric vehicles, photovoltaic power sources, and lights [24]. Within the 40% of energy consumption that buildings are responsible for, almost a third is HVAC [68], so that is the avenue of control that we will focus on in this project.

Traditionally, HVAC setpoint control has been approached through model-predictive control (MPC, [2, 36]) or a heuristic rule-based-controller (RBC, [46]). MPC is generally not scalable to high dimensional input or output spaces compared to RL, and is only as good as the model it is based on. RBCs are brittle and inflexible compared to RL algorithms; for example, an RBC deployed in an office building with a rule to always turn off the HVAC systems at night would fail to produce a comfortable environment in the event that someone works late. A data-driven approach, however, would eventually learn to turn the HVAC systems back on if there is an occupant late at night in order to maximize their thermal comfort.

Recently, RL-based HVAC setpoint control has grown in popularity. [57] used a specially formulated Q-learning algorithm to control HVAC setpoints in the presence of unseen disturbances. [42, 37] demonstrated how using RL and meta-RL could be used to train an RL HVAC agent that could quickly adapt to different buildings. In terms of trying to train an RL agent that is robust to different weather patterns, [75] demonstrated an RL HVAC controller beating a RBC baseline in several simulated buildings and climates. However, they noted that the RL controller is not robust to changes in indoor air temperature setpoint schedules, and needed special reward-shaping to deal with varying weather conditions.

As climate change continues, we will experience more droughts and heat waves, stronger and more intense hurricanes, general rising temperatures, and more frequent cold snaps[45]. All of these extreme weather events likely represent a tiny portion of the training data, but are the time when building controls must work the hardest to ensure safe, comfortable conditions for occupants. For example, an RL controller that was not prepared for these scenarios may raise the heat during a heat wave, or lower it during a cold snap, which are both unacceptable for occupant comfort.

There are an infinite number of these kinds of extreme scenarios, such that it is impossible for an engineer to enumerate them all. Thus there is a need for methods to automatically discover realistic extreme scenarios that the RL agent needs more data about.

1.2 Overview of Uncertainty Estimation in Deep Learning

At the same time, the need to understand where a neural network based model’s predictions are suboptimal in applications such as medicine, self-driving, and anything safety-critical, has spurred research into methods to accurately estimate the uncertainty of neural networks. By having a measure of the model’s uncertainty, we can tell when to trust the model or disregard its output. In this paper, we use the model’s uncertainty as a signal to identify what scenarios that our RL HVAC agent needs more data about. There are several ways in which one might quantify the uncertainty of a neural network. [22] showed how to use Monte-Carlo dropout to leverage a regularization technique that is often already incorporated in deep neural net training to estimate the uncertainty of a deep model by posing the training process as an approximation to training a deep Gaussian Process. [38] demonstrated how one could train multiple versions of the same model as an ensemble, and estimate the uncertainty of the ensemble by looking at its variance. Using a Bayesian neural network architecture also allows uncertainty estimation [66]. We use Monte Carlo dropout in this paper primarily because training Bayesian neural networks is significantly slower and more computationally demanding, and dropout is simpler to implement than a bootstrapped ensemble.

1.3 Related Work and Gaps

Uncertainty-Driven RL Exploration

Neural network uncertainty has been used before to facilitate exploration of RL algorithms [72, 47], so that they collect data that helps them generalize over more environments. For example, [52] takes uncertainty into account by estimating a distribution on Q values instead of a point estimate and using this to drive efficient exploration in the face of uncertainty. One of the most popular approaches is to provide an uncertainty-based exploration bonus to the Q function to encourage exploration [39, 51, 9]. However, exploration is not enough to ensure robustness in environments where agents’ actions are not enough to get to any state in the MDP. An RL HVAC control system cannot take actions to ”explore” what would happen if there was a heat wave because its actions do not control important aspects of the state like outdoor weather.

Active Learning

The process of automatically selecting what data to sample is sometimes called active learning, or optimal experiment design. These algorithms were designed to identify experimental parameters that could generally provide as much information about the search space as possible. However, most of them were designed for supervised learning. For example, [13, 21] attempt to find regions of uncertainty in the data distribution by looking at misclassification rates and output entropy. [44] uses conformal prediction to quantify the similarity of new data points to their dataset. EVOP [43] uses the sequential simplex method in order to identify experiment configurations that can maximize information gain. [6] use random exploration to identify new, promising data samples. Some of these concepts are already in use in many RL algorithms; for example, the RL algorithm we use in this paper PPO [60], is incentivized to explore new data samples via random exploration and increasing the output action distribution entropy.

Unsupervised Environment Design (UED)

However, the setting we are going to explore is slightly different than active learning, in that we are not directly selecting data *points* to sample, but selecting the environment parameters that impact which data points will emerge as the RL agent explores the environment. This is a more helpful problem setting in RL as RL algorithms work best with on-policy data that is collected by having the RL agent explore the environment, instead of just labeling sets of data samples with a "correct" action. In order to explore how to change environment parameters to investigate generalization of RL algorithms across diverse settings [56, 11, 12], Procedural Content Generation (PCG) environments have emerged as a promising category of challenges. These environments are procedurally generated according to some chosen configuration parameters via some Unsupervised Environment Design (UED, [17]) algorithm, which constructs an implicit curriculum of training environments at the edge of the RL agent's capabilities. For example, [53] and [17] find adversarial but feasible environment configurations with high regret to help the agent generalize. [32] re-sample previously seen environments based on their 1-step TD error: a metric of the learning potential of those environments. These algorithms focus on training an RL agent that performs well across a distribution of similar tasks. This may be undesirable in scenarios where the test environment is significantly different from the training distribution of environments that the UED algorithm provides. For example, if we are training an RL HVAC agent for a building in Arizona, our first priority is for the agent to perform well in the normal weather for Arizona, and our second priority is for it to be robust to different weather events that may arise due to climate change.

It is also possible that unrestricted UED may result in training environments that are completely unrealistic, for example simulating the RL building control problem for a building on the sun. Furthermore, none of them take advantage of the continuous nature of some environment configuration variables to optimize them via gradient descent.

1.4 Contributions

We present a novel uncertainty-based, gradient-based algorithm for UED in building control called ActiveRL. To the best of our knowledge, this is the first time neural network uncertainty has been incorporated into the problem of UED; most current works focus on some form of regret. To the best of our knowledge, this is also the first time environment configuration variables have been directly optimized under gradient descent rather than through some evolutionary process [53], resampling procedure [32], or training a separate teacher network to select new environments [17].

We demonstrate how ActiveRL trains RL HVAC controllers that are (1) more performant overall, (2) more robust to handcrafted extreme weather conditions, (3) more robust to a larger variety of more realistic weather conditions sampled from across the US, and (4) more robust to the Sim2Real transfer than the current state-of-the-art in UED. (1), (2), and (3) are important contributions that make RL HVAC control more effective at ensuring that the indoor environments we spend 90% of our time in are comfortable, while still reducing energy consumption. (4) shows that it is realistic to train an RL agent in simulation like we do here with ActiveRL and deploy it in the real world, with only a slight performance drop compared to other methods.

In this project, we propose a new UED algorithm that, with some reasonable assumptions on the structure of the environment, leverages uncertainty-aware neural networks to train an RL agent that is robust to extreme weather conditions without sacrificing overall performance.

1.5 Roadmap

We will first give a primer on the field of reinforcement learning in chapter 2 discussing the underlying framework of Markov Decision Processes, classical RL algorithms, and deep RL algorithms. We will then describe the methods of our paper in chapter 3, where we will describe all the different components of our novel ActiveRL algorithm. In chapter 4, we will provide a description of three different experiments we run in order to assess whether ActiveRL is successfully able to train an RL HVAC controller that is robust to extreme weather patterns and can handle the Sim2Real jump. In

chapter 5 we will show the results of those experiments. Finally, we will conclude with a discussion of future work and limitations in chapter 6.

Chapter 2

An Overview of Deep Reinforcement Learning

2.1 Overview

Before we dive into describing our new algorithm, we would first like to provide a technical description of reinforcement learning, its terminologies, and the types of problems it seeks to solve. We will first discuss the general class of problems that RL algorithms seek to solve, then describe some classical RL algorithms that will lead into a discussion of modern deep RL techniques using neural network function approximators.

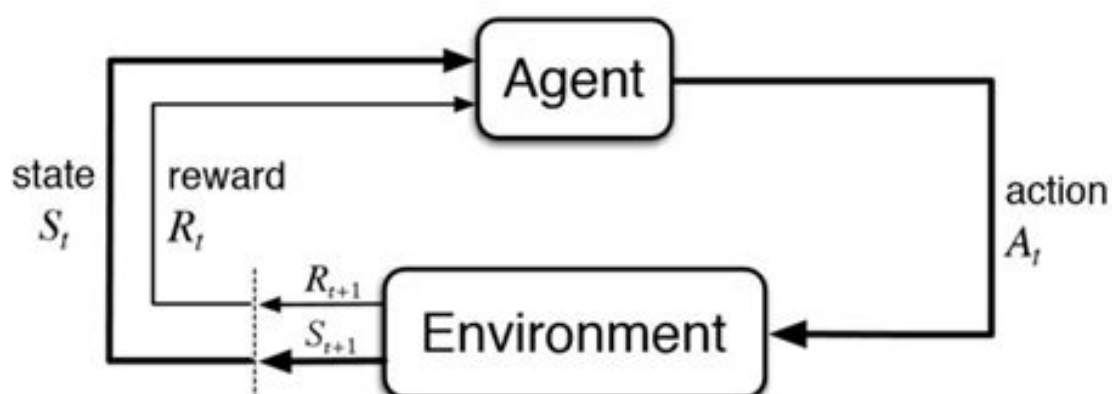


Figure 2.1: Flow of information in a typical RL setup, taken from [63]

2.2 Markov Decision Processes

RL algorithms generally attempt to solve sequential decision-making problems in the form of Markov Decision Processes (MDPs) [63]

$$M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{S}_0) \quad (2.1)$$

that are composed of a state space $\mathcal{S} \subseteq \mathbb{R}^n$, action space $\mathcal{A} \subseteq \mathbb{R}^m$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, transition probability matrix $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, and a probability distribution over initial states $\mathcal{S}_0 \in \mathcal{S}$.

An initial state is first sampled from \mathcal{S}_0 . At each timestep, an action $a \in \mathcal{A}$ can be taken in response to the observed state $s \in \mathcal{S}$, and a reward $\mathcal{R}(s, a)$ is computed based on those states and actions. Given the current state s and action a , the next state s' is sampled from the probability distribution $\mathcal{P}(s, a)$. In order to make decisions in this space, we define a policy $\pi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps states to actions, such that $a = \pi(s)$. The overall goal of RL is to identify a policy π so as to maximize an objective function J : the expected discounted sum of rewards:

$$J = \mathbb{E}_{\mathcal{P}, \pi, \mathcal{S}_0} \left[\sum_{t=0}^T \gamma^t \cdot \mathcal{R}(s_t, a_t = \pi(s_t)) \right] \quad (2.2)$$

Here, γ is a discount factor that forces π to care more about rewards that arrive sooner rather than later.

2.3 Classical Reinforcement Learning

If $|\mathcal{S}|$ and $|\mathcal{A}|$ are small enough, then the MDP can be easily solved through classical RL algorithms such as value iteration or policy iteration.

In order to describe value iteration and policy iteration, we will first define the value function $V : \mathcal{S} \rightarrow \mathbb{R}$ as a measure of the expected discounted sum of rewards, given that the agent is at a certain state s , and the Q function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ which computes the same expected discount sum of rewards, but given that the agent is at a certain state s and has taken a certain action a .

$$Q(s, a) = \mathbb{E}_{\mathcal{P}, \pi} \left[\sum_{t=0}^T \gamma^t \cdot \mathcal{R}(s_t, a_t = \pi(s_t)) \middle| s_0 = s, a_0 = a \right] \quad (2.3)$$

$$V_\pi(s) = \mathbb{E}_{\mathcal{P}, \pi} \left[\sum_{t=0}^T \gamma^t \cdot \mathcal{R}(s_t, a_t = \pi(s_t)) \middle| s_0 = s \right] \quad (2.4)$$

In order to estimate the value function of the optimal policy, one can start with arbitrary Q and V tables and iteratively estimate

$$\hat{Q}^k(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \sim \mathcal{P}(s'|s, a)} V^{k-1}(s') \quad (2.5)$$

$$\hat{V}^k(s) = \max_a(Q(s, a)) \quad (2.6)$$

for iterations $k = 1 \dots K$, and take the optimal policy $\pi = \operatorname{argmax}_a(Q(s, a))$.

Alternatively, one could optimize the policy directly instead of trying to estimate the optimal value function by using policy iteration. The policy is initialized as some random policy and is then iteratively refined by first evaluating the policy – computing $V_\pi(s)$ and $Q_\pi(s, a)$ for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$ – and then setting $\pi(s)$ for each state to $a = \operatorname{argmax}_{a'} Q(s, a')$. Calculating $V_\pi(s)$ and $Q_\pi(s, a)$ is done in the same manner as in value iteration, except $\hat{V}^k(s) = \mathbb{E}_\pi(Q^k(s, a))$

2.4 Deep Reinforcement Learning

Policy iteration and value iteration can be efficiently computed using dynamic programming for small state spaces and actions, but becomes intractable for large \mathcal{S} and \mathcal{A} . They also require knowledge of the transition probability matrix \mathcal{P} which is not feasible for many real-world RL problems. In practice, RL practitioners will train neural networks to approximate Q_θ , V_θ , and/or π_θ , where θ are the parameters of the neural network.

No matter what the neural network is attempting to predict, neural networks need data to make predictions. In RL, this data is usually collected by deploying the policy π (which is either directly π_θ or derived from Q_θ) into the MDP, and collecting tuples of the form $(s_t, a_t, s_{t+1}, r_{t+1})$ of the states, actions, and rewards that the agent observed in the MDP. We denote a full trajectory $[(s_0, a_0, s_1, r_1), \dots, (s_{T-1}, a_{T-1}, s_T, r_T)]$ as τ . Additionally, we will denote $\mathcal{R}(\tau) = \sum_{t=0}^{T-1} \gamma^t \cdot \mathcal{R}(s_t, a_t = \pi(s_t))$.

RL methods that only estimate π_θ are often referred to as policy-gradient methods. The most common approach to using a neural network to represent a policy is to have it output a probability distribution over actions. With a discrete action space, this is as simple as just outputting a probability for each of the possible actions. With a continuous action space, it is common to parameterize the output of the neural network as the mean and standard deviation of a Normal distribution [26] from which the action is sampled. The most basic variation of policy-gradient methods is REINFORCE [70], which updates the policy to maximize the following objective function:

$$J_{PG} = \mathcal{R}(\tau) \log(\pi_\theta(a_t|s_t)) \quad (2.7)$$

$$\theta_\pi = \theta_\pi - \eta \nabla_{\theta_\pi} J \quad (2.8)$$

where η is the learning rate. This directly attempts to optimize the expected reward under the policy π_θ . Similarly to policy iteration, we optimize our policy π after evaluating its value through trajectories τ .

Deep Q-learning methods [27, 67] only approximate the Q function, usually optimizing the mean squared error loss between Q_θ and some estimate of the Q value taken from data. Say we roll out a trajectory τ and collect tuples (s, a, s', r) . Then we can compute the loss L_Q as follows:

$$Q_{target} = r + \gamma \max_{a'} Q_\theta(s', a') \quad (2.9)$$

$$L_Q = (Q_\theta - Q_{target})^2 \quad (2.10)$$

The neural network Q_θ can then be optimized through standard gradient descent methods to minimize L_Q .

$$\theta_Q = \theta_Q - \eta \nabla_{\theta_Q} L_Q \quad (2.11)$$

. Just like value iteration, deep Q learning estimates the value of each state and action, and given Q_θ , we can extract a policy $\pi_Q(s) = \operatorname{argmax}_{a'} Q_\theta(s, a')$.

Similarly, V_θ can be trained to approximate the value function by reformulating the target:

$$V_{target} = r + \gamma V_\theta(s') \quad (2.12)$$

$$L_V = (V_\theta - V_{target})^2 \quad (2.13)$$

$$\theta_V = \theta_V - \eta \nabla_{\theta_V} L_V \quad (2.14)$$

Value networks V_{θ_V} are often trained together with a policy network π_{θ_π} in actor-critic methods like Advantage Actor Critic (A2C) [48] and Proximal Policy Optimization (PPO) [60], the RL algorithm we use in this paper. V_{θ_V} is often used to reduce the variance in policy network updates by estimating the advantage function

$$A(s, a) = \mathbb{E}_{s' \sim P(s'|s, a)} [R(s, a) + \gamma V_{\theta_V}(s')] - V_{\theta_V}(s) \quad (2.15)$$

For example, A2C directly substitutes A in for $\mathcal{R}(\tau)$ to compute its objective:

$$J_{A2C} = A(s, a) \log(\pi_{\theta_\pi}(s, a)) \quad (2.16)$$

We will go into detail about PPO in section 3.2.

Chapter 3

Active Reinforcement Learning

3.1 Overview

Our overall goal is to train RL agents that are robust to conditions that may not appear frequently in their training distribution. For example, an RL agent trained to manage a building’s HVAC controls should still perform reasonably when it encounters weather patterns that are underrepresented in its training data. In order to do this, we will present an uncertainty-based active learning algorithm that identifies which environments may have the highest learning potential.

First, we will describe the general framework of RL, then the RL algorithm we use in our experiments, then the method by which we estimate the uncertainty of the RL agent, followed by a novel approach to using that uncertainty to generate environment configurations that maximize learning. Finally, we will describe Prioritized Level Replay (PLR, [32]), the state-of-the-art baseline against which we compare our algorithm, and explore how our approach can be integrated into the PLR framework.

3.2 Proximal Policy Optimization

The RL algorithm we use in this paper is Proximal Policy Optimization (PPO)[60] due to its performance and ease of use. It has become the default algorithm for many use cases. For example, it is the default algorithm for OpenAI. In the building control setting, many works [8, 5, 74] have used PPO for HVAC control. Although we focus on PPO in this paper for these reasons, our algorithm should easily extend to any actor-critic RL algorithm.

PPO trains two neural networks: a policy network $\pi_{\theta_\pi} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps states $s \in \mathbb{R}^n$ to actions $a \in \mathbb{R}^m$, and a critic network $V_{\theta_V} : \mathbb{R}^n \rightarrow \mathbb{R}$ that maps states s to values. The latter approximates the value function $V(s)$.

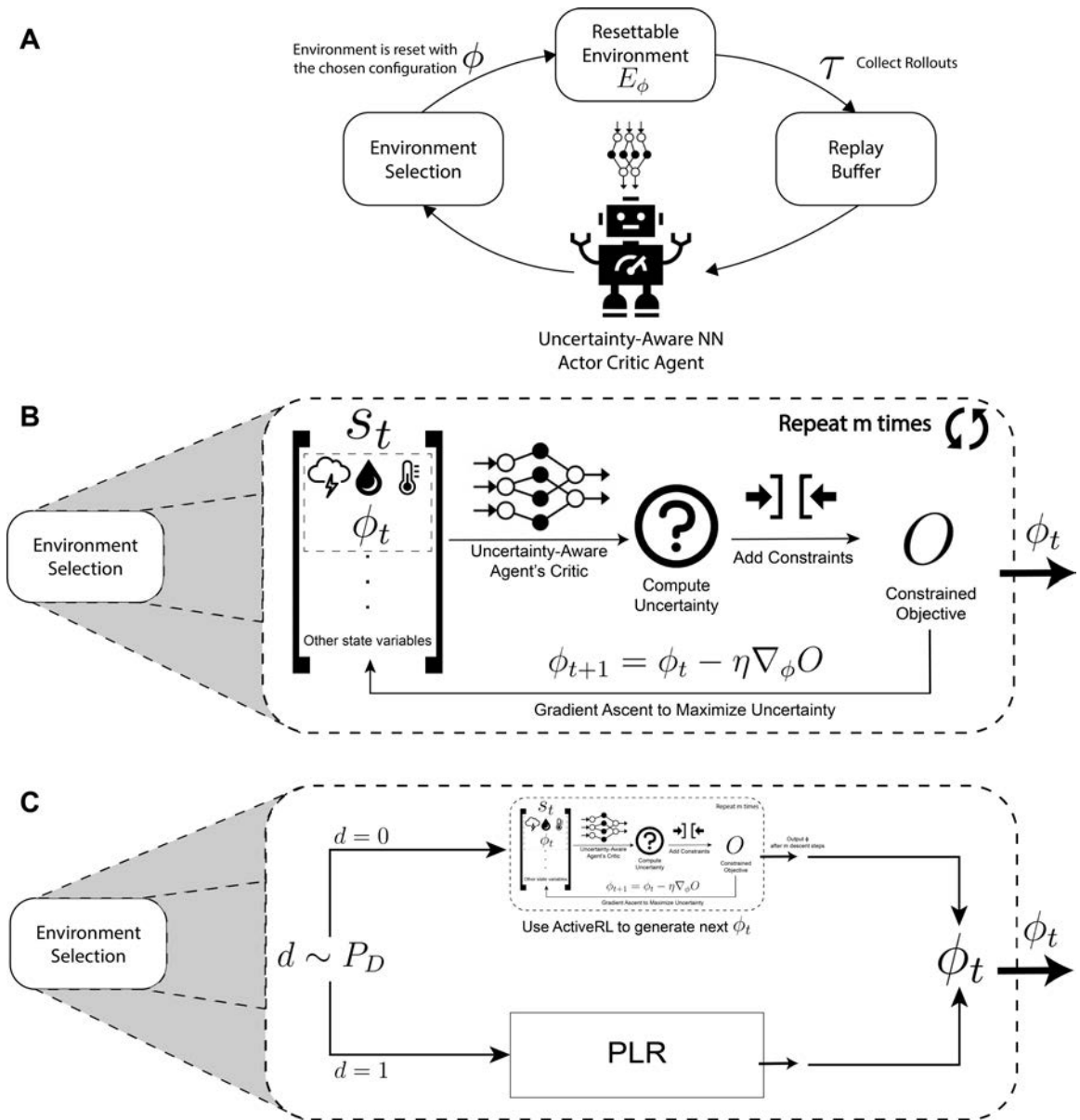


Figure 3.1: An illustration of ActiveRL and ActivePLR. **A**. The overall flow of data in the ActiveRL training procedure. **B**. The ActiveRL environment selection process. **C**. How ActiveRL can be integrated into the PLR framework to select environments through ActivePLR.

The actor is trained by first interacting with the environment and collecting data about the states it observed s_t , actions it took a_t , and rewards it received r_t . The actor is then updated through gradient descent to optimize the surrogate objective function

$$J = \hat{\mathbb{E}}_t[\min(z_t(\theta)\hat{A}_t, \text{clip}(z_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)], \quad (3.1)$$

$$\text{where } \hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{(T-t+1)}\delta_{T-1} \quad (3.2)$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (3.3)$$

$$z_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (3.4)$$

z_t is the ratio of the probability of the action a_t under the current policy to its probability under the policy that originally collected the data, and \hat{A}_t is an estimate of the advantage $Q(s, a) - V(s)$ obtained using generalized advantage estimation [59]. The minimization of this surrogate objective enables the use of multiple gradient updates per data sample, allowing PPO to enjoy higher sample efficiency than previous on-policy algorithms.

The critic network is trained to approximate the value function $V(s)$, and its outputs are used to compute \hat{A}_t to train the actor above. The critic is updated through gradient descent to minimize the loss:

$$V_{target} = r + \gamma V_\theta(s') \quad (3.5)$$

$$L = (V_\theta(s) - V_{target})^2 \quad (3.6)$$

3.3 Uncertainty Estimation

In order to estimate the uncertainty of our RL agent, we use Monte Carlo Dropout [22]. Dropout is a common regularization technique for deep neural networks [62] to avoid overfitting, in which nodes in the neural network are set to zero ("dropped out") at random. Traditionally, nodes are only dropped out at training time, and dropout is not conducted at inference time. [22] introduced Monte Carlo Dropout, in which dropout is also used at inference time to generate multiple predictions for an individual input. The variance in these predictions is then used as a measure of the model's uncertainty. We use Monte Carlo Dropout as opposed to other methods of estimating neural network uncertainty like bootstrapped ensembles [38] or Bayesian neural networks [66] because Bayesian neural networks are difficult and computationally expensive to train, and Monte Carlo Dropout is easier to implement and cheaper to train than bootstrapped ensembles while still providing good quantifications of uncertainty.

Formally, suppose that we have a neural network $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps n -dimensional input vectors to m -dimensional output vectors. f is parameterized by a set of l weight matrices $\theta = W_i|_{i=1}^l$, where $W_i \in \mathbb{R}^{(N_{i-1} \times N_i)}$ denotes the weight matrix for layer i of the neural network, and N_i denotes the number of neurons in that layer. Let us denote the dropout operation by d_p such that $d_p(\theta) : \mathbb{R}^k \rightarrow \mathbb{R}^k$ is a stochastic operation that sets each column in each W_i in θ to 0 with probability p . We can then define the uncertainty L for an input $x \in \mathbb{R}^n$ as:

$$L(x, \theta) = \text{Var}(f_{d_p(\theta)}(x)) \quad (3.7)$$

Empirically, we can estimate this as

$$L(x, \theta) = \frac{1}{C} \sum_{c=1}^T f_{d_p(\theta)}(x)^T f_{d_p(\theta)}(x) - \mathbb{E}(f_{d_p(\theta)}(x))^T \mathbb{E}(f_{d_p(\theta)}(x)) \quad (3.8)$$

Essentially, we conduct C independent stochastic forward passes through the model with dropout at inference time, and use the sample variance of the outputs as our uncertainty metric.

To estimate the uncertainty of our RL agent, we estimate the uncertainty of the critic network, similar to other works [3, 71].

3.4 Generating Environments

One key assumption we make in the design of this algorithm is that, during training, we are interacting with a Procedural Content Generation (PCG) environment [56]. A PCG environment is one that can take on different algorithmically created configurations at the beginning of each training episode; for example, a PCG physics simulation environment might be able to take on different gravity constants, friction values, etc. .

We also assume that at least some of the PCG environment configuration variables are continuous, and changeable.

In order to generate new environments at the frontier of the agent's uncertainty, we will backpropagate gradients from the uncertainty back to the state variable and do gradient descent. This will allow us to find the state at which the agent is most uncertain, and generate a new PCG environment that allows the agent to interact with the world at that state.

Formally, assume that we have a PCG environment E with some parameters $\phi \in \mathbb{R}^k$ that are part of the agent's initial state space S . That is, the state $s_0 \in \mathbb{R}^n$ can be divided into ϕ and $\bar{s}_0 := s_0 \setminus \phi \in \mathbb{R}^{n-k}$. We can define an objective function

that tries to maximize the uncertainty of the RL agent:

$$O(\phi_i, s_0, \theta) = -L([\phi, \bar{s}_0], \theta) \quad (3.9)$$

where $[\]$ is the concatenation operator and L computes our uncertainty estimate. We can then update ϕ :

$$\phi_{i+1} = \phi_i - \eta \nabla_{\phi_i} O(\phi_i, s_0, \theta) \quad (3.10)$$

We call this procedure Active Reinforcement Learning (ActiveRL), as it is an active learning method that seeks to identify what data would be most useful for the RL agent.

Trying to identify parameters ϕ by attempting to maximize uncertainty can often lead to unrealistic parameters that are outside of the test distribution, and not as useful for learning. Thus, we integrate both hard constraints and soft constraints on the ϕ generation process in ActiveRL. The hard constraints are useful when one is trying to train an RL agent that generalizes over a certain region of the ϕ space, and the soft constraints are useful when one is trying to train an RL agent that emphasizes performance near a particular ϕ_0 , which is still robust to different values of ϕ .

The hard constraints constrain the search space within some lower and upper limits specified by the user for ϕ using the extragradient [35] method. Suppose we have a lower bound constraint $\phi > b$ for some $b \in \mathbb{R}^k$ and an upper bound constraint $\phi < a$ for some $a \in \mathbb{R}^k$. Then we can express the Lagrangian as

$$\mathcal{L}(\phi, s_0, \theta, \lambda, \mu) = O(\phi, s_0, \theta) + \sum_i \lambda_i (b_i - \phi_i) + \sum_i \mu_i (\phi_i - a_i) \quad (3.11)$$

Now we can express the extragradient update. First, define the joint variable $\omega = (\phi, \lambda, \mu)$. We omit s_0 and θ from ω and the parameters to \mathcal{L} as they will be kept constant throughout the optimization process. Then, the extragradient optimization process can be described as:

$$F(\omega) = [\nabla_{\phi} \mathcal{L}(\omega), -\nabla_{\lambda} \mathcal{L}(\omega) - \nabla_{\mu} \mathcal{L}(\omega)]^T \quad (3.12)$$

$$\omega_{t+1/2} = P_{\Omega}[\omega_t - \eta F(\omega_t)] \quad (3.13)$$

$$\omega_{t+1} = P_{\Omega}[\omega_t - \eta F(\omega_{t+1/2})] \quad (3.14)$$

where $P_{\Omega}[\cdot]$ is the projection onto the constraint set. Throughout this paper, we will use the implementation of ExtragradientAdam from [23], which adjusts the learning rate η for each parameter according to the Adam algorithm [34]. This constrained optimization approach ensures that ϕ will stay within the specified bounds a and b , helping to avoid unrealistic values of ϕ .

In the setting where one is trying to train an RL agent that emphasizes performance near a particular ϕ_0 , which is still robust to different values of ϕ , hard

constraints are not enough as they provide no guarantee that states near ϕ_0 will be sampled. Thus we can introduce a soft constraint to the objective function that seeks to minimize the Euclidean distance from ϕ to ϕ_0 .

$$O(\phi_i, s_0, \theta) = -L([\phi, \bar{s}_0], \theta) + \gamma \|(\phi - \phi_0)\|_2 \quad (3.15)$$

where γ weights how much the soft constraint should be emphasized.

Pseudocode for ActiveRL can be found in section 3.4, and an illustration can be found in fig. 3.1.

Algorithm 1 ActiveRL

```

1: procedure ACTIVERL( $\theta, s_0, N, T, \eta, \gamma, a, b, p$ )
2:    $\triangleright \theta$ : policy parameters    $\triangleright s_0$ : initial state to seed environment generation    $\triangleright T$ :
   number of iterations to run PPO    $\triangleright n$ : number of iterations to optimize  $\phi$     $\triangleright \eta$ :
   Learning rate for optimizing  $\phi$     $\triangleright \gamma$ : Weight on soft constraint    $\triangleright a$ : Lower bounds
   of  $\phi$     $\triangleright b$ : Upper bounds of  $\phi$     $\triangleright p$ : Probability of sampling in the default
   environment  $\phi_0$ 
3:    $\phi_0 \leftarrow \text{ExtractPhi}(s_0)$ 
4:   for t=0 to T do
5:     for i=0 to N do
6:        $\phi \leftarrow \text{ExtractPhi}(s_0)$ 
7:        $O \leftarrow -\text{UncertaintyEstimate}(f_\theta, s_0) + \gamma \phi^T \phi_0$ 
8:        $\phi \leftarrow \text{ExtragradientUpdate}(\phi, O)$ 
9:        $s_0 = \text{Concatenate}([\phi, \bar{s}_0])$ 
10:    end for
11:     $\tau \leftarrow \text{PPOCollectTrajectories}(E_\phi)$ 
12:     $\theta \leftarrow \text{PPOUpdate}(\tau, \theta)$ 
13:  end for
14:  Return  $\theta$ 
15: end procedure

```

3.5 Prioritized Level Replay

Prioritized Level Replay (PLR, [32]) is a state-of-the-art algorithm in the domain of optimizing RL agents to generalize across PCG environments. PLR is a general framework for selectively sampling training levels ¹ in environments with procedu-

¹In our case, a "level" is just an environment configuration ϕ . We use the term "level" in describing PLR to be consistent with the original paper [32].

Algorithm 2 ActivePLR

```

1: procedure ACTIVEPLR( $\theta, s_0, N, T, \eta, \gamma, a, b, p$ )
2:    $\triangleright \theta$ : policy parameters  $\triangleright s_0$ : initial state to seed environment generation  $\triangleright T$ :
   number of iterations to run PPO  $\triangleright n$ : number of iterations to optimize  $\phi$   $\triangleright \eta$ :
   Learning rate for optimizing  $\phi$   $\triangleright \gamma$ : Weight on soft constraint  $\triangleright a$ : Lower bounds
   of  $\phi$   $\triangleright b$ : Upper bounds of  $\phi$   $\triangleright p$ : Probability of sampling in the default
   environment  $\phi_0$   $\triangleright c$ : Global episode counter  $\triangleright \Lambda_{seen}$ : Visited levels  $\triangleright S$ : Global
   level scores  $\triangleright C$ : Global level timestamps (when they were last sampled)
3:    $\phi_0 \leftarrow \text{ExtractPhi}(s_0)$ 
4:    $c \leftarrow c + 1$ 
5:   for  $t=0$  to  $T$  do
6:     Sample replay decision  $d \sim P_D$ 
7:     if  $d == 1$  then
8:       Sample  $\phi \sim (1 - \rho) \cdot P_S(\phi | \Lambda_{seen}, S) + \rho \cdot P_C(\phi | \Lambda_{seen}, C, c)$ 
9:     else
10:      for  $i=0$  to  $N$  do
11:         $\phi \leftarrow \text{ExtractPhi}(s_0)$ 
12:         $O \leftarrow -\text{UncertaintyEstimate}(f_\theta, s_0) + \gamma \phi^T \phi_0$ 
13:         $\phi \leftarrow \text{ExtragradientUpdate}(\phi, O)$ 
14:         $s_0 = \text{Concatenate}([\phi, \bar{s}_0])$ 
15:      end for
16:    end if
17:    Define new index  $i \leftarrow |S| + 1$ 
18:    Add  $\phi_i \leftarrow \phi$  to  $\Lambda_{seen}$ 
19:    Add initial value  $S_i = 0$  to  $S$  and  $C_i = 0$  to  $C$ 
20:     $\tau \leftarrow \text{CollectTrajectories}(E_\phi)$ 
21:    Update score  $S_i \leftarrow \text{PPOValueLoss}(\tau, \theta)$  and timestamp  $C_i \leftarrow c$ 
22:     $\theta \leftarrow \text{PPOUpdate}(\tau, \theta)$ 
23:  end for
24:  Return  $\theta$ 
25: end procedure

```

rally generated content. The key idea is to prioritize levels (environment configurations) with higher estimated learning potential when revisited in the future, thereby adapting the sampling of training levels and improving both sample efficiency and generalization performance.

To estimate the learning potential of a level, PLR utilizes the average magnitude of the GAE loss over latest trajectory over that level, which is equivalent to the L1 loss of the value estimator of PPO. The value loss provides an effective measure of the discrepancy between the agent’s current and target values, which can be used to estimate the potential for future learning. [32] pose the ”Value Correction Hypothesis”: the larger the value loss, the higher the learning potential of a given level.

In PLR, the sampling procedure is guided by the estimated learning potential of each level. Levels with higher value loss are prioritized, inducing an emergent curriculum of increasingly difficult levels. When deciding what environment configuration to train on, PLR first samples $d \sim P_D$, to decide whether to sample a new level from the training distribution Λ_{train} or pick one from the replay buffer Λ_{seen} . [32] parameterized P_D as a Bernoulli distribution with probability $p = \frac{|\Lambda_{\text{seen}}|}{|\Lambda_{\text{train}}|}$. Since we consider the setting where ϕ is continuous, $|\Lambda_{\text{train}}|$ is infinite. Thus we parameterize the denominator as a hyperparameter N_{PLR} , so $p = \frac{|\Lambda_{\text{seen}}|}{N_{PLR}}$.

The probability of each level in the replay buffer being sampled is determined by two components:

$$P_S(l_i|\Lambda_{\text{seen}}, S) = \frac{h(S_i)^{1/\beta}}{\sum_j h(S_j)^{1/\beta}} \quad (3.16)$$

where S_i is the L1 value loss, $h(S_i) = 1/\text{rank}(S_i)$ is a prioritization function, $\text{rank}(S_i)$ is the rank of S_i among all the scores of levels in the replay buffer in descending order, and

$$P_C(l_i|\Lambda_{\text{seen}}, C, c) = \frac{c - C_i}{\sum_{C_j \in C} c - C_j} \quad (3.17)$$

where c is the count of total episodes sampled so far during training, and C_i is the episode count at which level i was last used. P_S is a probability distribution that places higher weight on levels with higher value loss. P_C places higher weight on levels that have not been seen in a while, whose recorded value loss may be less accurate. Thus the final probability distribution over the replay buffer is

$$P_{\text{replay}} = (1 - \rho) \cdot P_S(l_i|\Lambda_{\text{seen}}, S) + \rho \cdot P_C(l_i|\Lambda_{\text{seen}}, C, c) \quad (3.18)$$

If we do not sample a new level from the replay buffer, we sample a new one from the training distribution. We consider two possibilities for this sampling procedure: sampling uniformly at random from the set of all possible training environments, and selecting a training environment that maximizes uncertainty via ActiveRL. We

will refer to PLR with the former approach as just PLR, and the latter approach as ActivePLR.

Pseudocode for ActiveRL can be found in section 3.4, and an illustration can be found in fig. 3.1.

Chapter 4

Experiments to Assess Robustness of ActiveRL to Extreme Weather Conditions

4.1 Overview

In order to assess the efficacy of ActiveRL in training an RL agent that is robust to extreme weather patterns, we conduct three experiments. These experiments use the OpenAI Gym [7] environment Sinergym [33], which provides a testbed for RL HVAC control agents and allows us to control the different weather conditions that those agents experience. Our first experiment tests whether or not the ActiveRL agent is robust to extreme weather patterns by evaluating its performances in different handcrafted extreme weather scenarios. Since these handcrafted environments may not be necessarily representative of real weather patterns, we will also assess the performance of the ActiveRL agent in controlling the HVAC for a building experiencing 120 different real weather patterns sampled from across the US in the second experiment. Finally, because our method and RL in general rely heavily on training in simulation, we evaluate whether the ActiveRL agent trained in simulation still performs well in the "real world" (which we simulate with a higher fidelity simulation).

In this chapter, we will first describe the Sinergym environment, then go into detail about the three experiments that we ran. The first experiment assesses each HVAC control agent on handcrafted extreme environments. The second experiment assesses each HVAC control agent on 120 different realistic weather patterns sampled from across the US. Finally, the third experiment assesses how well the HVAC control agents that were trained in the Sinergym simulation fare in the real world (approximated by a more accurate Sinergym simulation). A visual illustration of Sinergym

and the three experiments are shown in fig. 4.1.

4.2 Sinergym Environment for Simulating HVAC Control in Buildings

In order to test whether or not ActiveRL gives rise to more robust RL agents, we apply it to a realistic application: RL for heating, ventilation, and air conditioning (HVAC) control in the face of extreme weather events. There exists a rich body of work on the use of RL for HVAC control [73, 41, 16, 55], and it has high potential to help save energy while enhancing resident comfort at the same time. These RL systems take into account features of the world around them, such as the indoor and outdoor temperature, humidity, occupant count, etc. in order to control HVAC systems. However, as climate change continues, the frequency of extreme weather events and unusual weather conditions [31] will increase. Thus, it is imperative that we ensure our RL systems are robust to states that may have been uncommon in their training distribution, such as extreme droughts, storms, or cold snaps.

General Description

We use a modified version of the Sinergym [33] OpenAI Gym [7] environment to simulate buildings in different weather conditions. The environment utilizes the EnergyPlus [14] building energy simulation engine to model the dynamics of building systems, including HVAC, lighting, and other energy-consuming components in response to simulated occupants and weather. We use the "5Zone" building provided with Sinergym, which is a $463.6m^2$ single-story building equipped with a DX cooling coil and gas heating coils that is divided into 5 zones (1 indoor and 4 outdoor).

The action space in the Sinergym environment is a continuous, multi-dimensional vector, denoted as $\mathcal{A} \subseteq \mathbb{R}^m$, where m is the number of action variables. The action space defines the set of control decisions that can be made to influence the building's performance. For the 5Zone building, we use a two-dimensional action space, where the agent can control the heating and cooling setpoints for the HVAC systems.

We use a reward function that encourages the agent to find policies that both maximize occupant comfort and minimize energy use. To quantify occupant comfort, we use the Fanger Percentage of People Dissatisfied (PPD, [20]) metric predicted by EnergyPlus. Energy use is just the total HVAC electricity demand rate throughout the whole building in Watts. The reward can be formulated as:

$$R_t = -\rho * \lambda_E * P_t - (1 - \rho) * \lambda_P * PPD_t * \mathbb{1}_{(occupancy_t > 0)} * \mathbb{1}_{PPD_t > 20} \quad (4.1)$$

where ρ is a hyperparameter that controls how much to weight comfort against energy use, P_t is the electricity demand rate, λ_E and λ_P are scaling factors that ensure that the PPD units (%) are comparable to the power units (W), $\mathbb{1}_{(\text{occupancy}_t > 0)}$ is an indicator variable that ensures the agent is not penalized for uncomfortable conditions when there are no occupants, and $\mathbb{1}_{PPD_t > 20}$ ensures the agent is not penalized if the PPD is below the ASHRAE comfort threshold of 20% [4]. For our experiments, we use $\lambda_E = 0.0001$, $\lambda_P = 0.1$, $\rho = 0.5$.

The state space in the Sinergym environment is a continuous, multi-dimensional representation of the building's current conditions, capturing various aspects of its performance. The state space is denoted as $\mathcal{S} \subseteq \mathbb{R}^n$, where n is the number of state variables. Since we are using the "5Zone" building, the state space is 20 dimensional, and includes:

1. Indoor air temperature ($^{\circ}\text{C}$) for each thermal zone,
2. Relative humidity (%) for each thermal zone,
3. Outdoor air temperature* ($^{\circ}\text{C}$),
4. Outdoor relative humidity* (%),
5. Zone thermal comfort clothing value,
6. Zone thermal comfort,
7. Current HVAC setpoints,
8. Wind speed* and direction*,
9. Facility total HVAC electricity demand rate,
10. Solar irradiance* (W/m^2),
11. Occupancy count, and
12. The current date (year, month, day, hour)

Environment Configuration Parameters that Change Weather Patterns in Sinergym

In order to simulate the outdoor weather, Sinergym takes as input a file that contains hourly measurements of each of the outdoor weather variables denoted with a '*' above, as well as several others. Originally, Sinergym added noise to the measured

outdoor temperature through an Ornstein-Uhlenbeck (OU, [19]) process, to help prevent overfitting the RL agent to the static weather pattern. We modified Sinergym so that it could add this noise to the other weather variables denoted with a '*' as well. An OU process has three parameters: σ , μ , and τ . If we have a noise vector x_t , then

$$x_{t+1} = x_t + dt * (-(x_t - \mu)/\tau) + \sigma * \sqrt{\frac{2}{\tau}} Z \quad (4.2)$$

where $Z \sim Normal(0, 1)$. so σ controls the magnitude of the noise that is added, μ is the average value of the noise, and τ determines how quickly the noise reverts to the mean. Notably, if we have a recorded weather variable w_t , then adding the noise results in $Mean_{w_t+x_t} = Mean_{w_t} + \mu$. For each of our 5 weather variables, we estimate realistic values for σ and τ by doing linear regression of the difference between that weather variable and its moving average. That is, if we assume our recorded weather variable w_t was generated via adding noise generated through an OU process, then we can generate measurements $x_t = w_t - MA(w_t)$, where MA is the moving average. By applying linear regression onto the generated x_t 's, we can estimate values of σ and τ that will generate weather with a similar amount of noise to real weather conditions. A detailed description of this linear regression process is detailed in section 4.4.

Since we have reasonable values for σ and τ for each weather variable, we have 5 remaining parameters that can be used to customize the configuration of Sinergym: the μ offset parameters for each weather variable. Thus the ϕ for Sinergym that we will be varying to attempt to train a robust RL agent, is the 5 dimensional vector $\langle \mu_1, \mu_2, \mu_3, \mu_4, \mu_5 \rangle$. Essentially, we will be changing the average outdoor temperature, relative humidity, wind speed, wind direction, and solar irradiance over the course of the building simulation. Varying the environment configuration $\phi \in \mathbb{R}^5$ enables us to collect training data from different outdoor weather conditions.

Baseline Algorithms for HVAC Control

We will compare ActiveRL against several baseline algorithms to assess its efficacy in comparison to other HVAC control and PCG environment control algorithms. First, we will discuss RL-based baselines like PPO, domain-randomization with PPO, and PLR with PPO. Then we will discuss some simple Non-RL baselines like a random controller and a heuristic, rule-based controller.

RL Baselines

Our most basic RL baseline is a PPO agent composed of a standard neural network with two layers, each with 256 neurons, using dropout and ReLU activations, that is trained on only ϕ_0 .

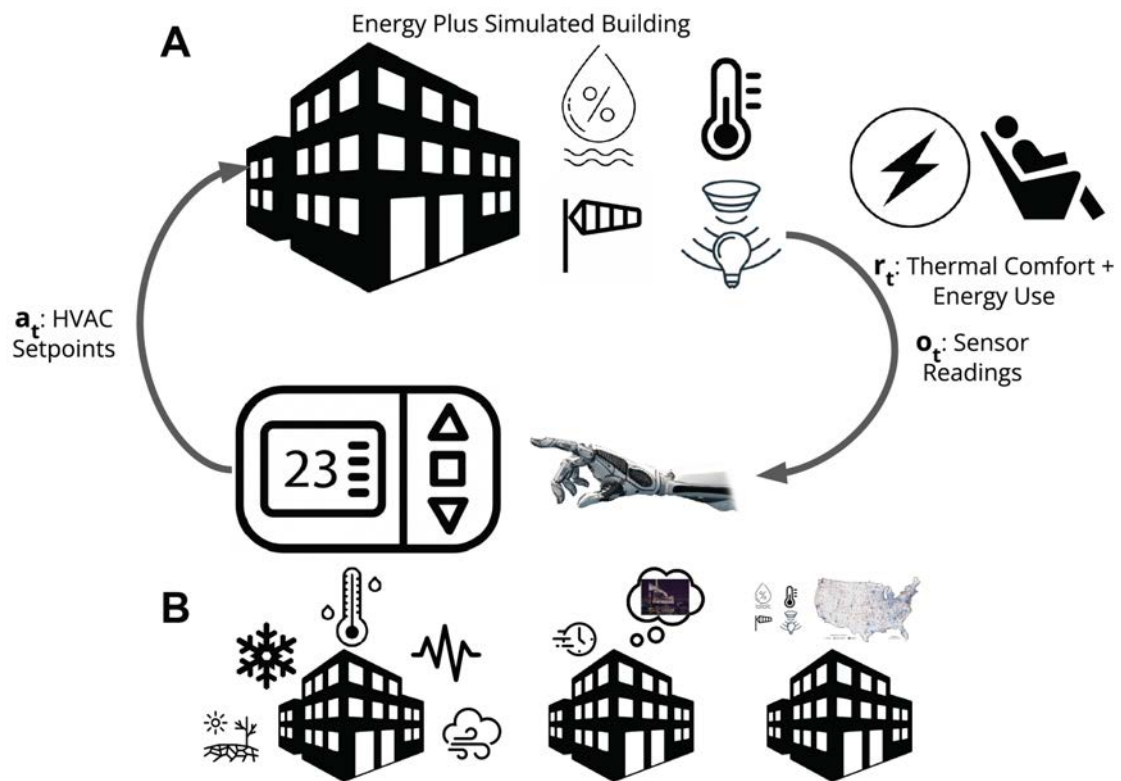


Figure 4.1: The Sinergym environment. **A.** A breakdown of the Sinergym environment. A simulated building sends sensor data as observations to an RL agent, which responds with HVAC setpoints as actions, and is rewarded according to energy use and thermal comfort. **B.** We conduct three experiments using the Sinergym environment: evaluating our agents in handcrafted extreme weather conditions, in a Sinergym simulation with higher fidelity than during training to simulate the Sim2Real jump, and in realistic weather conditions sampled from across the US.

The most common method of training agents that generalize across PCG environments is domain randomization, where environment configurations are just selected uniformly at random. This method is frequently used to ensure that agents trained in simulation can effectively transfer from simulation to the real world [10, 65, 64]. In the buildings domain, [30, 18] used domain randomization to train an RL energy pricing agent that would be robust to the Sim2Real transfer. In our setting: we have some lower bounds $a \in \mathbb{R}^5$ and upper bounds $b \in \mathbb{R}^5$ for each variable, described in table 4.1. We simply sample a $\phi \sim Uniform(a, b)$.

Our last RL-based baseline, PLR, is described in section 3.5.

Non-RL Baselines

In addition to the Domain Randomization, RL, and PLR baselines, we also use a simple rule-based controller (RBC) based on the one that is provided with Sinergym as a baseline, and a random controller.

The random controller simply outputs a cooling setpoint at random $a[0] \sim Uniform(22.5, 30.0)$ and a random heating setpoint $a[1] \sim Uniform(15, 22.5)$. The specific values are taken from Sinergym.

The RBC sets the heating setpoint to $26^\circ C$ and the cooling setpoint to $29^\circ C$ during the summer. During the winter, it sets heating to $20^\circ C$ and cooling to $23.5^\circ C$. This part of the RBC is the same as the RBC provided by Sinergym, and generally follows the philosophy of setting the desired temperature range higher in the summer and lower in the winter to minimize energy consumption, since it is more difficult to cool the building during the summer due to high outdoor temperatures, and difficult to heat the building during the winter due to low temperatures.

We added an additional rule that if no occupants are detected in the building, the RBC sets heating to $0^\circ C$ and cooling to $50^\circ C$; this is a wide enough range that the HVAC system is essentially turned off during these times. We added this new occupancy-based rule for the sake of a fair comparison with a realistic RBC because we included occupancy information in the reward.

Implementation Details

Unless specified otherwise, all the experiments we run are simulated with a timestep dt of 1 hour. That is, EnergyPlus simulates the building dynamics with a timestep of 1 hour, and the RL agent gets to change the temperature setpoints each hour. In section 5.4 we will explore how we estimate the Sim2Real gap by increasing the EnergyPlus simulation fidelity by decreasing its timestep length dt to 0.25hours during evaluation.

For ActiveRL, ActivePLR, Domain Randomization, and PLR, we bound our search to between the lower and upper bounds for each of the environment configuration variables provided by Sinergym. We provide a table of these bounds in table 4.1.

We pick the base weather configuration ϕ_0 to be one of Sinergym’s default weather patterns. This weather pattern is based on a Typical Meteorological Year (TMY) recording of hourly meteorological data that spans 1 year (8760 hours) of weather from Davis-Monthan, Arizona. We believed that an RL controller trained in this hot, dry environment would likely have problems adapting to extreme weather events like cold snaps that we could help solve with ActiveRL. More details about the TMY data can be found section 4.4.

For all experiments except for the US weather experiment described in section 4.4, we run each algorithm over 5 random seeds. In section 4.4 we use the random seeds which got the median average reward for each algorithm in the handcrafted extreme weather evaluation described in section 4.3 due to computational constraints.

Table 4.1: Bounds for Sinergym environment configuration variables

Weather Variable	Lower Bound	Upper Bound
Outdoor Air Temperature (°C)	-31.05	60.7
Outdoor Air Relative Humidity (%)	3	100
Outdoor Wind Speed (m/s)	0.0	23.1
Outdoor Wind Direction (°)	0	360
Direct Solar Radiation Rate (W)	0	1033

¹Note that all other algorithms share these hyperparameters

²We use the default parameters from [32] for the PLR part of ActivePLR because our hyperparameter sweep showed that none of the PLR-specific hyperparameters made much of a difference for ActivePLR. Other hyperparameters are the same as ActiveRL

Table 4.2: Relevant hyperparameters

Algorithm	Hyperparameters
PPO ¹	lr= 0.00005, clip_param= 0.3, discount_factor= 0.8, $p_{dropout} = 0.1$, # of inner SGD steps= 40
ActiveRL	$\gamma = 0.5$, $\eta = 0.01$, $p_{dropout} = 0.1$, $N = 91$, $C = 10$
PLR	$\rho = 0.045$, $\beta = 0.0015$, $N_{PLR} = 10$
ActivePLR ²	$\rho = 0.1$, $\beta = 0.1$, $N_{PLR} = 100$

4.3 Evaluating ActiveRL’s Robustness to Extreme Weather Events

In order to evaluate how robust agents trained by each algorithm are to extreme weather events, we evaluate the agent as it trains in a suite of 5 different extreme weather environments, parameterized by 5 different ϕ weather configurations. The agent is trained on a mix of the base environment ϕ_0 and automatically generated environments, depending on the algorithm. It is then evaluated in the following environments:

- ϕ_1 simulates an extremely hot and dry drought
- ϕ_2 simulates a wet and windy storm
- ϕ_3 simulates a humid heatwave
- ϕ_4 simulates a cold snap
- ϕ_5 simulates erratic weather

Our hypothesis is that using uncertainty to identify new environments to collect data from will allow us to train RL agents that are more robust to extreme weather conditions.

In order to find hyperparameters that worked well for each method, we first conducted a hyperparameter sweep over parameters for PPO, such as the clipping threshold, number of internal update steps, and learning rate. We then conducted a hyperparameter sweep for each of the different UED algorithms. The sweep ranges are

detailed in table 4.3, and the final values in table 4.2. A detailed exploration of how hyperparameters interact with ActiveRL can be found in section 5.5.

Table 4.3: Hyperparameter sweep ranges

Algorithm	Hyperparameters included in Sweep
PPO ³	lr $\in \{0.0005, 0.00005, 0.000005\}$, clip_param $\in \{0.1, 0.2, 0.3\}$, discount_factor $\in \{0.8, 0.9, 0.99\}$, # of inner SGD steps $\in \{20, 30, 40\}$
ActiveRL	$\gamma \in \{0, 0.0005, 0.005, 0.05, 0.5\}$, $\eta \in [e^{-10}, 1]$, $p_{dropout} \in \{0.1, 0.25, 0.5\}$, $N \in [1, 100]$
PLR	$\rho \in [e^{-8}, 1]$, $\beta \in [e^{-8}, 1]$, $N_{PLR} \in \{10, 50, 100, 200\}$
ActivePLR	$\gamma \in \{0, 0.0005, 0.005, 0.05, 0.5\}$, $\eta \in [e^{-10}, 1]$, $p_{dropout} \in \{0.1, 0.25, 0.5\}$, $N \in [1, 100]$, $\rho \in [e^{-8}, 1]$, $\beta \in [e^{-8}, 1]$, $N_{PLR} \in \{10, 50, 100, 200\}$

4.4 Evaluating ActiveRL’s Generalization to US Weather Conditions

Since we handcrafted the extreme weather environments in section 4.3, it is possible that these environments are unrealistic. In order to properly assess the viability of our RL HVAC controller in a range of different weather scenarios, we constructed a dataset of 120 randomly sampled, recorded weather patterns from across the US. We deployed the HVAC controller for ActiveRL and each baseline in a building that simulated each of those 120 weather patterns.

To construct the dataset of 120 weather patterns, we first scraped the EnergyPlus weather data website⁴ to get recorded weather patterns from across the US. These were Typical Meteorological Year (TMY) weather patterns [69] from NREL, which contain hourly meteorological information⁵ from specific weather stations over the

³Note that all other algorithms share these hyperparameters

⁴<https://energyplus.net/weather>

⁵e.g. temperature, humidity, wind, etc.

course of 1 year (8760 hours). This meteorological information is specially collated from multiple historical recordings of the weather data in that location to present the range of weather phenomena that typically occur there, while still keeping to annual averages that are consistent with long term averages for that location. TMY weather data is used often for building simulations.

After we obtained a dataset of historical weather data recordings, we converted them into a realistic dataset of environment configuration parameters ϕ . We modeled each weather variable in each weather pattern as a variation generated by an OU process (eq. (4.2)) from the corresponding weather variable in the base environment configuration ϕ_0 . Formally, let us suppose we have some recorded weather variable $y \in \mathbb{R}^{8760}$, corresponding to the value of that weather variable for each hour in a year. We also have a recording corresponding to the base environment $\phi_0, y^0 \in \mathbb{R}^{8760}$. Since Sinergym takes the parameters of an OU process as its environment configuration, we model the difference $x_t = y_t - y_t^0$ as having been generated from an OU process, like in eq. (4.2). We rearrange the terms in eq. (4.2) as:

$$x_{t+1} = \left(1 - \frac{dt}{\tau}\right)x_t + \frac{\mu dt}{\tau} + \sigma * \sqrt{\frac{2}{\tau}}Z \quad (4.3)$$

$$x_{t+1} = mx_t + b + E \quad (4.4)$$

where $m = \left(1 - \frac{dt}{\tau}\right)$, $b = \frac{\mu dt}{\tau}$, and $E = \sigma * \sqrt{\frac{2}{\tau}}Z$. We can then run linear regression to find what parameters m and b estimate x_{t+1} from x while minimizing the error term E . Once we have estimated m and b with linear regression, we can compute the residual error $E = x_{t+1} - mx_t - b$ and compute the standard deviation of E as an estimate for $\sigma * \sqrt{\frac{2}{\tau}}$. Finally, we can estimate $\tau = \frac{dt}{1-m}$, $\mu = \frac{b\tau}{dt}$, $\sigma = \frac{\sqrt{Var(E)}}{\sqrt{\frac{2}{\tau}}}$. We then repeat this process for each of the 120 US weather patterns, for each of the 5 weather variables that compose the environment configuration: outdoor humidity, air temperature, wind speed, wind direction, and solar irradiance. Thus we have a dataset⁶ $X \in \mathbb{R}^{120 \times 5 \times 3}$ of environment configurations⁷ that Sinergym can take in and simulate.

Our hypothesis is that by conducting an uncertainty-driven environment exploration that is constrained to realistic environments, ActiveRL will be able to generalize to different weather patterns across the US better

⁶The final 3 dimension comes from the fact that we have 3 variables μ, σ, τ for the OU process for each weather variable

⁷Note that the environment configuration variables that go into ActiveRL, PLR, or Domain Randomization $\phi \in \mathbb{R}^5$ are a subset of the full $\mathbb{R}^{5 \times 3}$ environment configuration that can be provided to Sinergym, as ϕ only contains the offset parameters μ .

than the baseline methods. In particular, our hypothesis for Domain Randomization and PLR is that they will end up training the RL algorithm to focus on performing well in unrealistic environments, and cause performance to degrade on this set of more realistic environments.

4.5 Evaluating ActiveRL’s Generalization from Sim2Real

One flaw in this work and UED algorithms in general is that a simulator is required to train the model in different environments. Thus it is important to ask whether or not the RL policies trained in simulation can be extended to the real world.

In order to test this, we evaluated each RL algorithm in each of the 6 environments by running the EnergyPlus simulator at a higher fidelity than the agents were trained on, thus simulating the ”Sim2Real” jump. The agents were trained on a simulator operating at a granularity of $dt = 1$ hour per timestep, and we evaluate on a granularity of $dt = 0.25$ hours per timestep. During evaluation, each of the RL agents’ actions are simply repeated four times so that it still takes an action every hour.

The Sim2Real problem occurs because modeling errors in the low fidelity training simulation will cause the vanilla RL agent to learn a policy based on inaccurate environment dynamics. When the RL agent is deployed in the ”real world” or a higher fidelity simulation, it will make control errors that compound to cause the agent to take actions that will take it out of state space supported by the training data distribution, which will cause further performance degradation. **Our hypothesis is that by increasing the state space supported by the training data distribution, ActiveRL will help the agent be more robust to compounding errors caused by the Sim2Real jump.**

4.6 Ablations Exploring Components of ActiveRL

It is common in machine learning papers to provide an idea of how important each component of one’s method is to its overall performance by performing ”ablation” experiments. These ablation experiments remove or modify components or hyperparameters in the algorithm in order to better understand how those components and hyperparameters affect the overall performance.

In order to further understand the driving factors behind the performance of ActiveRL, we conducted two ablation experiments. First, we explored the impact of

the γ soft constraint term. Second, we explored the impact of the learning rate η on the performance of ActiveRL.

Constraints

First, we explore the necessity of constraining ActiveRL from generating environment configurations ϕ that are too far away from ϕ_0 . γ shows up in eq. (3.15), as a coefficient that regulates how much the distance of the generated environment configuration ϕ from the base environment ϕ_0 contributes to the objective function of ActiveRL. As γ increases, the algorithm is encouraged to generate values of ϕ that are closer to ϕ_0 . We varied γ between four different values $\{0, 0.005, 0.05, 0.5\}$ while keeping the other hyperparameters the same as our other experiments to better understand how the algorithm performs under different constraint strengths.

Our hypothesis with this experiment was that there would be a tradeoff between performance in the extreme environments, and performance in the base environment ϕ_0 that would be modulated by γ .

Learning Rate

The learning rate η determines the step-size used by the Adam optimizer when ActiveRL is conducting gradient descent on ϕ . The smaller η is, the more fine-grained the search for an uncertain environment configuration becomes. We mainly explore this hyperparameter to assess how sensitive ActiveRL is to the user's choice of η ; if there are many values of η that yield optimal performance, then ActiveRL becomes much easier to use for other problems. We varied η between five different values $\{0.0001, 0.001, 0.01, 0.1, 1.0\}$ while keeping the other hyperparameters the same as our other experiments.

Our hypothesis for this experiment was that there would be some optimal value of η that yielded the best performance by striking the perfect balance between being large enough to avoid local minima, and being small enough to actually converge.

Chapter 5

ActiveRL Experimental Results

5.1 Overview

In this chapter, we will present the results of the experiments described in chapter 4.

First, we will discuss how well the ActiveRL agent performs in HVAC control on handcrafted extreme environments. Then we will assess its performance on 120 different realistic weather patterns sampled from across the US. Finally, we will estimate the performance that actually deploying an ActiveRL trained HVAC controller on the real world might have. Finally, we will explore what effect that different hyperparameters have on ActiveRL.

5.2 ActiveRL is Robust to Extreme Weather Events

In order to evaluate how robust agents trained by each algorithm are to extreme weather events, we evaluate the agent over the course of the training process, in a suite of 5 different extreme weather environments, parameterized by 5 different ϕ weather configurations. The agent is trained on either the base environment ϕ_0 (vanilla RL) or automatically generated environments (Domain Randomization, PLR, ActiveRL, ActivePLR), depending on the algorithm. It is then evaluated in the following environments:

ϕ_1 simulates an extremely hot and dry drought, ϕ_2 simulates a wet and windy storm, ϕ_3 simulates a humid heatwave, ϕ_4 simulates a cold snap, and ϕ_5 simulates erratic weather.

Our hypothesis is that using uncertainty to identify regions of the state space to collect data from will allow us to train RL agents that are more

robust to extreme weather conditions.

Visualizations of the performance of each algorithm in each environment can be seen in fig. 5.1.

Surprisingly, Domain randomization and PLR did not have significant improvements over the vanilla RL algorithm. By the end of training, domain randomization and PLR achieved 9% higher reward than the RBC on the base environment ϕ_0 after 3M timesteps of training. However, the vanilla RL policy had a 9% improvement over the domain randomization and PLR policies with ϕ_0 . Over the 5 extreme weather environments, domain randomization only did about as well as the RBC. This may mean that the environments generated using domain randomization were too unrealistic to be useful for learning how to perform in extreme weather conditions. Over the extreme weather conditions, PLR did beat domain randomization and the RBC, but still performed worse than the vanilla RL policy. This suggests that, because PLR uses domain randomization to sample new environments, it still suffers from generating unrealistic environments. However, its prioritized environment resampling procedure helps it generalize across all environments better than naive domain randomization, even though it is resampling from unrealistic environments.

We found that training the HVAC controller with ActiveRL resulted in agents that performed better in both the extreme environments and the base environment. Generally, ActiveRL and ActivePLR performed similarly. In three out of the five extreme environments, the hot drought, cold and windy, and cold snap environments, ActiveRL performed significantly better than all other baselines and performed competitively in the remaining two environments. In the base environment, ActiveRL and ActivePLR provide a 9% improvement over vanilla RL, and a 24% improvement over RBC. Over all 6 environments, it provides a 3% improvement over vanilla RL. The fact that ActiveRL significantly outperforms all baselines indicates that there is considerable value in seeking out realistic new training environments that maximize an agent’s uncertainty rather than choosing environments at random or merely replaying old ones.

5.3 ActiveRL Generalizes to US Weather Conditions

Although the ActiveRL agent seems to perform well in these handcrafted extreme weather conditions, it is possible that these environments are unrealistic. In order to properly assess the viability of our RL HVAC controller in a range of different weather scenarios, we deployed the HVAC controller for ActiveRL and each baseline in a building that simulated each of those 120 different weather patterns sampled

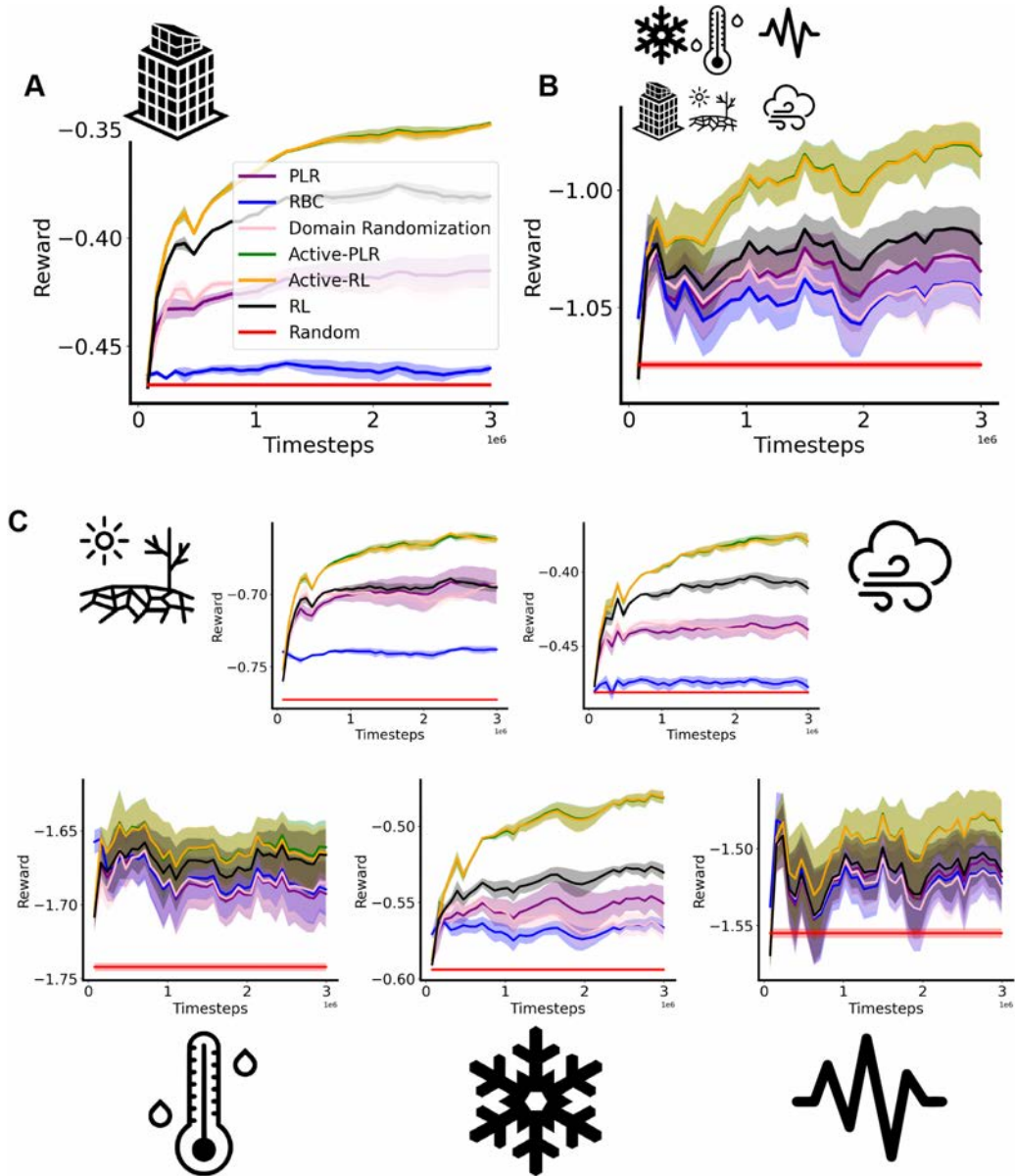


Figure 5.1: Performance of each algorithm on training an RL HVAC agent, tested on various weather patterns. ActiveRL and ActivePLR outperform all baselines. We report the standard error of the mean over 5 trials for each result. **A.** Average reward achieved by each algorithm on the base environment ϕ_0 throughout training. **B.** Average reward achieved by each algorithm averaged over all 6 environments throughout training. **C.** Average reward achieved by each algorithm in each of the 5 extreme weather environments throughout training. Note that none of the algorithms were trained on these specific extreme weather environments.

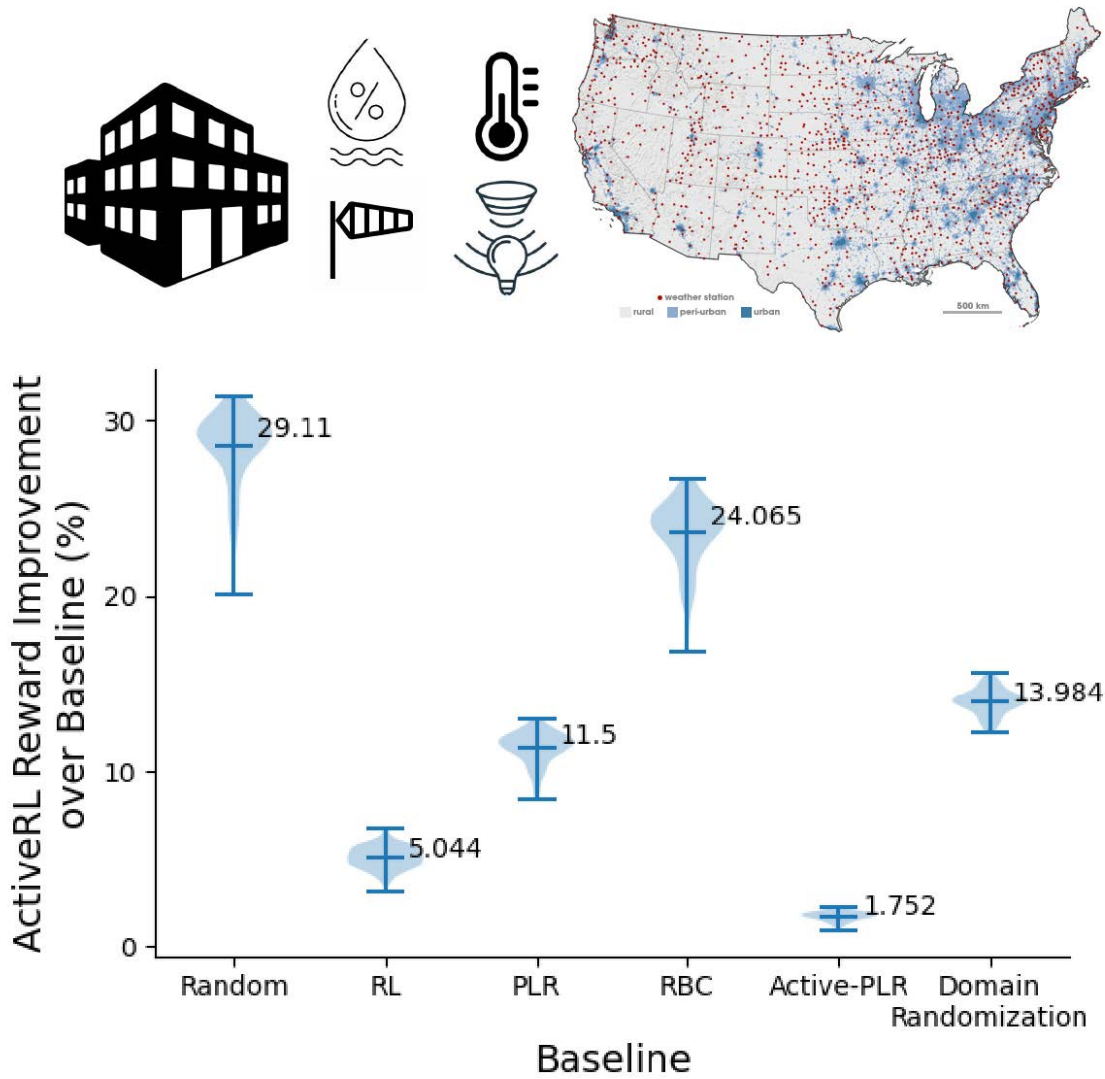


Figure 5.2: Improvement in reward achieved by using ActiveRL instead of each baseline, on 120 randomly sampled weather patterns from across the US. A higher number here indicates that ActiveRL performs well in comparison to that baseline. On the other hand, a higher number indicates the baseline performs poorly.

from across the US.

On all 120 environments, ActiveRL achieves a higher reward than every baseline, showing that our uncertainty-driven UED approach can train an RL HVAC agent that is more robust to realistic extreme weather patterns than any of our baselines. Since ActiveRL outperforms all the baselines on every environment, we visualize how much better the reward achieved by ActiveRL is relative to each baseline in each of the 120 different weather patterns in fig. 5.2. The median improvement of ActiveRL over the RBC is 24%. Over vanilla RL, it is 5%.

Interestingly, there is a very small improvement using ActiveRL over ActivePLR; this, combined with the similar performance between ActiveRL and ActivePLR from section 5.2 suggests that ActiveRL and ActivePLR have very similar behavior. One possible reason is that there may be some attractive local (or global) optimum that both algorithms fall into, resulting in them appearing to have similar performance. Another possible reason is that since PLR tries to sample environment configurations that result in high value loss, and ActiveRL tries to sample environment configurations that result in high value uncertainty, PLR and ActiveRL actually optimize for very similar objectives. Thus ActiveRL and ActivePLR end up having similar behavior, at least with respect to their responses to weather. One advantage that ActiveRL has in optimizing for uncertainty rather than value loss is that it can be used to identify novel environments to learn from rather than having to sample from old ones, or worry about the staleness of the value loss estimates of the environments in the replay buffer.

There is a significant difference between both Domain Randomization and PLR, and vanilla RL. Both UED methods seem to perform poorly compared to vanilla RL. This seems to indicate that randomly sampling environment configuration parameters results in environments that are very unrealistic, causing poor generalization performance compared to the vanilla RL algorithm or ActiveRL. Although PLR has the ability to control what environments in its replay buffer are sampled, its replay buffer is still populated through the same uniform random sampling process as used in Domain Randomization, resulting in training on unrealistic environments.

5.4 ActiveRL Generalizes from Simulation to Reality (Sim2Real)

One flaw in this work and UED algorithms in general is that a simulator is required to train the model in different environments. Thus it is important to ask whether or not the RL policies trained in simulation can be extended to the real world. In order to test this, we evaluated each RL algorithm in each of the 6 environments by

running the EnergyPlus simulator at a higher fidelity than the agents were trained on, thus simulating the "Sim2Real" jump. The agents were trained on a simulator operating at a granularity of 1 hour per timestep, and we evaluate on a granularity of 15 minutes per timestep. During evaluation, each of the RL agents' actions are simply repeated four times so that it still takes an action every hour. Essentially, all the RL agents are evaluated in a simulation that is four times more realistic than the one that they were trained on.

An illustration of the performances of each algorithm on each of the 6 handcrafted environments from section 5.2 are shown in fig. 5.3.

When the agents are transferred from the simulation to our surrogate for the real world, we see that there is a significant performance drop across all algorithms. In particular, regular RL achieves a reward that is 8.5% lower on average across the 6 handcrafted environments when evaluated on the higher fidelity simulation. We see that Domain Randomization and PLR have smaller relative drops of about 7%. However, since the 7% is relative to the performance of Domain Randomization and PLR in the original low fidelity simulation, vanilla RL still performs better in terms of absolute reward over the six environments, as can be seen in fig. 5.3.A.

Random and RBC have very small or no relative performance degradation, which is to be expected because they are not data-driven models. They still perform the worst in terms of absolute reward.

ActiveRL and ActivePLR, however, achieve both smaller relative drops in performance and higher absolute reward across all the different extreme weather scenarios. ActiveRL has only a 6.1% relative drop in reward while ActivePLR has only a 3.1% relative drop. These are promising results that indicate that these algorithms would still perform well if deployed in the real world after being trained in simulation. Furthermore, these algorithms result in agents that are more robust to the Sim2Real transfer than other methods.

We found that ActivePLR actually makes the agent more robust to the Sim2Real transfer than ActiveRL, which is a surprising result since ActivePLR did not appear to be significantly different from ActiveRL in the previous two experiments. It is possible that there is some attractive local optimum in the HVAC control task in the low fidelity simulation that both ActivePLR and ActiveRL fall into, resulting in them having similar performance in the experiments described in section 5.2 and section 5.3. Once they are evaluated in the higher fidelity simulation however, this local optimum may not exist anymore, resulting in the more robust method, ActivePLR, shining through. As for why ActivePLR seems to be more robust, perhaps the recorded value loss that PLR and ActivePLR use is a stronger signal than the value uncertainty than ActiveRL uses since the value loss is obtained by actually collecting data while the value uncertainty is estimated using only the model weights through the Monte Carlo Dropout process.

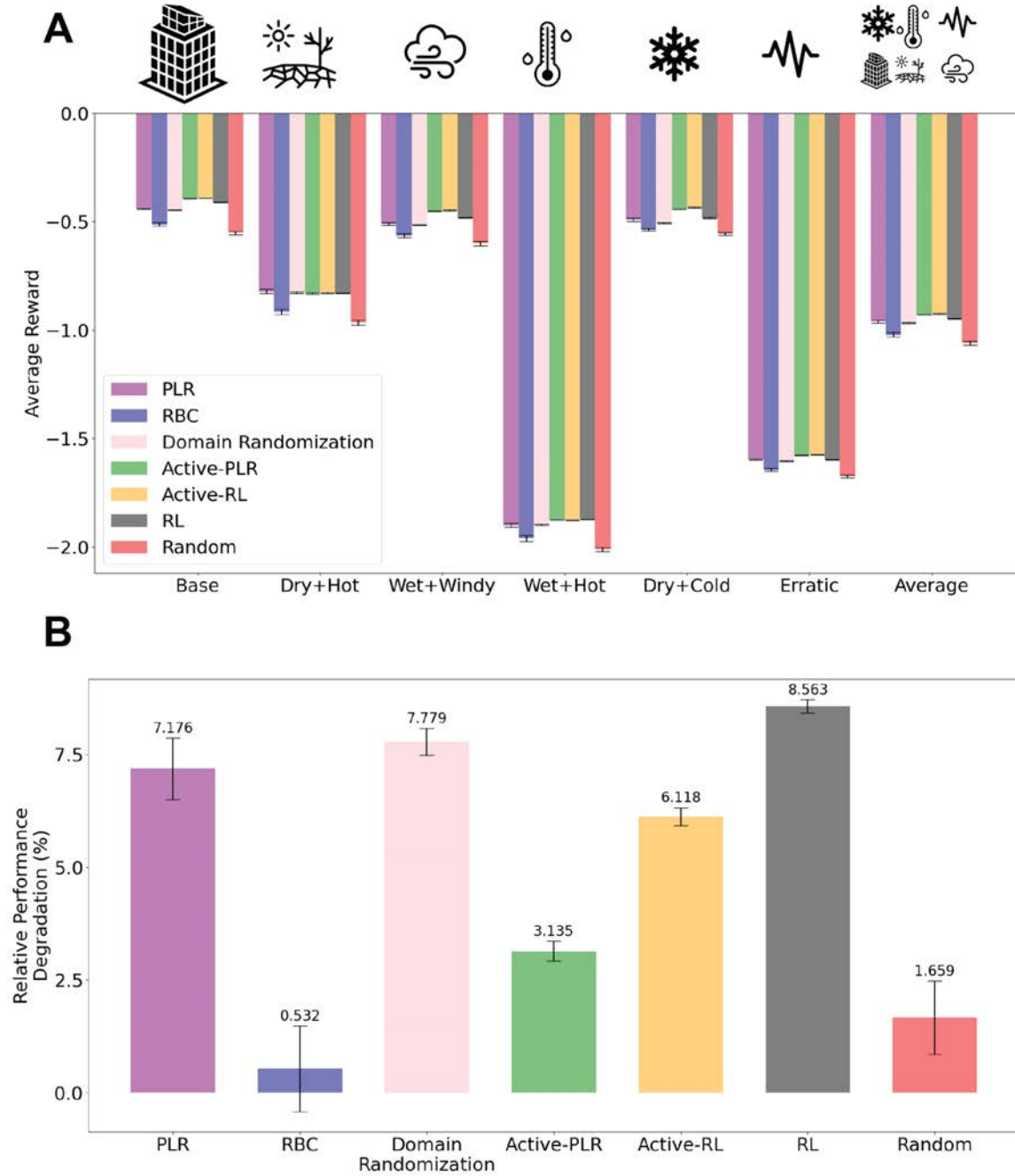


Figure 5.3: Results of Sim2Real experiments. All error bars represent the standard error of the mean over 5 trials. **A.** Each controller is trained on a building simulation with $dt = 1$ hour and evaluated on a simulation with higher fidelity $dt = 0.25$ hours for the base environment configuration ϕ_0 and five handcrafted extreme weather scenarios. The rightmost bars show the average performance of each algorithm over all 6 scenarios. **B.** The average drop in reward over all environments when evaluating in the higher fidelity simulation compared to evaluation in the lower fidelity simulation. Lower is better here.

5.5 Ablations Exploring Components of ActiveRL

In order to assess what factors contribute to the performance of ActiveRL, we ran some ablation experiments described in detail in section 4.6 that show how ActiveRL changes as certain hyperparameters change. We look at the γ hyperparameter that controls the strength of the soft constraint on ActiveRL’s environment design process, and the η parameter that controls the step-size of the optimization procedure used to generate new environments. The results of changing these two parameters can be seen in fig. 5.4, where panel A corresponds to testing different values of γ and panel B corresponds to different values of η .

Contrary to our original hypothesis that there would be some tradeoff between realism and robustness modulated by γ , we actually found that having a relatively high value of γ contributes to good performance in both the base environment and the extreme environments. There was a clear pattern that larger values of γ correlated well with better performance. This may be because ActiveRL will generate more unrealistic environment configurations ϕ with weaker regularization that are not similar enough to ϕ_0 and the extreme environments to aid performance in those settings.

We found that smaller values of η helped performance across all environments, but decreasing it below 0.001 did not change the agent’s learning trajectory at all. It is possible that having a large η results in an unstable gradient descent process which is unable to successfully find a ϕ that maximizes the agent’s uncertainty.

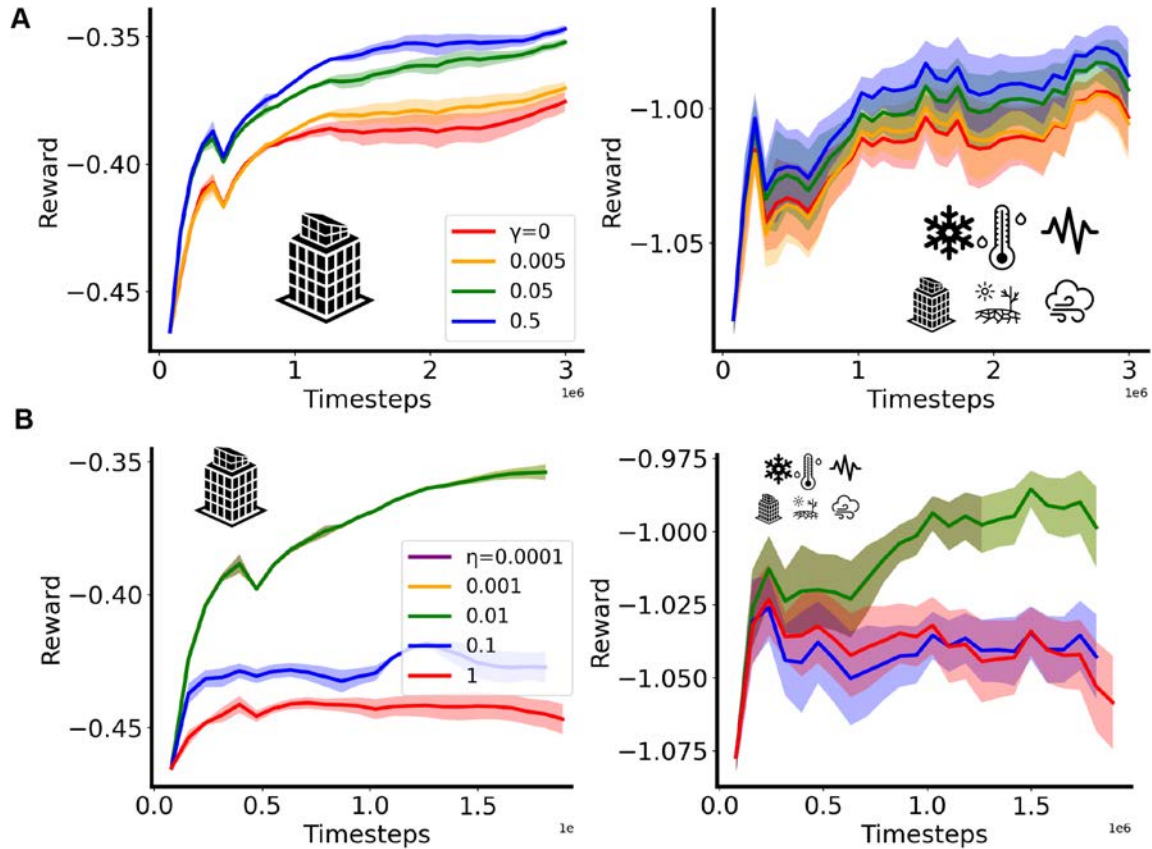


Figure 5.4: **Ablation A.** We explore how the γ regularization parameter affects the performance of ActiveRL. Higher values of γ mean that ActiveRL is forced to propose training environment configurations ϕ that are close to the base environment ϕ_0 . The left graph shows the average reward obtained throughout training on the base environment ϕ_0 . The right graph shows the reward averaged over ϕ_0 and the 5 extreme weather environments. ActiveRL seems quite sensitive to γ . **B.** Similarly, we explore how the η learning rate parameter affects ActiveRL. η is the learning rate used by ActiveRL to conduct gradient descent on ϕ . Higher values of η means a coarser-grained search for the ϕ that maximizes the agent’s uncertainty. ActiveRL seems insensitive to η once it gets small enough (< 0.1).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We explored the utility of a novel uncertainty-driven, gradient based algorithm called ActiveRL for unsupervised environment design in the context of training RL building control agents that are robust to climate change. We found that incorporating uncertainty into UED through ActiveRL led to HVAC controllers that better optimized thermal comfort and energy usage, even in extreme weather scenarios that were never in the training distribution. Our experiments showed that other UED algorithms perform poorly when generating new environment configurations for weather patterns because they may output unrealistic weather patterns that do not help the RL agent perform well in more realistic weather scenarios. These results held true for 120 different unseen, realistic weather scenarios sampled from across the US. Furthermore, we showed that ActiveRL and its variant ActivePLR would have a much smaller degradation in performance when transferring from the simulated domain to the real world compared to other techniques, making them a practical option for training robust RL HVAC agents that are ready for real deployment.

6.2 Limitations and Future Work

One limitation of ActiveRL is that it requires the environment configuration variables to be continuous in order to conduct gradient descent. This could be mitigated by applying dequantization techniques [15] that transform categorical variables into continuous variables. By intelligently adding noise to one-hot categorical variables, one can transform a categorical variable into a continuous one, through techniques like uniform dequantization or variational dequantization [29]. The inputs to the neural network RL agent then, will be continuous, so we can identify an environment

configuration ϕ that maximizes uncertainty for that agent. We can then re-quantize the continuous values of ϕ into a discrete representation we can feed into the simulator.

Another limitation of our work, and work that optimizes PCG environments in general is that they rely heavily on simulations. We conducted some simulated experiments to assess whether or not the policies learned in simulation can be transferred to the real world in section 5.4, but ideally we would test this out in a real building. There are probably many ways that real building HVAC control is different from simulated HVAC control, even if we have Sinergym simulate the building with high fidelity.

One limitation in the context of real-world deployment, is that we assume that no sensors are ever faulty or noisy in our simulation. In the real world, this would not be the case, as sensors often fail or become less accurate over time. A future project could explore whether or not an uncertainty-driven approach could identify which sensors that the agent is most reliant on and drop those in order to increase robustness to faulty sensors. Building an RL agent that is robust to sensor failures would greatly increase the viability of RL HVAC control in smart buildings.

6.3 Roadmap to Deployment

To close, we would like to give a brief description of the steps one might take to deploy an RL HVAC controller like this into the real world.

The first step is to find or build some simulator¹ that is reasonably accurate for building control. One commonly used simulator is EnergyPlus [14]. Next, one would identify how to transform this into an OpenAI Gym environment [7], as this is the API that most RL libraries use to collect data from their simulated environments. EnergyPlus has a Gym wrapper called Sinergym [33] that we use extensively in this paper.

One should then identify what aspects of the building configuration can be changed at the start of each simulation; these are the variables that ActiveRL (or other UED algorithm) would attempt to optimize over to train a more robust RL agent. Then the next step would be to actually train the RL agent in simulation using ActiveRL, most likely employing some RL library like Stable Baselines[54] or RLLib [40].

After training the RL agent using ActiveRL in simulation, it is time to attempt to deploy it in the real world. First one should fit their desired smart building with sensors for indoor and outdoor temperature, occupancy, humidity, wind, etc; everything that was included in the state space of the RL agent in the simulation.

¹Note that one could also collect offline building control data from a real building and train the RL agent using offline reinforcement learning techniques. However, this is only feasible if you have access to a real building's data.

Then they should ensure that all the sensors and the HVAC systems can communicate with some central server using, for example, the BACnet protocol [50]. This central server can then host the trained RL agent, which will take in the sensor data as input and send HVAC setpoints out through BACnet to the HVAC systems in the smart building.

While the RL agent is deployed in the building, one should store all of the sensor data, energy usage, and HVAC setpoint actions that the RL agent interacts with. After enough data has been collected, one can update the RL agent using PPO on that data, so that the RL agent can learn to take better actions in the real world, and adapt to trends like climate change making regions warmer or cooler.

Bibliography

- [1] US Energy Information Administration. *Annual Energy Outlook 2020*. Government Printing Office, 2011.
- [2] Abdul Afram and Farrokh Janabi-Sharifi. “Theory and applications of HVAC control systems – A review of model predictive control (MPC)”. In: *Building and Environment* 72 (2014), pp. 343–355.
- [3] Gaon An et al. “Uncertainty-based offline reinforcement learning with diversified q-ensemble”. In: *Advances in neural information processing systems* 34 (2021), pp. 7436–7447.
- [4] ASHRAE ANSI and M Ashrae. “Standard 55—thermal environmental conditions for human occupancy”. In: *Amer. Soc. Heat, Refrigerat. Air Condition. Eng* 1451992 (2017).
- [5] Donald Azuatalam et al. “Reinforcement learning for whole-building HVAC control and demand response”. In: *Energy and AI* 2 (2020), p. 100020.
- [6] Djallel Bouneffouf. “Exponentiated gradient exploration for active learning”. In: *Computers* 5.1 (2016), p. 1.
- [7] Greg Brockman et al. “Openai gym. arXiv”. In: *arXiv preprint arXiv:1606.01540* 10 (2016).
- [8] Bingqing Chen, Zicheng Cai, and Mario Bergés. “Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy”. In: *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 2019, pp. 316–325.
- [9] Richard Y Chen et al. “Ucb exploration via q-ensembles”. In: *arXiv preprint arXiv:1706.01502* (2017).
- [10] Xiaoyu Chen et al. “Understanding domain randomization for sim-to-real transfer”. In: *arXiv preprint arXiv:2110.03239* (2021).
- [11] Maxime Chevalier-Boisvert et al. “Babyai: A platform to study the sample efficiency of grounded language learning”. In: *arXiv preprint arXiv:1810.08272* (2018).

- [12] Karl Cobbe et al. “Leveraging procedural generation to benchmark reinforcement learning”. In: *International conference on machine learning*. PMLR. 2020, pp. 2048–2056.
- [13] David Cohn, Les Atlas, and Richard Ladner. “Improving generalization with active learning”. In: *Machine learning* 15 (1994), pp. 201–221.
- [14] Drury B Crawley et al. “EnergyPlus: creating a new-generation building energy simulation program”. In: *Energy and buildings* 33.4 (2001), pp. 319–331.
- [15] Hari Prasanna Das and Costas J Spanos. “Improved dequantization and normalization methods for tabular data pre-processing in smart buildings”. In: *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2022, pp. 168–177.
- [16] Xiangtian Deng, Yi Zhang, and He Qi. “Towards optimal HVAC control in non-stationary building environments combining active change detection and deep reinforcement learning”. In: *Building and Environment* (2022).
- [17] Michael Dennis et al. “Emergent complexity and zero-shot transfer via unsupervised environment design”. In: *Advances in neural information processing systems* 33 (2020), pp. 13049–13061.
- [18] Davy Didden et al. “Sample efficient reinforcement learning with domain randomization for automated demand response in low-voltage grids”. In: *IEEE Journal of Emerging and Selected Topics in Industrial Electronics* 3.4 (2021), pp. 891–900.
- [19] Joseph L Doob. “The Brownian movement and stochastic equations”. In: *Annals of Mathematics* (1942), pp. 351–369.
- [20] Poul O Fanger. “Calculation of thermal comfort: introduction of a basic comfort equation”. In: *ASHRAE Trans, Part II* 73 (1967), pp. III4–1.
- [21] Bruno Faria et al. “The Joint Role of Batch Size and Query Strategy in Active Learning-Based Prediction-A Case Study in the Heart Attack Domain”. In: *Progress in Artificial Intelligence: 21st EPIA Conference on Artificial Intelligence, EPIA 2022, Lisbon, Portugal, August 31–September 2, 2022, Proceedings*. Springer. 2022, pp. 464–475.
- [22] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.
- [23] Jose Gallego-Posada and Juan Ramirez. *Cooper: a toolkit for Lagrangian-based constrained optimization*. <https://github.com/cooper-org/cooper>. 2022.

- [24] Kai Gong et al. “Comprehensive review of modeling, structure, and integration techniques of smart buildings in the cyber-physical-social system”. In: *Frontiers in Energy* (2022), pp. 1–21.
- [25] Lindsay T Graham, Thomas Parkinson, and Stefano Schiavon. “Lessons learned from 20 years of CBE’s occupant surveys”. In: *Buildings and Cities* 2.1 (2021).
- [26] Vijaykumar Gullapalli. “A stochastic reinforcement learning algorithm for learning real-valued functions”. In: *Neural networks* 3.6 (1990), pp. 671–692.
- [27] HADO van Hasselt, ARTHUR Guez, and DAVID Silver. “Deep Reinforcement Learning with Double Q-learning. arXiv e-prints”. In: *arXiv preprint arXiv:1509.06461* (2015).
- [28] Hasan Hayat et al. “The state-of-the-art of sensors and environmental monitoring technologies in buildings”. In: *Sensors* 19.17 (2019), p. 3648.
- [29] Jonathan Ho et al. “Flow++: Improving flow-based generative models with variational dequantization and architecture design”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2722–2730.
- [30] Doseok Jang et al. “Offline-online reinforcement learning for energy pricing in office demand response: lowering energy and data costs”. In: *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2021, pp. 131–139.
- [31] Safieh Javadinejad et al. “Climate change management strategies to handle and cope with extreme weather and climate events”. In: 2020.
- [32] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. “Prioritized level replay”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4940–4950.
- [33] Javier Jiménez-Raboso et al. “Sinergym: A Building Simulation and Control Framework for Training Reinforcement Learning Agents”. In: *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 319–323. ISBN: 9781450391146. DOI: 10.1145/3486611.3488729. URL: <https://doi.org/10.1145/3486611.3488729>.
- [34] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [35] Galina M Korpelevich. “The extragradient method for finding saddle points and other problems”. In: *Matecon* 12 (1976), pp. 747–756.

- [36] Xiao Kou et al. “Model-Based and Data-Driven HVAC Control Strategies for Residential Demand Response”. In: *IEEE Open Access Journal of Power and Energy* 8 (2021), pp. 186–197.
- [37] Kuldeep R. Kurte et al. “Evaluating the Adaptability of Reinforcement Learning Based HVAC Control for Residential Houses”. In: *Sustainability* (2020).
- [38] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. In: *Advances in neural information processing systems* 30 (2017).
- [39] Kimin Lee et al. “Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6131–6141.
- [40] Eric Liang et al. “Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page”. In: *arXiv preprint arXiv:1712.09381* (2017).
- [41] Xiangfei Liu et al. “A multi-step predictive deep reinforcement learning algorithm for HVAC control systems in smart buildings”. In: *Energy* (2022).
- [42] Xiaoqi Liu. “Exploration of Intelligent HVAC Operation Strategies for Office Buildings”. PhD thesis. Purdue University Graduate School, 2020.
- [43] Donald P Lynch. *EVOP design of experiments*. Tech. rep. Citeseer, 2003.
- [44] Lázaro Emilio Makili, Jesús A Vega Sánchez, and Sebastián Dormido-Canto. “Active learning using conformal predictors: application to image classification”. In: *Fusion Science and Technology* 62.2 (2012), pp. 347–355.
- [45] Valérie Masson-Delmotte et al. “Climate change 2021: the physical science basis”. In: *Contribution of working group I to the sixth assessment report of the intergovernmental panel on climate change 2* (2021).
- [46] Edward Henry Mathews et al. “HVAC control strategies to enhance comfort and minimise energy usage”. In: *Energy and Buildings* 33 (2001), pp. 853–863.
- [47] Alberto Maria Metelli, Amarildo Likmeta, and Marcello Restelli. “Propagating uncertainty in reinforcement learning via wasserstein barycenters”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [48] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.
- [49] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).
- [50] H Michael Newman. *Bacnet: the global standard for building automation and control networks*. Momentum Press, 2013.

- [51] Brendan O’Donoghue et al. “The uncertainty bellman equation and exploration”. In: *International Conference on Machine Learning*. 2018, pp. 3836–3845.
- [52] Ian Osband et al. “Deep exploration via bootstrapped DQN”. In: *Advances in neural information processing systems* 29 (2016).
- [53] Jack Parker-Holder et al. “Evolving curricula with regret-based environment design”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 17473–17498.
- [54] Antonin Raffin et al. “Stable-baselines3: Reliable reinforcement learning implementations”. In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 12348–12355.
- [55] Naren Srivaths Raman et al. “Reinforcement learning for control of building HVAC systems”. In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 2326–2332.
- [56] Sebastian Risi and Julian Togelius. “Increasing generality in machine learning through procedural content generation”. In: *Nature Machine Intelligence* 2.8 (2020), pp. 428–436.
- [57] Syed Ali Asad Rizvi and Amanda J Pertzborn. “Experimental Results of a Disturbance Compensating Q-learning Controller for HVAC Systems”. In: *2022 American Control Conference (ACC)*. IEEE. 2022, pp. 3353–3353.
- [58] Mikayel Samvelyan et al. “The starcraft multi-agent challenge”. In: *arXiv preprint arXiv:1902.04043* (2019).
- [59] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).
- [60] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [61] David Silver et al. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.
- [62] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [63] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [64] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

- [65] Quan Vuong et al. “How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?” In: (2019).
- [66] Peng Wang et al. “Bayesian neural networks uncertainty quantification with cubature rules”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–7.
- [67] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8 (1992), pp. 279–292.
- [68] AP Wemhoff and MV Frank. “Predictions of energy savings in HVAC systems by lumped models”. In: *Energy and Buildings* 42.10 (2010), pp. 1807–1814.
- [69] Stephen Wilcox and William Marion. *Users manual for tmy3 data sets (revised)*. Tech. rep. National Renewable Energy Lab.(NREL), Golden, CO (United States), 2008.
- [70] RJ Williams. “Toward a theory of reinforcement-learning connectionist systems”. In: *Technical Report* (1988).
- [71] Yue Wu et al. “Uncertainty weighted actor-critic for offline reinforcement learning”. In: *arXiv preprint arXiv:2105.08140* (2021).
- [72] Tianpei Yang et al. “Exploration in deep reinforcement learning: a comprehensive survey”. In: *arXiv preprint arXiv:2109.06668* (2021).
- [73] Liang Yu et al. “Multi-Agent Deep Reinforcement Learning for HVAC Control in Commercial Buildings”. In: *IEEE Transactions on Smart Grid* 12 (2020), pp. 407–419.
- [74] Xiangyu Zhang et al. “Grid-interactive multi-zone building control using reinforcement learning with global-local policy search”. In: *2021 American Control Conference (ACC)*. IEEE. 2021, pp. 4155–4162.
- [75] Zhiang Zhang. “A Reinforcement Learning Approach for Whole Building Energy Model Assisted HVAC Supervisory Control”. In: 2019.