Can we stick a PINN in it?

Physics-Informed
Deep Reinforcement Learning
for
Partial Differential Equation Control
in the Context of
Smart-Building Management

Jacob Turner

Master of Science in Machine Learning and Autonomous Systems
The University of Bath
2021

Physics-Informed Deep Reinforcement Learning for Partial Differential Equation Control in the context of Smart-Building Management

Submitted by: Jacob Turner

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Conventional building management techniques are growing obsolete due to recent trends in renewable grid design (Wang and Hong (2020)). This has prompted significant research into the application of Reinforcement Learning (RL) to build systems capable of intelligent building control. Many of these applications show promising results. However, learning an optimal control strategy is often computationally intensive and time-consuming, typically requiring a large number of interactions. This research aims to address this challenge by proposing a novel method which encodes the agent with prior understanding of its environment to improve sample efficiency and performance. This is accomplished by integrating a physics-informed neural network (PINN) into the agent architecture. The PINN contains the environment-governing partial differential equation (PDE) encoded in its loss function. This acts as a regularization technique which informs the agent of physical laws such as conservation of mass and momentum. A physics-informed and standard instance were trained using the Deep Q-Network, REINFORCE, and Advantage Actor-Critic RL algorithms to control a diffusion and convection environment. This resulted in the physics-informed instance requiring 64% fewer interactions on the simple diffusion environment and 92.5% fewer interactions to control the more complex convection environment. In addition, performance by the physics-informed instance on the diffusion and convection environments increased by 34.9% and 27.7% respectively. This approach could be used to reduce training requirements and improve physics-control in many engineering disciplines.

Acknowledgements

I would like to begin by thanking my supervisor Dr. Vinay Namboodiri for his advice and guidance in completing this project. I would also like to thank my friends and family for their support during the program and dissertation period as well as their constructive feedback in the written portion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advances in physics and mathematics have allowed for the accurate description of many complex phenomena. For example, by using principles of calculus known as a differential equation, one can model the change in a quantity with respect to time. Learning to better control events described by differential equations may be invaluable within countless industries. One such industry, building management, has become increasingly complex in recent years. This trend is caused by the need to handle new challenges such as intermittent renewable energy sources, energy storage, and vehicle charging, in addition to traditional services (Wang and Hong (2020)). Such recent challenges have shifted significant research into Reinforcement Learning (RL). This is a field of machine learning concerned with developing a control strategy through exposure to trial and error experiences. It has been successfully applied to interact with and control complex physical systems. However,an RL agent typically has to develop the understanding of its environment from first principles. This contributes to the large number of interactions often necessary for learning which leads to extensive training time (Yu (2018)). This challenge includes events with heavily researched and well understood physics, many of which can be described concisely using a differential equation. The lack of prior understanding inhibits the agent performance and severely impacts the sample efficiency (i.e. the number of interactions necessary to achieve satisfactory performance). This work introduces a novel method which integrates prior mathematical physics understanding into RL algorithms. This approach expedites the search for an optimal policy and reduces the number of interactions necessary for learning to control a physics-governed environment. Specifically, this work aims to incorporate a Physics-Informed Neural Network (PINN) layer into the deep RL agent neural network architecture. This will provide the agent with a prior understanding of the physics that is governing its environment and improve its sample efficiency and overall performance.

1.1 Background

Optimal control of a partial differential equation (PDE) is a complex, high-dimensional problem with countless applications ranging from fluid flow control, to aerodynamic shape optimization, and automated building management. The latter is the primary context in which this work will take place. Specifically, a contrived example of a simplified building management system will be considered as a foundation for the exploration of new methods for physics control using deep RL techniques.

Consider a room where the temperature profile is governed by a simple PDE describing the transfer of heat. A Heating Ventilation and Air Conditioning (HVAC) unit controls the temperature of the boundary at each end of the room, which can be set to cool or heat the room. At the center of the room is an occupant with typical external body temperature, who experiences the heat transfer from the temperature profile of the room. A simple diagram depicting this environment is shown in Figure 1.1. Should the occupant experiences a positive heat transfer from the environment, this will manifest by affecting their comfort level and causing them to feel hot. Conversely, a negative heat transfer would result in them losing heat to the environment and affect their comfort by causing them to feel cold. A neutral heat transfer, close to zero, would theoretically maximize the comfort level of the occupant by matching their external body temperature. Therefore, the primary goal of the RL agent is to successfully control the HVAC unit so to minimize heat transfer between the occupant and its environment. This environment framework will be the primary mechanism with which PDE control using RL techniques will be explored.

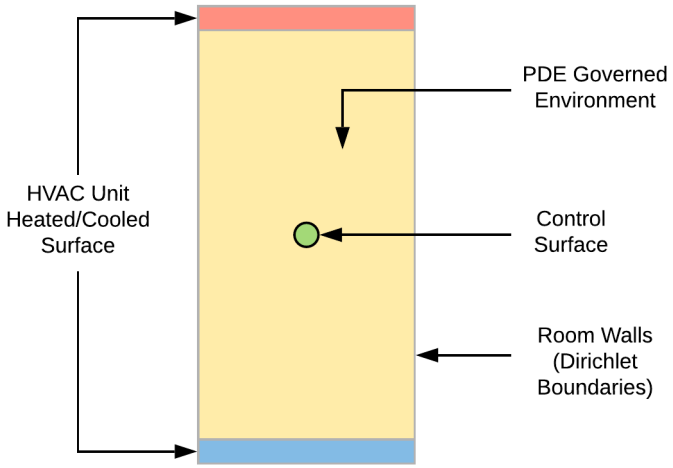

Figure 1.1: Building management environment framework for exploration of PDE control using RL techniques.

While the agent interacts with the environment by controlling the HVAC boundaries, it does not have direct control over the temperature profile in the environment. This is

governed by a system of PDEs, including the diffusion, convection, and Navier-Stokes equations. The diffusion equation is considered initially for simplicity and to provide a proof-of-concept as a foundation for later extension to a convection-based environment. A PDE solver is necessary to compute the temperature profile in the environment as it is affected by the changing boundaries controlled by the agent. A python library, FEniCS, a Finite Element Method (FEM)-based PDE solver is utilized as the primary engine of the simulated environment.

1.2 Scope

The primary objective of this work is to investigate an application for prior mathematical understanding in a controllable physics-governed environment. Specifically, it is hypothesized that a PINN can be used within the architecture of a deep RL algorithm to improve performance and reduce training requirements. In addition, the application of deep RL agents to control complex high-dimensional environments governed by diffusion, convection, and the Navier-Stokes equations is investigated as it relates to HVAC and autonomous smart-building control. The application of deep RL to this area of building management has been explored previously. For example, Farahmand et al. (2016), utilized the Regularized Fitted Q-Iteration (RFQI) algorithm to control a two-dimensional convection-diffusion PDE in the context of an HVAC unit. Similarly, Dermardiros, Bucking and Athienitis (n.d.) used the Proximal Policy Optimization (PPO) RL algorithm for HVAC control in an OpenAI gym building emulator. Alternatively, this work aims to explore the utilization of prior mathematical understanding in calculus and physics to improve upon standard deep RL methods in a similar context.

A new package, Gym-HVAC, is developed to model a simplified building management scenario. The package contains two instances of an environment representing a room, HVAC unit, and a control surface. The first instance is governed by a simple heat diffusion equation while the second is governed by a system of equations including the convection-diffusion and Navier-Stokes equations. The general goal of either instance is to efficiently control the HVAC unit to maximize the comfort of the occupant. Gym-HVAC is implemented to utilize the OpenAI gym framework (Brockman et al. (2016)) for efficient and easy interaction by externally designed agents. This package incorporates a simple modular design, with many configurable environmental parameters, and a 2D heat-map visualization of the environment state.

The development of Gym-HVAC allowed for the investigation of a novel technique to incorporate prior mathematical understanding into RL models for improved learning and control of physics-based environments. This prior mathematical understanding can be encoded using a physics-informed neural network formulated to use the governing

PDE in its loss function. Following the implementation of the PINN layer into a custom Deep Q-Network (DQN), several experiments were conducted on various scenarios in Gym-HVAC to investigate the benefit of the modified algorithm. In addition, a wrapper was implemented to allow the PINN to interface with more complex RL algorithms such as REINFORCE and advantage actor-critic (A2C), while training on the same environment.

Ultimately, the objectives of this project are as follows:

- Design and implement a physics-informed deep reinforcement learning agent by incorporating a PINN into the agent neural network architecture.
- Train the physics-informed RL agent to successfully control the PDE-governed environment, using Gym-HVAC.
- Investigate performance of the physics-informed RL agent when compared to the standard RL agent on an identical environment.

Chapter 2

Literature and Technology Review

2.1 Introduction

The mathematical understanding of physical systems was crucial in the development of countless modern conveniences. The continued improvement of this understanding can allow for better control of the complex dynamical systems that affect daily life. One such system that is often overlooked, yet ubiquitous in our everyday lives, involves the effect of the temperature profile in a room on its occupants and their comfort level. This concept initially sounds simple - just turn up the thermostat, right? - When in fact, temperature in a room, and occupant comfort, is influenced by many environmental factors and a system of Partial Differential Equations (PDE). Namely, the Navier-Stokes equations, which models the movement of fluid flow, and the Convection-Diffusion equation, which models the transfer of heat. Control of systems governed by PDE's is challenging due to its high-dimensional nature. Developing new methods for successful control of these complex physics-governed systems can allow for new techniques in environmental design (Saenz-Aguirre et al. (2019)), aircraft control (Goecks et al. (2018)), robotics (Liu et al. (2021)), and building design (Lin et al. (2020)). The latter will be the primary topic of exploration and experimentation for the remainder of this work.

HVAC control consists of interacting with a complex environment to achieve the desired (comfortable) target. The environment, in this context, is governed by a complex system of PDE's, and hence has an infinite dimensional state space consisting of the temperature profile and airflow velocity throughout a room. In addition, the action space of the HVAC unit can be continuous, where the heater and fan can be set to any feasible power level. Alternatively, the HVAC unit can also utilize discrete control, where the temperature can only be set to finite predetermined values. Conventional HVAC systems utilize Proportional Integral Derivative (PID) controllers. However, this method is known to result in inefficient energy use, occupant discomfort, and semi-regular maintenance costs

due to system tuning (O'Neill, Li and Williams (2016)).

Another approach to control problems, known as Reinforcement Learning (RL), involves an agent capable of improving at a task by learning from its interactions with an environment. This avenue allows for automated improvement of a task, with less human input in the overall system design. However, due to the curse of dimensionality, it can be extremely difficult for an agent to learn in an environment with an infinitely large state/action space. To address this issue, the agent must be able to generalize across similar states of the environment so that an accompanying optimal action can be correctly mapped. This is the underlying role of Artificial Neural Networks (ANN) in deep RL, where the value or quality of a future state-action pair is approximated to better inform the agent what the best action may be for any given situation (Sutton and Barto (2018)).

Physics-informed Machine Learning (ML), a recently popular topic of study in computational physics, seeks to apply standard ML techniques to more efficiently and accurately model physics problems. Recent advances in the field have shown that a Physics-Informed Neural Network (PINN) can be used to accurately predict the solution of PDE's in fluid flow, quantum mechanics, and chemical diffusion problems (Raissi, Perdikaris and Karniadakis (2019)). The underlying hypothesis of this work proposes that the merging of these two areas of ML research can allow for a more efficient, higher-performing reinforcement learning algorithm when interacting with a physics-based environment.

This project will employ deep RL in conjunction with physics-informed deep learning to control an HVAC unit while optimizing occupant comfort, energy efficiency, and cost. A simulation is created using the OpenAI Gym framework and the FEniCS library for solving the environments governing PDE's in an isolated manner. A PINN layer, with the corresponding PDE hard-coded, will then be implemented as the first layer of the RL agent neural network to estimate the state of the environment governed by the PDE. Theoretically, this will encode the necessary physical laws governing the environment into the agents neural network architecture which is evaluating for the optimal action, given the current imperfect state observation. This should provide the RL agent with a prior understanding of the environment. This understanding is provided by relatively recent research in physics and mathematics. It is hypothesized that the physics-informed implementation will improve the sample efficiency and performance of the agent. This technique will be evaluated on environments governed by varying complexities of PDE's. In addition, a baseline standard deep RL agent will be trained on the environment without the physics-encoded state prediction provided by a PINN layer.

2.2 Reinforcement Learning

This section will describe the general agent-environment interaction in RL. In addition, an overview of temporal-difference learning, deep reinforcement learning, and the relevant algorithm foundations will be provided.

2.2.1 Agent and the Environment

Reinforcement learning problems involve control of an environment through interactions determined by an agent. Where, each action chosen by the agent influences the next state of the environment. The agent is then rewarded for successful control of the environment through a signal, which ultimately allows for the agent to learn. The reward is crucial in shaping the ultimate behavior of the agent. This framework requires the problem to be formulated as a Markov Decision Process (MDP), where the future state of the environment depends on the immediately previous state. A diagram of this agent-environment interaction can be seen in Figure 2.1.

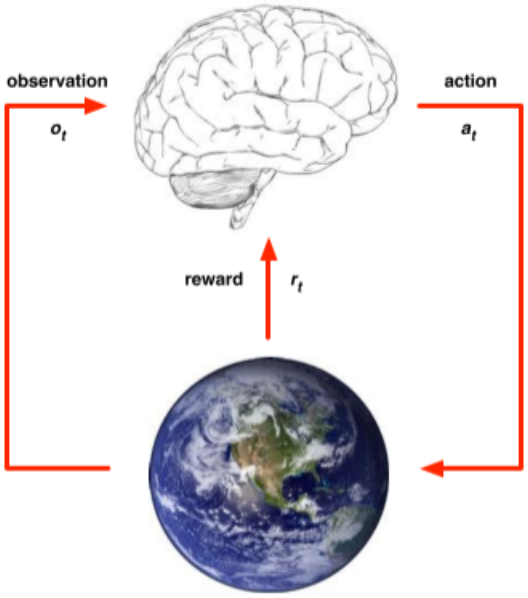

Figure 2.1: Agent environment interaction (Silver (2019))

Figure 2.1 depicts this fundamental interaction, where, at each discrete time-step the agent chooses some action a_t. This decision directly affects the next observed state o_t received by the agent, along with some reward signal r_t. Ultimately, the agent develops a policy, commonly denoted as π, which guides its decision-making to optimally interact with its environment. This policy can be instantiated in a tabular form, often referred to as a Q-table, in which the value for each observed state-action pair is stored. Alternatively, the

value can be approximated via a policy which is developed by training a neural network to estimate the value of these state-action pairs, this is known as a Deep Q-Network (DQN). In general, the goal of the agent is to maximize the total reward it is able to collect. A variety of algorithms have been developed to allow the agent to choose an optimal action in response to a new state. There are two main paradigms in deep RL: value-based and policy-based learning, each of which will be discussed in their own section.

2.2.2 Value-based Reinforcement Learning

Value-based RL utilizes a value approximation function to allow the agent to learn the expected value of an action given their associated future state. This is defined by (Sutton and Barto (2018)):

$$V^{\pi}(s) = E_{\pi}[R_t|s_t = s] \tag{2.1}$$

Here, equation 2.1 represents the expected value E_{π} of the state s at time-step t following the policy π. The expected value for taking action a at state s, known as the state-action value function, is therefore given by:

$$Q^{\pi}(s, a) = E_{\pi}[R_t|s_t = s, a_t = a] \tag{2.2}$$

Merging equation 2.1 with 2.2 demonstrates the relationship between these two concepts:

$$V^{\pi}(s) = E_{\pi}[Q^{\pi}(s, a)] \tag{2.3}$$

Equation 2.3 implies that the value of a state following π is simply the expectation, or weighted average, of the probability of each state with the state-action value function $Q^{\pi}(s, a)$. One of the most central value-based methods is Q-Learning, which is a temporal-difference (TD) learning algorithm, that uses a Q-table to store this expected value of each state-action pair. The values of this Q-table are updated iteratively by applying the Q-function until Bellman optimality (Bellman and Dreyfus (1962)) is satisfied:

$$Q^{\pi}(s, a) = R(s, a) + \gamma max_{a'} Q^{\pi}(s', a') \tag{2.4}$$

Where, γ is a future discount rate which informs the agent to value immediate rewards over temporally distant ones. The Bellman equation (2.4) indicates that the Q-value of a state-action pair is the reward of the current state-action, combined with the next action that leads to the highest reward. This allows the Q-table to be updated throughout

training by changing its policy while exploring its action space.

2.2.3 Policy-based Reinforcement Learning

Contrary to value-based RL, a policy-based approach to RL involves directly learning a parameterized policy $\pi(a|s,\theta)$ instead of requiring a value function to estimate the optimal policy (Sutton and Barto (2018)). In addition, this method provides three primary advantages (Garnier et al. (2021)):

- Can handle high dimensional and continuous action spaces.
- Able to learn stochastic policies.
- Reduces noise from parameter changes on action probabilities.

The agent performance, and thus the objective function to be optimized is defined as (Sutton and Barto (2018)):

$$J(\theta) = E_{\pi\theta}[R_t] = v_{\pi\theta}(s) \tag{2.5}$$

where the parameters θ are subject to maximization, typically using a stochastic gradient descent method, such that:

$$\theta = arg\ max_\theta E_{\pi\theta}[R_t] \tag{2.6}$$

This requires the gradient with respect to the policy parameters. This can be formulated according to the policy gradient theorem (Sutton and Barto (2018)) as an expected value using the log-probability trick as follows (Williams (1992)):

$$\nabla_\theta J(\theta) = E_{\pi\theta}[\nabla_\theta log \pi_\theta(a|s,\theta) \cdot Q^{\pi\theta}(s,a)] \tag{2.7}$$

Equation 2.7 is then applied to update the policy parameters with a learning rate α:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.8}$$

The expected value in equation 2.7 allows the gradient to be averaged over a whole set of exemplars in a single episode. This allows the negative effect of harmful actions to be countered with the most beneficial actions in the distribution (Garnier et al. (2021)). Furthermore, this difficulty is addressed with the introduction of actor-critic methods, which effectively merges policy gradient optimization with a value function.

2.3 Physics-Informed Machine Learning

This section will provide a brief overview of the relevant Differential Equations and the intersection of machine learning techniques used in this field. In addition, a detailed formulation of an example application of a physics-informed neural network will be outlined along with an exploration of the current research.

2.3.1 Differential Equations

A differential equation describes the relationship between functions and how these change with respect to a defined variable such as time or space (Zill and El-Iraki (2018)). These types of relationships appear often in physics, engineering, finance, or even biology. Within these domains, the functions typically represent values such as temperature, pressure, concentration, price, or population. In this case, the derivatives represent the instantaneous rate of change with respect to another variable (again, often time or space). A differential equation where the derivatives are taken with respect to one variable is known as an Ordinary Differential Equation (ODE). Alternatively, a differential equation containing functions changing with respect to multiple variables is referred to as a Partial Differential Equations (PDE). Many simple ODEs and PDEs can be solved analytically, meaning an exact solution can be obtained. While many are analytically intractable, meaning they can only able be approximated by a numerical method. Runge-Kutta-Fehlerg (RKF), Finite Difference, or Finite Element Analysis (FEA) are some conventional numerical methods for approximating these equations. These methods typically involve discretizing the equation so that it can be solved with a defined time step. In addition, FEA requires the weak or variational formulation of the differential equation. This effectively involves converting the functions to vector space so that the solution can be represented by a distribution that fits some provided boundary conditions (Larson and Bengzon (2015)).

There are three primary PDEs that are relevant to accurately describing temperature in a room. This includes diffusion, convection, and the Navier-Stokes equations. Diffusion, in this context, describes the natural transfer of heat from high to low temperature zones; essentially, it is the tendency for a material or medium to reach a steady-state temperature (Demirel (2002)). Convection describes the movement of heat stored in a fluid due to differences in density, which is typically caused by a temperature gradient (Campbell (2015)). Lastly, the Navier-Stokes equations more holistically describe the movement of a fluid by merging convection and diffusion into one relationship with the pressure gradient and flow velocity in a domain. The PDE describing time-dependent heat diffusion through a material with a thermal diffusivity κ can be seen below (Kirkwood (2018)):

$$\frac{\partial T}{\partial t} - \kappa \nabla^2 T = f(x, t)$$

$$T(0,t) = T(1,t) = 0$$

$$T(x,0) = T_0$$

The necessary boundary and initial conditions are also shown below the PDE. To provide additional context, one can imagine this equation describing the time-dependent diffusion of heat in one spatial dimension through a metal rod. Where the domain of length 1 reflects the length of the rod and the initial condition T_0 describes the temperature throughout the rod at time zero. A heat source in the domain can be implemented on the right hand side with an arbitrary function $f(x,t)$, setting this to zero would describe no source. This equation can be extended further to the convection-diffusion equation below (Becker and Vexler (2007)):

$$\frac{\partial T}{\partial t} = \frac{1}{Pe}\nabla^2 T - \nabla \cdot (uT) + f(x,t)$$

Where Pe refers to the Peclet number, which is a function of the length of the domain and the thermal diffusivity κ of the transport material. A function $f(x,t)$ can also describe a source within the domain. Lastly, the velocity u is a vector field throughout the domain which can be calculated by solving the Navier-Stokes Equations shown below (Chorin (1968)):

$$\rho\frac{\partial u}{\partial t} - u\Delta u + \rho(u \cdot \nabla)u + \nabla p = f(x,t)$$

$$\Delta . u = 0$$

This is the Navier-Stokes momentum equation for an incompressible Newtonian fluid. To respect conservation of mass, the velocity field must be divergence free, denoted by $\Delta \cdot u = 0$ (Chorin (1968)). A function $f(x,t)$ can be defined to describe a source term for momentum or force within the domain, which would describe increased movement in the fluid.

2.3.2 Physics-Informed Deep Learning

The field of deep learning is relatively new compared to the centuries of research in physics or mathematics. In addition, our understanding of physics has progressed in a non-linear fashion, with much of the progress being made in the last century or two. Initially, the idea of utilizing deep learning techniques to approximate high-dimensional solutions of PDEs sounds overly simplistic and unlikely to yield accurate results. However, a neural network can be formulated to utilize this prior mathematical understanding of the physical system it is tasked with approximating. Hard coding the physical laws that govern a

complex system can act as a regularization technique that limits the neural network to finding a solution within the laws of physics. This constrains the search space to a more tractable size and automatically rejects unrealistic solutions that break a physical law such as conservation of mass (Raissi, Perdikaris and Karniadakis (2019)).

Previously, Bayesian machine learning techniques have been implemented to solve similar problems in physics modelling. Utilizing Bayesian approaches for this application makes sense because it allows for the implementation of prior knowledge into the machine learning models. One approach used Bayesian numerical analysis for numerical homogenization, or the discretized approximation of solutions for PDEs with rough coefficients (Owhadi (2015)). Another more recent approach to a similar problem applies gaussian process regression to model physical conservation laws that are expressed within parametric linear equations, which includes ordinary and partial differential equations (Raissi and Karniadakis (2017)). These applications of machine learning towards solving partial and ordinary differential equations were loosely inspired by the first published application of ANNs used for solving ODEs by Lagaris et al in 1997. This publication involved training a neural network on the boundary and initial conditions to ultimately satisfy the differential equation (Lagaris, Likas and Fotiadis (1998)). The results of this application were then compared to Finite Element Method (FEM) approximations and found that solutions by the neural network are not limited to a discrete solution, while this is a necessary step in FEM. In addition, it was found that the neural network solutions had a lower deviation error (Lagaris, Likas and Fotiadis (1998)). This technique was extended further by Raissi, Perdikaris and Karniadakis (2019), where auto-differentiation is applied using the tensorflow Python library (Abadi et al. (2015)) to solve several PDEs in two dimensions. The formulation of a neural network to approximate a PDE based on the last two applications (Lagaris et al & Raissi et al) will be discussed in more detail below.

This "data-driven" approach to solving differential equations involves utilizing a neural network that encodes the physical laws into its loss function. This takes advantage of prior understanding in physics when estimating a PDE solution. A formulation, based on Raissi, Perdikaris and Karniadakis (2019), of a neural network solving the diffusion equation will be provided. Recall the diffusion equation with the following boundary and initial conditions below:

$$\frac{\partial T}{\partial t} - k\nabla^2 T = f(x,t)$$

$$T(0,t) = T(1,t) = 0$$

$$T(x,0) = T_0$$

These can be formulated to equal zero so that they can be minimized in a mean square error (MSE) loss function, which the neural network will be able to perform back-propagation on. This can be seen below, where each equation is given a new variable for later reference:

$$D = \frac{\partial T}{\partial t} - k\nabla^2 T - f(x,t) = 0$$

$$BC = T(0,t) = T(1,t) = 0$$

$$IC = T(x,0) - T_0 = 0$$

The loss function of the neural network is then formulated as:

$$\mathscr{L}_{nn} = MSE_D + MSE_{BC} + MSE_{IC}$$

where,

$$MSE_D = \frac{1}{N_D}\sum_{n=1}^{N_D}(D-0)^2$$

$$MSE_{BC} = \frac{1}{N_{BC}}\sum_{n=1}^{N_{BC}}(BC-0)^2$$

$$MSE_{IC} = \frac{1}{N_{IC}}\sum_{n=1}^{N_{IC}}(IC-0)^2$$

A specified number of points referred to as N_D, N_{BC}, and N_{IC} indicate the number of coordinates used to constrain the solution during training. The loss MSE_D limits the solution in the domain, while MSE_{BC} at the boundary conditions, and MSE_{IC} at the initial conditions of the problem. The combined loss allows the neural network to learn weights that converge on a general solution provided with any, 'test set', or coordinates that are within the domain. In modern deep learning applications, automatic-differentation or the back-propagation algorithm is used to adjust the neural network weights for learning. The parameters are differentiated, a gradient descent algorithm is then used to allow for improvement relative to some training data. In this case, automatic-differentiation is used to calculate the derivatives (1st and 2nd order when necessary) with respect to the input coordinates, typically space and time, while the PDE is explicitly programmed into the loss function as shown above (Raissi, Perdikaris and Karniadakis (2019)). A gradient descent-based optimizer is then used on the loss function to adjust the weights and biases of the neural network for learning.

2.4 Partial Differential Equation Control

This section will define the state and action space for a PDE control problem, as well as an example control problem described by a physics-based PDE. In addition, an outline of current methods used for control of PDEs, with a focus on HVAC control methods will be provided.

2.4.1 PDE State Space

The state is represented by a temperature profile and a velocity flow vector field throughout the two-dimensional domain. The domain represents the extent of the room to be simulated when solving the PDE system. This state representation is shown as follows:

$$S = \{T_x, T_y, u_x, u_y\}$$

where the parameters T_x, and T_y represent the temperature in the x and y dimensions and u_x and u_y, the directional flow velocity at each point in space. To more accurately simulate a real-world system the agent will not have perfect information of the state. This more accurately reflects reality because it is unrealistic to know the temperature and flow velocity at every point in a space at any given time. While the actual state of the environment will be simulated by solving the relevant PDE with FEA using the Python library, FEniCS.

2.4.2 PDE Action Space

The agent must have a defined set of actions that it can perform to influence the environment. In this case, the agent must act on the environment through the HVAC unit. The HVAC unit has three main components relevant to the simulation, the furnace (to produce hot air), evaporator coil (to produce cool air), and the fans (to introduce airflow) (McQuiston, Parker and Spitler (2018)). Utilizing these components, the HVAC unit can introduce an airflow of either cold or hot air into the room. This is represented as:

$$A = \{P_T, P_{\dot{m}}\} \in [-1, +1] \times [0, 1]$$

where P_T is the temperature of the inflow air on a continuous power scale of the HVAC unit. Here -1 represents the HVAC unit using full power to output cold air, and +1 represents the HVAC on full power to produce hot air. The fan produces a mass flow rate $P_{\dot{m}}$ on a continuous scale from 0 to 1, where 0 is off and 1 is full power. Within the FEniCS simulation, the HVAC unit is represented on a portion of the boundary conditions. The

previously defined actions will effect the simulation by changing the flow and temperature of the defined portion on the boundary conditions.

2.4.3 Classical Control Methods

The industry standard for the control of HVAC systems is through a Proportional Integral Derivative (PID) controller. While PID controllers are estimated to be used in approximately 95% industrial HVAC controllers, other intelligent controllers such as fuzzy logic and pattern recognition controllers have recently been developed. However, these methods are still not widely used (O'Neill, Li and Williams (2016)). The PID controller is provided with a set-point, or the desired room temperature. One or more temperature sensors in a room are then used to guide the direction of adjustment the controller will make through the HVAC unit (Blasco (2012)).

A simplified model of a room can be represented by an ODE describing the thermal system generated by applying an energy balance (Yamazaki et al. (2011)):

$$C\frac{d\theta}{dt} = \omega_s(\theta_s - \theta) + \alpha(\theta_0 - \theta) + q_L$$

where:

C = heat capacity of the space,
θ_s = setpoint temperature,
θ = temperature read by sensor,
θ_0 = outside temperature,
α = transmittance factor,
q_L = thermal source term,
ω_s = heat from airflow rate

This energy balance describes the energy present in the room at any instance in time. This is equal to the difference in energy being added and removed from the domain. Specifically, this includes the energy being supplied or removed from the HVAC unit, the energy change through the walls of the room, and any heat source due to occupants or infiltration within the domain (Yamazaki et al. (2011)). The optimal control of this equation is the general goal of the PID controller. The most common PID control actions are binary on/off states, while modulating actions (continuous in range) is also fairly common in industry (McQuiston, Parker and Spitler (2018)). A diagram depicting the general pattern for binary or discrete control is shown in Figure 2.2.

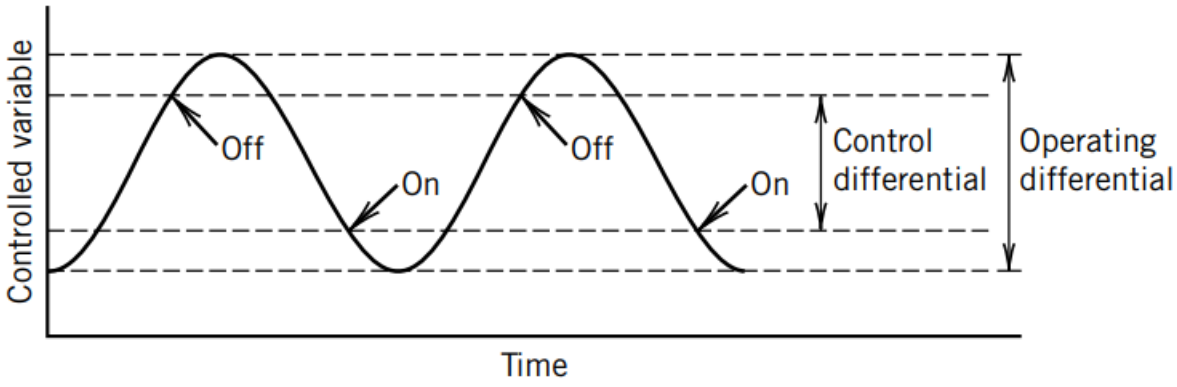

Figure 2.2: Control variable change with time for binary PID controller (McQuiston, Parker and Spitler (2018))

This diagram depicts a resulting high variance in the control variable (temperature), which can have a negative effect on energy efficiency, cost, as well as occupant comfort. The variance in the control variable due to a modulating PID actions demonstrates a marginal improvement as can be seen in Figure 2.3.

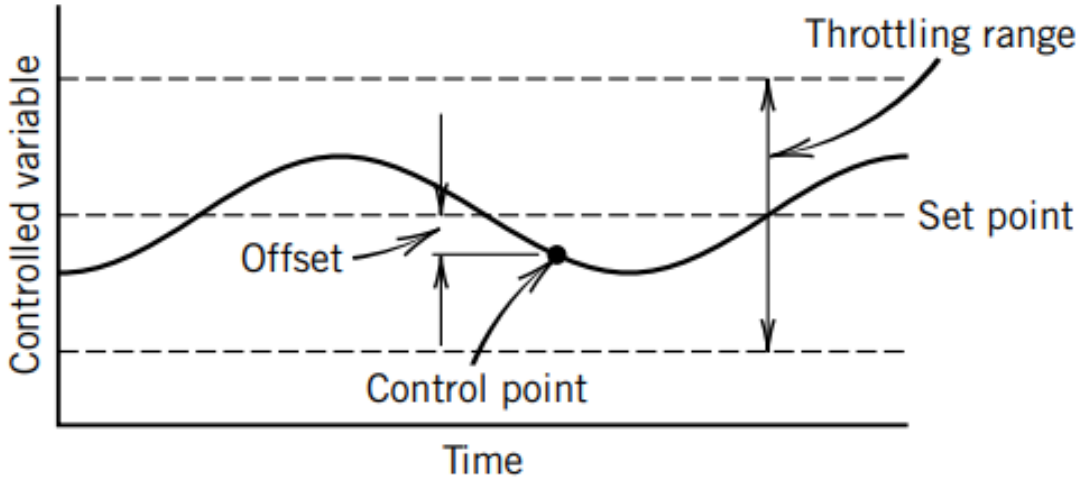

Figure 2.3: Control variable change with time for a modulating PID controller (McQuiston, Parker and Spitler (2018))

The modulating PID controller is likely to experience less variance in the control variable, however there is still significant error where the target temperature is not being met much of the time. This can lead to similar reduction in efficiency and less than optimal performance. In addition, HVAC PID controllers are often shipped with default tuning parameters, where the user must manually tune the parameters for a new environment. Conditions are also subject to change significantly due to seasonality. It is generally

recommended to tune the PID controller regularly to better optimize performance, which is highly labor intensive and subject to human error (O'Neill, Li and Williams (2016)).

2.4.4 Reinforcement Learning for Physics Control

Extensive research has been conducted involving the application of deep RL for control of physical systems. This typically includes an agent learning an optimal control strategy through some physical movement in the system. One application used a three-dimensional physics simulation, combined with RL and a genetic algorithm for training bipedal creatures to walk (Geijtenbeek, Panne and Stappen (2013)). This, along with a related study apply an RL agent to control a model of the creature skeletal and muscular system to produce bipedal locomotion with impressive results (Reda, Tao and Panne (2020)). Another study utilized RL, combined with imitation learning for real-world control of a robot. This allowed a human to guide a robot to swing a ball from a still position into a cup. The robot was then able to learn through human-guidance along with further trial and error, how to perform the task on its own (Tan and Kawamura (2011)). Kalmar et al introduced Module-based RL for use in robotics experiments. This merges elements from deep Q-learning, where the agent uses local controllers to break a large task into smaller sub-tasks by using prior information of the system (Kalmár, Szepesvári and Lorincz (1998)). Lastly, a recent success was demonstrated by formulating an actor-critic method using a policy-gradient algorithm to complete more than 20 physics-based tasks (Lillicrap et al. (2019)). The tasks included classic control problems such as the Cart-pole balancing problem to more complex tasks such as bipedal locomotion, and autonomous navigation of city streets.

Another avenue for physics-based RL is seen in the control of dynamic environments as opposed to a dynamic agent. In one study, deep RL is used to develop an optimal swimming strategy for a pair of fish in an incompressible flow (Novati et al. (2016)). This experiment involved solving the Navier-Stokes PDE with a dynamic viscosity similar to water to simulate the environment of the agent. The agent was found to improve in its energy efficiency by 20% compared to the baseline when interacting with the PDE governed environment. Another study (Xu et al. (2020)), presented one of the first applications of active flow control using a deep RL agent. This experiment involved a 2D Navier-Stokes simulation of a cylinder in a channel with incompressible flow. The goal was to minimize drag on the cylinder using small jets on the sides of the cylinder. This allowed the agent to control the wake separation by increasing the flow rate of the small jets. This RL application was able to successfully control the PDE-governed environment and reduce the drag from the flow by about 93%. A similar experiment (Fan et al. (2020)), applies deep RL to simulated flow control experiments. This study aimed to reduce drag on a cylinder in incompressible flow while simultaneously maximizing power gain. Here, the

agent could select any continuous rotational speed for two small cylinders downstream of the larger cylinder. Over 100 real-world experiments were conducted using an agent trained on a simulated environment. The agent was then tested by using the trained model on a physical system. This was the first successfully demonstrated application of RL in physical experiments for flow control.

Recent works have applied deep RL towards smart-building management using various simulation models. A survey by Wang and Hong (2020) on the real-world potential application of RL for building controls found that conventional rule-based control techniques are less than optimal due to the increasingly complex variables present in building management. It is becoming increasingly necessary for control systems to efficiently manage a number of trade-offs between multiple goals, including: occupant comfort, grid supply/demand, energy conservation, and carbon emissions. This change in control flexibility is leading to significant research increase in deep RL as evidenced by a 50% rise in HVAC control research since 2015 (Wang and Hong (2020)). Research by Raman et al. (2020) compared the performance of HVAC control using deep Q-Learning to industry standard model predictive control and PID control techniques. The experimental results demonstrated superior performance by the RL algorithm for maintaining temperature and humidity while also reducing energy usage when compared to the baseline. Another study by Azuatalam et al. (2020) proposes a novel RL architecture for efficient control of an HVAC system in smart commercial buildings. This work focused on increasing energy efficiency as opposed to improving the occupant comfort. The resulting simulations using the trained agents reduced energy consumption by up to 22% on a weekly basis when compared the their baseline rule-based controller. Dermardiros, Bucking and Athienitis (n.d.) created a building energy emulator using the OpenAI framework designed to investigate HVAC control techniques. A PPO RL agent was then trained to maintain occupant comfort in a house while minimizing HVAC energy usage. The performance was compared to a PID controller and showed less overall variance relative to the set-point, while also reducing energy consumption. Lastly, a study by Farahmand et al. (2016) focused on building management using HVAC as a PDE control problem. Specifically, the Regularized Fitted Q-iteration (RFQI) RL algorithm was trained to control a simplified environment governed by the convection-diffusion equation. This study simplified the governing equation by choosing a constant air-flow velocity, as opposed to computing it via the Navier-Stokes equations. In addition, this research focused on controlling a dynamic heat source in the domain, which is idendified as a "heat invader". The RL algorithm showed a 40% increase in reward when compared to the rule-based controller on the same environment.

2.5 Technology and Research Review

2.5.1 Environment and Agent Implementation

The agent will learn from interacting with a simulated environment governed by a PDE. To create this type of environment, a PDE solver is required. The Finite Element Method (FEM) will be used with the FEniCS Python library (Kirby (2004)) as the primary engine for the environment. FEM is one of the primary methods for approximating solutions to PDEs as well as in many physics simulations (Bi (2019)). The FEniCS library enables the user to solve a PDE by declaring the domain of the environment along with the PDE and its boundary conditions in its variational form. In addition, it has been used for similar research in physics and RL (Rabault and Kuhnle (2019), Belus et al. (2019), Ma et al. (2018), Verma, Novati and Koumoutsakos (2018)). The environment will be created with the OpenAI Gym (Brockman et al. (2016)) framework so to interface efficiently with externally designed RL agents. The deep RL agent implementation will use packages from the Tensorforce (Kuhnle, Schaarschmidt and Fricke (2017)) Python library, which contains highly maintained RL algorithms that can interface with custom ANN architectures. The neural network architecture will then be modified to incorporate a PINN layer using the Pytorch (Paszke et al. (2019)) Python libary. The project will initially aim to control environments governed by simple PDEs such as the diffusion equation and will ultimately implement an agent capable of more complex PDE control involving a system of PDEs.

2.5.2 Objectives

The primary objectives of this project are as follows:

- Produce a physics-informed deep reinforcement learning agent by merging concepts from computational physics with machine learning.
- Train the agent to successfully control a PDE governed environment.
- Evaluate the agents performance in controlling a PDE-governed environment to a standard RL agent.

Contingent on project progress, other secondary objectives may include:

- Expand the PDE complexity within the environment, this could include incorporating the Navier-Stokes equations with convection/diffusion PDE's.
- Incorporate a PINN layer to interface with more complex RL algorithms such as REINFORCE or A2C.
- Test various neural network architectures merged with a PINN layer used by the agents for optimal performance.

Chapter 3

Methodology

This work utilized a test-and-iterate approach to reach each milestone necessary to achieve the primary objectives. This included creating a configurable environment governed by the necessary PDEs, a standard RL agent and PINN-enabled design, as well as testing and experimentation. The remainder of this chapter will discuss the foundation of PDE control as a Markov Decision Process (MDP) along with its state space, action space, reward function, and the design of the experiments conducted.

3.1 PDE Control as a Markov Decision Process

Ideally, when the HVAC unit is initiated, the temperature in a room will reach the set point as quickly as possible, and will remain near constant to the desired temperature with minimal deviation. Initially, a diffusion environment is implemented as a proof-of-concept, it is then extended to include convection with airflow guided by the Navier-Stokes equations. In addition, each scenario will be tested with discrete and continuous action spaces. Therefore, multiple MDPs, describing increasingly complex scenarios, will be formulated with the necessary initial conditions, state space, action space, reward function, and terminal conditions. The MDPs will be discussed in the context of the standard RL agent and the PINN-enabled agent as well as multiple control scenarios involving the discrete and continuous action space.

3.1.1 State Space

The state representation is formulated with consideration for the information necessary for the agent to achieve optimal control while simply modelling reality and maintaining the Markov property. A more accurate proxy for the real-world can be achieved by constraining the agent to operate with imperfect information, meaning it doesn't have complete access to the full state of the environment. This is analogous to a standard HVAC

PID controller, which also operates with imperfect information by using the readings from a few sensors to determine its next action. The complete state space is then the full temperature profile or heat-map of the environment. This can be expressed as:

$$S_{diffusion} = \{T_{x,y}\} \; \forall \; [x, y] \in D$$

Similarly, the state of the convection-governed environment will include the full temperature profile along with a vector-field of the air-flow throughout the domain, shown as follows:

$$S_{convection} = \{T_{x,y}, u_x, u_y\} \; \forall \; [x, y] \in D$$

The parameter $T_{x,y}$ indicates the temperature at each x and y coordinate in the domain D. The parameters u_x and u_y represent the velocity at each coordinate in their respective directions. However, in either scenario, the agent is only able to observe finite temperature data at each sensor location. Three sensors will be considered, with its respective state representation:

$$S_{observation} = [T_{x_1,y_1}, T_{x_2,y_2}, T_{x_3,y_3}]$$

The sensor locations are chosen to more closely reflect a real-world scenario, with two being near to the corner edges and the third being closer to the center of the room. The sensor locations are at (-0.1,0.9), (0.9,0.9), and (0.5,0.45). This is also depicted in Figure 3.1, which shows a rendering of the diffusion environment at time zero.

3.1.2 Action Space

Initially, the diffusion environment is considered. In this scenario, the agent is able to interact with the environment by controlling the temperature of two boundaries at the top and bottom of the domain. This is given by:

$$a_{diffusion} = \{P_{temp}\} \times [-1, 1]$$

This indicates the binary control the agent possesses with a discrete action space. The agent controls the power delivered to the system, where the boundaries are either set to a hot (+1) or cold (-1) temperature. The set temperature of the boundaries affects the profile of the room by diffusing from a high to low temperature. In this case the boundaries, when hot, is $20°C$ above room temperature, while when set to cold, is $20°C$

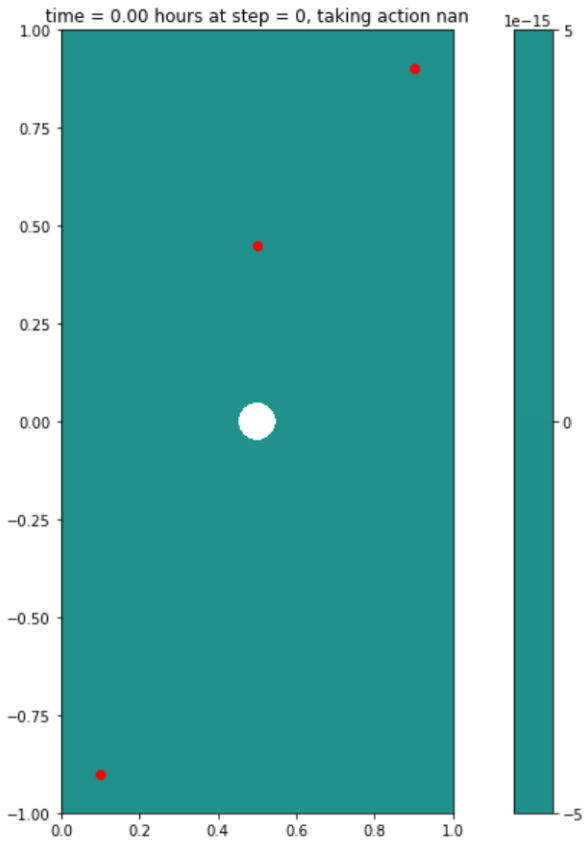

Figure 3.1: Diffusion-governed environment state after initialization at time zero, prior to any action has been take to affect the state.

below room temperature. Conversely, a more complex agent can exert continuous control over the environment as follows:

$$a_{diffusion} = \{P_{temp}\} \in [-1, 1]$$

Here, the agent is able to set the top and bottom boundary temperatures to any continuous value between -20°C and +20°C relative to room temperature, again by controlling the delivered power. This gives the agent more granular control and thus more influence over its environment. However, the search space for the optimal strategy is also expanded by an order of magnitude.

The convection environment introduces a new mechanism with which the agent can exert control over its environment. It is now able to control the fan speed of the HVAC system at each boundary. This induces an air-flow and thus a velocity vector-field throughout the domain, which is computed with the Navier-Stokes equations. This can influence the speed with which the heat is spread in the room and also increases the complexity and uncertainty in the environment which more closely approximates a real-world scenario. A discrete convection environment action space is therefore given by:

$$a_{convection} = \{P_{temp}, P_{fan}\} \times \{[-1, 1], [0, 0.5, 1.0]\}$$

This represents the control by the agent over the power delivered to the heating/cooling surfaces and to the fan. In this case the agent can set the temperature of the boundary to the maximum hot or cold setting, which remain +/- 20°C relative to room temperature. The agent can also set the power of the fan to zero, half, or full speed. This is similar to the continuous case:

$$a_{convection} = \{P_{temp}, P_{fan}\} \in \{[-1, 1], [0, 1]\}$$

In this scenario, the agent chooses any continuous value as the power delivered to both the heater/cooler and the fan. It is important to clarify that negative notation of the action space for the temperature is only representative of the direction of the temperature change. Hence, a (-1) will use the same power as a (+1), however it will have the inverse temperature affect on the environment. Conversely, the fan can only direct airflow into the room, and is not capable of becoming a vent (directing airflow out of the room), and thus has an action space between 0 and 1.

3.1.3 Reward Function

The reward function is crucial in defining the goals and shaping the ultimate behavior of the agent. Hence, it is designed to encode the following behavior:

- Minimize the bi-directional heat transfer $Q_{occupant}$, which occurs between the environment and the occupant.
- Reach a comfortable temperature as quickly as possible from an uncomfortable initial temperature profile.
- Minimize energy costs used by the HVAC unit.
- Maintain a consistently comfortable temperature profile for as long as possible during the simulation.

A reward function can be constructed based on each of previously mentioned behavioral goals. This is shown as follows:

$$R(s) = -R_{comfort}(s) - R_{cost}(s) + R_{streak}(S)$$

In reality, the comfort of an occupant in a space is determined by a number of complex environmental factors not limited to temperature, humidity, and wind chill. However, for

the purpose of this project, temperature will be the primary focus of control. Specifically, the temperature profile as it affects the comfort of an occupant. This is largely determined by the direction of heat transfer between the occupant and the environment. Using FEniCS, this is computed by integrating over the surface of the occupant, denoted $Q_{occupant}$, to calculate the instantaneous heat transfer in joules of energy. As previously mentioned, a negative heat transfer would indicate heat loss from the occupant, causing a cooling sensation, while a positive heat transfer implies heat gain, causing the occupant to feel a hot sensation. This entails an ideal heat transfer of zero ($Q_{target} = 0$) is desired. The first term of the reward function can therefore be defined as:

$$R_{comfort}(s) = (Q_{target} - Q_{Occupant})^2$$

This will effectively penalize the agent with the square of the total energy transferred between the occupant and the environment. The square of the term will cause the agent to minimize this differential with high importance because of its larger proportion with respect to the other terms in the reward function.

It's important for the agent to take energy costs into account so efficiency of the system is prioritized. This is addressed with the Q_{cost} term, which penalizes the agent based on inefficient use of the HVAC system. It's important to note the difference in power allocation by the agent in discrete and continuous scenarios. For simplicity and to eliminate action bias, it is assumed throughout all cases, that the inverse of any action will use the same amount of energy. For example, full power to the heater (action of 1) will have the same energy cost as full power used to cool the room (an action of -1). For the discrete cases, the energy costs will effectively counteract each other, eliminating any signal to influence the desired agent behavior. To address this, an action switching penalty is used as a form of energy cost. This is shown as:

$$R_{cost}(s) = \begin{cases} 0, & a_t = a_{t-1} \\ \delta_{switching}, & else \end{cases}$$

The parameter $\delta_{switching}$ is a predefined penalty which reflects a real-world higher energy cost of changing the temperature at the boundary. This has the added benefit of punishing the naive behavior of switching temperature every other time-step. Conversely, in the continuous cases the agent will choose any level of power with proportional energy costs shown as follows:

$$R_{cost}(s) = \alpha_{cost} \cdot |a_t|$$

Here, α_{cost} is the energy importance scaling factor, which will affect the priority of this measure to the agent. In addition, to account for HVAC inefficiency at higher power levels a custom Sigmoid function is developed to translate the agent actions a_t into a corresponding boundary temperature. This function is shown in equation 3.1.

$$\tau(a_t) = \frac{40}{1 + e^{-5x}} - 20 \tag{3.1}$$

This function maps the agents action a_t, or chosen power level, to a temperature with which the boundary will be set according to Figure 3.2.

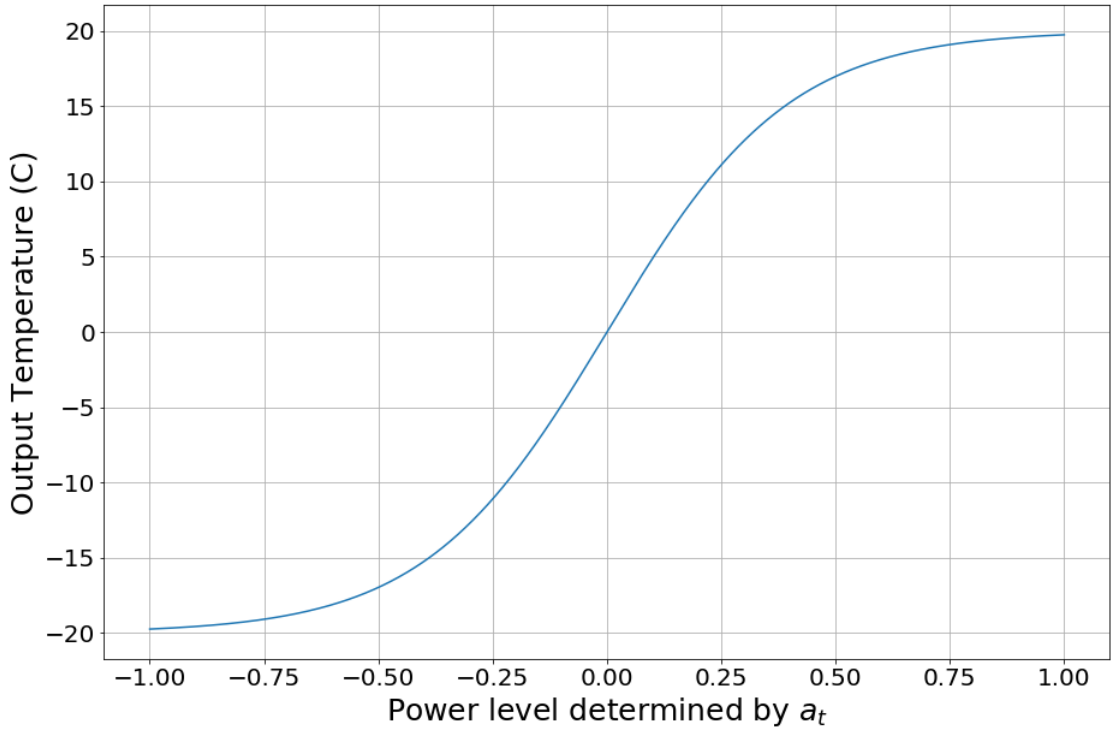

Figure 3.2: Efficiency relationship used to map the chosen power level by action a_t to the corresponding boundary temperature.

This function mirrors the loss in energy efficiency by causing diminishing returns with higher power levels. This will likely cause the trained agents to avoid using higher power levels, except when the loss in efficiency may be overcome by the gain in occupant comfort.

Lastly, the R_{streak} term is introduced to ensure the agent prioritizes consistency and is able to maintain occupant comfort for as long as possible. This behavior is encoded by rewarding the agent according to the number of time-steps X_{streak} where the heat transfer to the occupant is consecutively below some comfort threshold $\epsilon_{comfort}$. This relationship is shown as:

$$R_{streak}(s) = 5 \times 10^{-5} \cdot X_{streak}^2$$

This produces a curve which increasingly rewards the agent for larger streaks where $Q_{occupant}$ is below the comfort threshold, this relationship is shown in Figure 3.3.

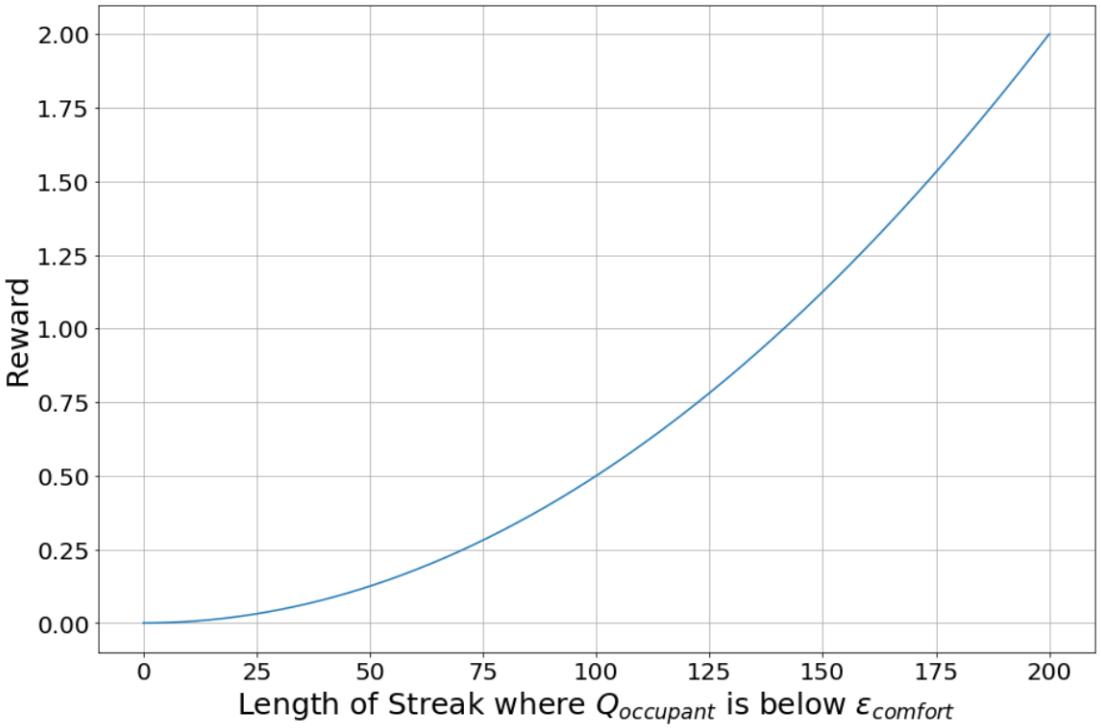

Figure 3.3: Corresponding reward to the length of streak where the heat transfer is below specified threshold $\epsilon_{comfort}$.

In a last effort to improve agent learning the reward will be scaled according to a decay function so that the range lies between $[-1, 0]$. Keeping the reward small has been shown to improve the learning of the neural network by preventing the gradients from becoming too large and potentially blowing up the weights. In addition, this has also been shown to improve stability of learning in RL resulting in a decreased variance in the reward curve (Karpathy (2016)). This relationship ensures that as the absolute reward becomes increasingly negative, $R(s) \rightarrow -1$ to give the agent consistent feedback. This relationship is shown in Figure 3.4.

3.1.4 Initial and Terminal Conditions

The environment requires consistent rules for beginning and ending an episode. It is desirable for the agent to be able to act accordingly to varied initial (though physically reasonable) temperature distributions throughout the environment. To ensure this behavior in the agent the initial temperature for each episode is chosen according to a uniform distribution between +/- $20°C$ from room temperature. This provides the agent with varied experience so that it is capable of reaching a comfortable temperature distribution in the environment as quickly as possible. Though, this will have the added effect of

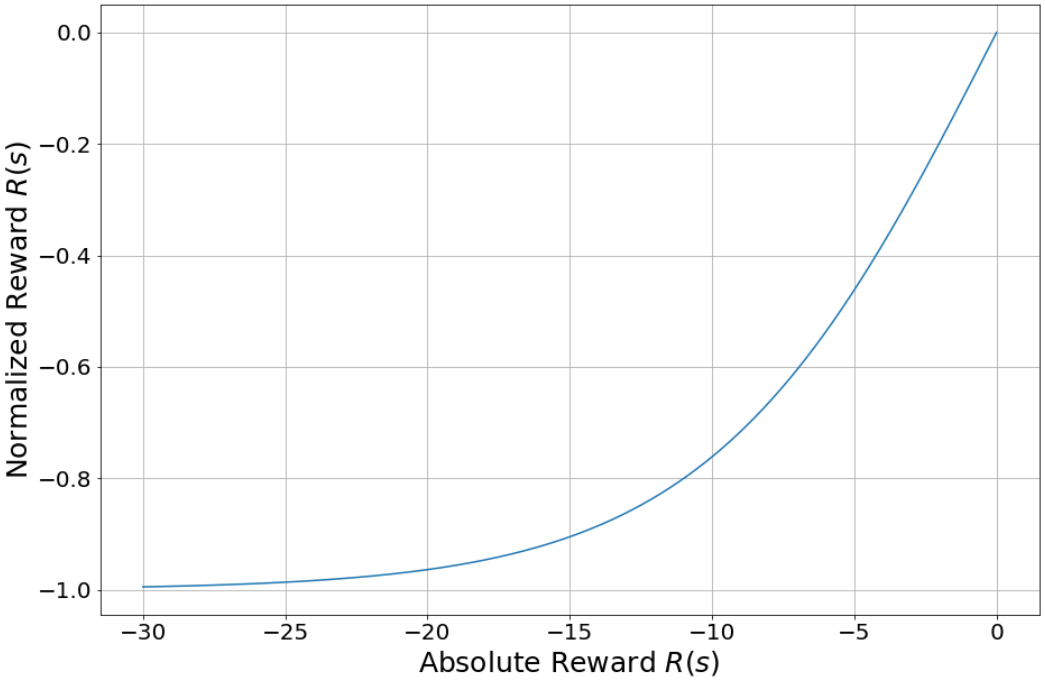

Figure 3.4: Relationship of the reward shaping function used to scale rewards between [-1,0].

increasing training difficulty due to the random nature of each episode. Lastly, a terminal state occurs when the number of time-steps $t = 200$, the episode is then ended and the environment is reset during training.

3.2 Experimental Design

The overall aim is to investigate the effect of the added PINN architecture on the learning and ultimate performance of an RL agent. This section will briefly describe the experiments conducted for each scenario. In addition, the parameters used for the corresponding RL algorithm and the experiment will be detailed. It should be noted that limited resources prevented hyper-parameter optimization, hence they were largely chosen based on industry heuristics and by experimentation. The performance data for all training experiments was recorded using Weights and Biases (Biewald (2020)), for later visualization and analysis. In addition, external agents utilized their default neural network architecture while the the DQN agent used architecture defined by Mnih et al. (2015).

3.2.1 Experiment 1: Discrete Diffusion

The first experiment used a custom DQN agent. A physics-informed instance was then compared to the standard DQN. Each instance was run on the same environment 5 times, and the mean of the resulting learning curves was taken. As another metric for

comparison, the trained agents were tested on the discrete diffusion environment with the same parameters. The agent hyper-parameters are shown in table 4.1. For this experiment, both the PINN-enabled agent and standard agent were trained for 400 runs or 80,000 steps. The buffer size, or agent memory, was set to 10,000 for each instance. No training occurred during this stage and therefore performance was not recorded for the first 50 episodes. The standard agent ran for about 3-4 hours, while the PINN-enabled agent ran for about 5 hours to complete the 400 episodes of training.

Table 3.1: Discrete action space diffusion environment with DQN Hyper-parameters.

hyper-parameter	Value	Description
replay buffer size	10,000	Number of experience tuples to store in memory
model update frequency	5	Frequency with which to train the value network
target update frequency	1000	Frequency with which to train the target network
learning rate	1×10^{-4}	learning rate for the agent neural network
discount rate	0.99	rate with which to discount future expected rewards
initial exploration rate	0.5	initial exploration rate by the agent
final exploration rate	0.05	minimum exploration rate
exploration decay rate	6×6^{-5}	rate used in the exploration decay function

3.2.2 Experiment 2: Continuous Diffusion

This experiment used the diffusion-governed environment with a continuous action space. Again, five training runs on each agent were conducted and the average of each was taken for better comparison. In order for the agent to interact with a continuous action space a more complex RL algorithm is necessary. A vanilla policy-gradient, known commonly as the REINFORCE algorithm, was used from the Tensorforce Python libary. This algorithm was chosen for its simplicity and ability to handle continuous action spaces. The PINN-agent wrapper class was used to allow the PINN to interface with the agent while interacting with the environment. Both instances were set to train for 400 episodes, however, due to hardware limits in memory (RAM) the PINN-enabled agent could not run past 280 episodes. The agent hyper-parameters are shown in Table 3.2

Table 3.2: Continuous action space diffusion environment with REINFORCE Hyper-parameters.

hyper-parameter	Value	Description
batch size	5	number of episodes used in network update
update frequency	1.0	frequency of updates relative to batch size
discount rate	0.99	rate with which to discount future expected rewards
Learning rate	1×10^{-4}	network learning rate
exploration rate	0.1	rate with which to explore the environment

3.2.3 Experiment 3: Continuous Convection-Diffusion

The final experiment is the most complex scenario for the agent to learn to control. The performance of the physics-informed agent is investigated on the convection-diffusion environment with a multi-action continuous action space. This scenario involves the agent controlling both the heating/cooling as well as the fan speed of the HVAC system. This environment instance operates on a smaller time-step because the addition of air-flow drastically affects the rate of heat transfer through the material. In addition, this environment is significantly more computationally expensive and takes longer to run each experiment. Due to this, only 3 runs were conducted for comparison between agent types. Each instance was set to train for 400 episodes, however session time-limits made this infeasible for the PINN-enabled agent. The PINN-enabled agent trained for approximately 12 hours to complete 250 episodes, while the standard agent trained for 7.5 hours to complete 400 episodes. Initially, the REINFORCE algorithm was tested, however the standard variant of this algorithm had difficulty learning to control an environment with this level of complexity. A more robust algorithm, known as Advantage Actor-Critic (A2C), was used to address this issue. Finally, the algorithm hyper-parameters are shown in Table 3.3.

Table 3.3: Continuous action space convection environment with A2C Hyper-parameters.

hyper-parameter	Value	Description
batch size	100	number of time-steps per network update
update frequency	1.0	frequency of updates relative to batch size
discount rate	0.99	rate with which to discount future expected rewards
Learning rate	1×10^{-4}	network learning rate
exploration rate	0.1	rate with which to explore the environment

Chapter 4

Software and Model Design

This chapter outlines the design and implementation of each of the primary software packages developed for this project. This includes the FEniCS-based environment Gym-HVAC, the PINN implementation using Pytorch, the physics-informed DQN agent, as well as the PINN-agent wrapper.

4.1 Environment Design

A contrived building management case is created for the exploration of a physics-informed approach to PDE control using deep RL. Two environment classes, simulating the transfer of heat, were created with varied levels of PDE complexity. The first environment is governed by the heat diffusion equation. This scenario simulates a room without heat transfer due to fluid flow or particle movement. This of course is not realistic. However, changing the thermal diffusivity κ of the environment medium, could allow for simulation of heat transfer through a number of materials with countless other applications in PDE/physics control. The second environment instance adds significant complexity by simulating heat transfer in a room with airflow. This environment is governed by the convection-diffusion equation and the Navier-Stokes equations. The convection environment is much more realistic in simulating the heat transfer in a room by modeling the movement of air as a vector-field and using this in the heat transfer computations. The remainder of this section will outline the governing PDEs for each environment case, detail the design, and provide an overview of the implementation.

4.1.1 Heat Diffusion Environment

The heat diffusion equation is shown in 4.1, where the boundary and initial conditions are shown in equations 4.2 - 4.4 as they relate to the environment. Equation 4.2 describes the temperature of the left and right boundary, which remain static throughout the simulation.

Equation 4.3 describes the temperature output of the HVAC unit at the top and bottom boundary. Lastly, equation 4.4 sets the initial temperature throughout the domain to an arbitrary initial temperature T_0.

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T + f(x, y, t) \tag{4.1}$$

$$T(0, y, t) = T(1, y, t) = 0 \tag{4.2}$$

$$T(x, -1, t) = T(x, +1, t) = \tau(a_{t-1}) \tag{4.3}$$

$$T(x, y, 0) = T_0 \tag{4.4}$$

The parameter κ, known as the thermal diffusivity, describes the rate of heat transfer through a material in $\frac{m^2}{hr}$. In addition, the HVAC power level chosen by the agent is converted to a temperature at each time-step using the function τ from equation 3.1. The resulting output of these equations is computed at each step using the FEniCS library for solving PDEs. Example renderings of the environment at different time-steps is shown in Figure 4.1.

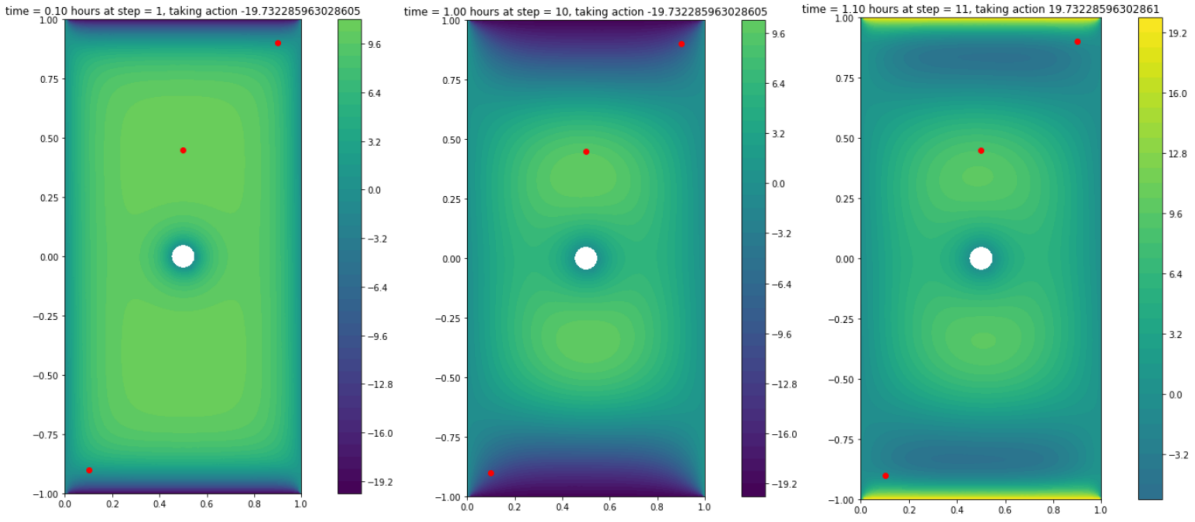

Figure 4.1: Renderings of the heat diffusion environment at various time-steps.

The above renderings provide a demonstration of the heat transfer simulation behavior in the environment. The effect of each action by the agent is calculated for the next time-step. To reiterate, the goal of the agent is to maximize comfort for the occupant in the center of the room, by maximizing its reward. The boundaries of the occupant

are set to a constant temperature of zero, which represents the average external body temperature. The left most rendering shows the simulation after the first step, where the initial temperature is $10°C$ and the HVAC boundary temperatures are set to $-20°C$. The center image shows the temperature profile of the domain after 10 steps of full-power cooling via the top and bottom boundary. The right image shows the temperature profile at the next time-step after the agent switched the boundary to heat the environment. One can see their will be a delay between the output of the HVAC boundaries and when the temperature will be felt by the occupant in the center. This delay is largely determined by the aforementioned thermal diffusivity κ, and is a variable the agent will have to learn to approximate in order to maintain a sufficiently comfortable environment. The relevant environmental parameters and their values are shown in Table 4.1

Table 4.1: Diffusion PDE environment parameters.

Variable	Value	Units	Description
κ	0.08	$\frac{m^2}{hr}$	Thermal diffusivity
dt	0.1	hours (hr)	time-step
$length$	200	steps	simulation length
t	20	hours (hr)	simulation length

4.1.2 Convection-Diffusion Environment

The convection-diffusion equation is shown in equation 4.5 along with its boundary and initial conditions (4.6 - 4.8). The descriptions of the boundary and initial conditions remain the same as the previously discussed diffusion-governed case. The primary addition is the convection term $(\nabla \cdot (uT))$, which is now a function of the airflow velocity throughout the domain.

$$\frac{\partial T}{\partial t} = \frac{1}{Pe}\nabla^2 T - \nabla \cdot (uT) + f(x, y, t) \tag{4.5}$$

$$T(0, y, t) = T(1, y, t) = 0 \tag{4.6}$$

$$T(x, -1, t) = T(x, +1, t) = \tau(a_{t-1}) \tag{4.7}$$

$$T(x, y, 0) = T_0 \tag{4.8}$$

The airflow velocity u is a vector field computed using the incompressible Navier-Stokes equation shown in equation 4.9.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \nabla^2 u = f(x, y, t) \tag{4.9}$$

Here, ν is the kinematic viscosity, which is a material property that dictates its ease of movement in the environment. A source for momentum within the domain can be described on the right side by the term $f(x, y, t)$. The variational form of these equations is programmed in FEniCS, which behaves as the simulation engine for the environment. Example renderings of the resulting environment at various time-steps is shown in Figure 4.2.

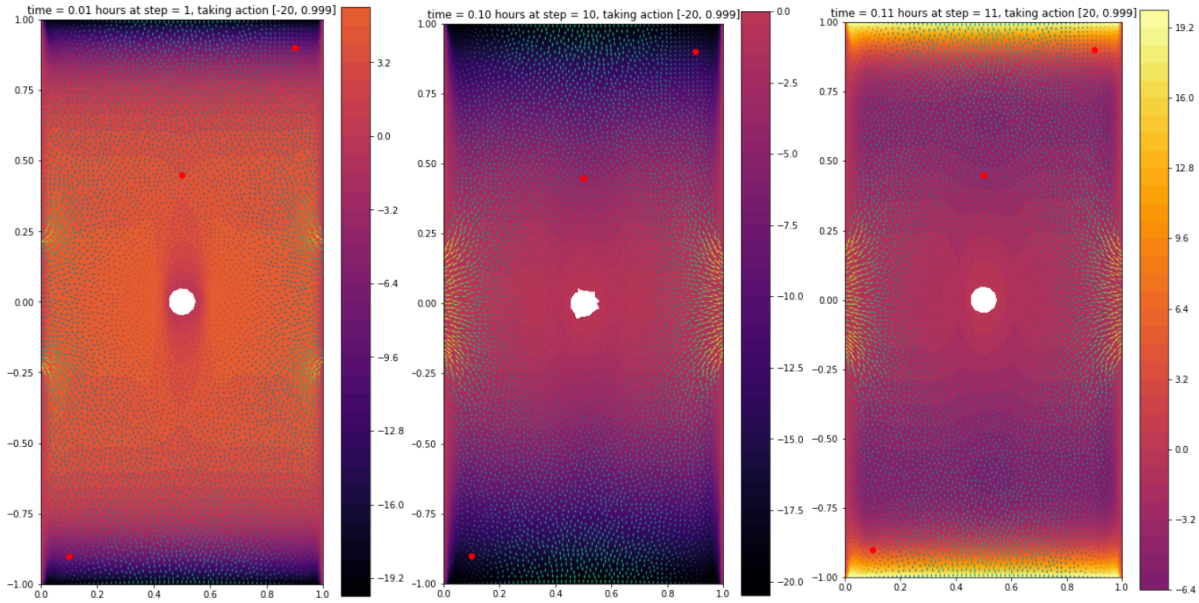

Figure 4.2: Renderings of the convection-diffusion environment at various time-steps.

To reiterate, in the convection case, the agent is able to control the top and bottom boundary conditions for both governing PDEs. The agent is then able to make two decisions that affect its environment. The first being, the power delivery to the heater/cooler, which sets the temperature of the top and bottom boundary. The second is the power of the fan, which influences the air-flow in the room. One will notice the renderings now depict the temperature distribution of the room along with a vector field of the airflow. The left image shows the environment state after the first step, with an initial temperature of $10°C$ throughout. The center rendering shows the full state after 10 time-steps of full-power to the cooler and to the fan. The right image shows the state immediately after switching the heater to use full power. Again, one will notice the time delay between the agent actions and when the occupant comfort level will be affected. In this case, this is largely a function of the Peclet (Pe) number and the kinematic viscosity ν, both a material property of air. In addition, the center left and right boundaries on each side of the occupant are set as the outflow, or vents for the room. Allowing for heat transfer

induced by airflow drastically changes the time-scale necessary for the agent to heat or cool the room. This change can be noticed in the step size and length of the simulation. A table of relevant PDE parameters is shown in Table 4.2

Table 4.2: Convection-Diffusion PDE parameters.

Variable	Value	Units	Description
Pe	0.1	$\frac{m^2}{hr}$	Peclet Number
ν	0.008	$\frac{m^2}{hr}$	Kinematic Viscosity
dt	0.01	hours (hr)	time-step
$length$	200	steps	simulation length
t	2	hours (hr)	simulation length

4.1.3 Design and Implementation

Gym-HVAC is designed in an object-oriented fashion, comprising a collection of classes that encapsulate relevant data and methods. The difference in behavior and interaction with the simulation engine require two separate classes to be implemented in the Gym-HVAC package. However, both instances `heat_diffusion` and `Convection` are implemented with the same OpenAI Gym framework, by inheriting the `Gym.Env` class, and utilize methods with identical names and functionality respective to their own physics model. The package was developed with the following requirements in mind:

- Implement the necessary MDPs described in Chapter 3, incorporating a highly configurable environment design.
- Provide simple computationally efficient framework to solve the necessary PDEs using FEniCS as the simulation engine.
- Maintain compatible interface with existing RL agent libraries and the standard RL framework commonly used in research.

The methods in each class interact with one another to maintain the PDE simulation, whilst also providing feedback and receiving input from the agent. A diagram depicting this interaction between the methods of the environment, denoted by the pill shaped blocks, as well as with the agent is shown in Figure 4.3. In addition, a list describing the functionality of each method follows.

- `_init_`: Receives environment parameters as input, declares all necessary initial attributes of the environment such as the action space, observation space, and reward range.
- `start_sim`: Declares the initial and boundary conditions in FEniCS. Instantiates the PDE in its variational form to be used in the FEniCS solve function.

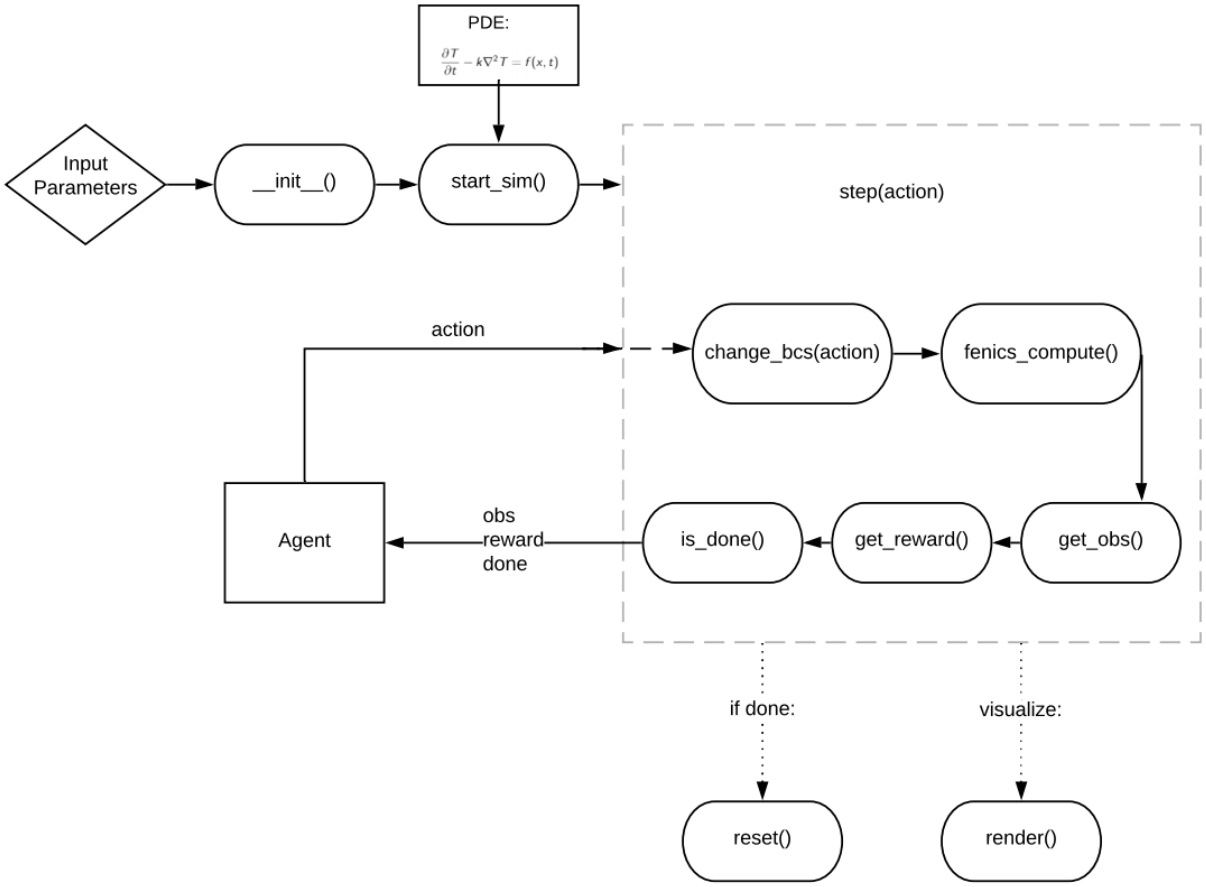

Figure 4.3: A diagram of the environment class interaction with the agent. In addition, the internal interactions between environment methods is shown.

- `step`: Receives the action to take from the agent. Utilizes an efficiency function to transform chosen power level to boundary value.
- `change_bcs`: Receives transformed action value. Adjusts boundaries to the specified action value.
- `fenics_compute`: Uses the FEniCS engine to compute the new temperature throughout the domain after the action has been taken.
- `get_obs`: Helper method to extract the temperature sensor values at the specified coordinates from the FEniCS engine.
- `get_reward`: Computes reward at current time step t using previously described conditions and equations.
- `is_done`: Checks if a terminal condition has been met.
- `reset`: Resets all variables to initial values and calls `start_sim` to re-initialize the

simulation with parameters input to `_init_`.

- `render`: Outputs a visualization of the current environment state with necessary metrics at the top.

4.2 PINN Design

The application of integrating a PINN into the RL agent design is intended to encode the necessary physics understanding allowing for more informed interactions with the environment. The PINN-enabled agent receives the same input accessible to a standard agent and is able to create an extrapolation based on this. Specifically, the PINN receives the same temperature sensor values and attempts to extrapolate these to a matrix of temperatures based on this input. The PINN agent is able to treat this extrapolation as an image, which necessitates the incorporation of a Convolutional Neural Network (CNN) as the feature extraction technique, where a typical agent uses a standard ANN. This section will detail the remaining PINN formulation for the relevant problem in the context of deep RL.

4.2.1 Neural Network Loss Formulation

Recall the convection-diffusion equation (4.10) with the following boundary and initial conditions (4.11 - 4.13):

$$\frac{\partial T}{\partial t} = \frac{1}{Pe}\nabla^2 T - \nabla \cdot (uT) + f(x, y, t) \tag{4.10}$$

$$f_{bc} : T(0, y, t) = T(1, y, t) = 0 = \hat{f}_{bc} \tag{4.11}$$

$$f_{bc} : T(x, -1, t) = T(x, +1, t) = \tau(a_{t-1}) = \hat{f}_{bc} \tag{4.12}$$

$$f_{ic} : T(x, y, 0) = T_0 = \hat{f}_{ic} \tag{4.13}$$

Where τ is the boundary temperature function (equation 3.1) applied to the previous action a_{t-1}, T_0 is an arbitrary initial temperature, and u is the flow velocity vector field. The left and right boundaries are static and described by equation 4.11, while the top and bottom boundaries (equation 4.12) represent the HVAC unit with dynamically adjusted values. Ultimately, the PINN will generate an approximate solution T for all coordinates $x, y \in t$. Equation 4.10 can be subject to minimization at an arbitrary number

of collocation points N_D by reformulating it to equal zero. After distributing the Laplacian (∇^2) and divergence (∇) operators this then can be written as equation 4.14, now denoted f_D.

$$f_D : \frac{\partial T}{\partial t} - \frac{1}{Pe}\frac{\partial^2 T}{\partial x^2} - \frac{1}{Pe}\frac{\partial^2 T}{\partial y^2} + u\frac{\partial T}{\partial x} + u\frac{\partial T}{\partial x} - f(x, y, t) = 0 = \hat{f}_D \tag{4.14}$$

The neural network loss function can then be formulated by computing the mean square error (MSE) of equations 4.11, 4.12, 4.13, and 4.14, this is shown in equation 4.15.

$$\mathscr{L}_{nn} = \frac{1}{N_f}\sum_{n=1}^{N_f}\left[f_D - \hat{f}_D\right]^2 + \frac{1}{N_{bc}}\sum_{n=1}^{N_{bc}}\left[f_{bc} - \hat{f}_{bc}\right]^2 + \frac{1}{N_{ic}}\sum_{n=1}^{N_{ic}}\left[f_{ic} - \hat{f}_{ic}\right]^2 \tag{4.15}$$

The loss function effectively allows the neural network to constrain the predicted solution at a specified number of coordinates (N_D, N_{bc}, N_{ic}) in the domain f_D, boundary conditions f_{bc}, and initial conditions f_{ic}. After sufficient training, the neural network is ultimately able to predict a general solution for T that minimizes the difference between f and $\hat{f}$ for all categories of solution. A depiction of this process is shown in Figure 4.4. As previously mentioned in section 2.3.2, the first and second derivatives in equation 4.14 are computed using automatic-differentiation, while the loss $\mathscr{L}_{nn}$ is minimized using back-propagation, primarily with the Adam optimizer in Pytorch.

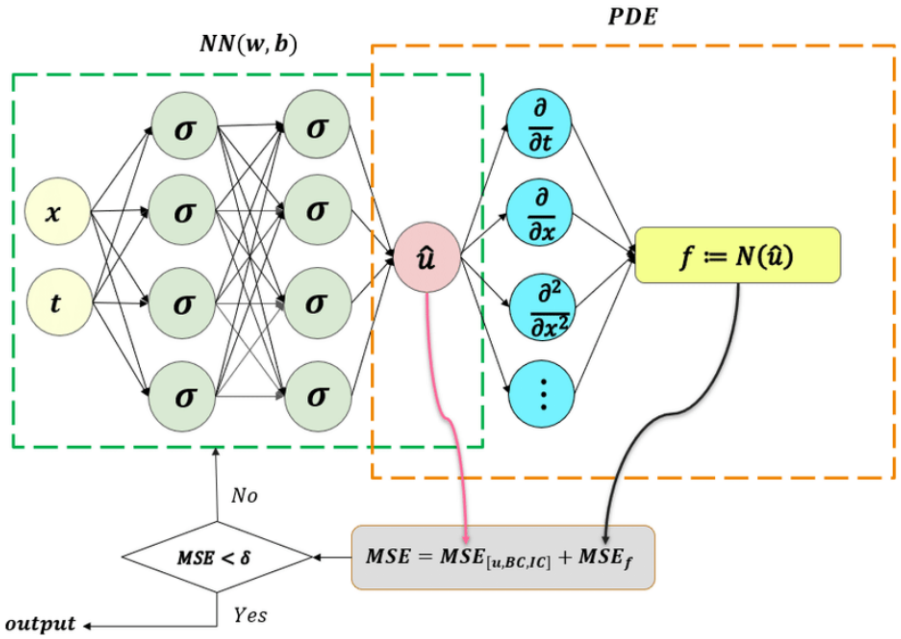

Figure 4.4: Physics-informed neural network architecture showing the modified loss function to include the various aspects of the encoded PDE (Guo et al. (2020))

4.2.2 PINN Algorithm

Several modifications were necessary for the PINN to interface properly in the RL agent neural network architecture. The detailed instructions for training the PINN using the previously described convection-diffusion formulation is shown in Algorithm 1. The remainder of this section will review the PINN algorithm, outline the modifications, as well as describe necessary parameters in the PDE and the PINN.

Algorithm 1 Training a Physics-Informed Neural Network

Result: $T \; \forall \; [x, y] \in t$

Input:$X_{ic}, X_{bc}, X_D, \hat{f}_{ic}, \hat{f}_{bc}$
Initialize $NN(\omega, \beta)$
$X_{ic} \leftarrow [x, y, t] \times N_{ic} \in IC$
$X_{bc} \leftarrow [x, y, t] \times N_{bc} \in BC$
$X_D \leftarrow [x, y, t] \times N_D \in D$
$\hat{f}_D \leftarrow [0, 0, 0, ...N_D]$
for *Epoch = 1, Max Epochs* **do**
    $f_{ic} = NN(X_{ic})$
    $f_{bc} = NN(X_{bc})$
    $T = NN(X_D)$
    dTdx, dTdy, dTdt = $(\frac{dT}{dx}, \frac{dT}{dy}, \frac{dT}{dt})$ w.r.t. [x,y,t] $\in$X$_D$
    d2Tdx2, d2Tdy2, d2Tdt2 = $(\frac{d^2T}{dx^2}, \frac{d^2T}{dy^2}, \frac{d^2T}{dt^2})$ w.r.t $[\frac{dT}{dx}, \frac{dT}{dy}, \frac{dT}{dt}]$
    f_D = dTdt - K * d2Tdx2 - K * d2Tdy2 + u*dTdx + u*dTdy - source
    $\mathscr{L}_{nn} = MSE(f_D, \hat{f}_D) + MSE(f_{bc}, \hat{f}_{bc}) + MSE(f_{ic}, \hat{f}_{ic})$
    Perform back-propagation step on $NN(\omega, \beta)$ using gradient descent to minimize $\mathscr{L}_{nn}$
end

The PINN training algorithm takes as input: the initial condition coordinates X_{ic}, the boundary condition coordinates X_{bc}, the collocation coordinates X_D, corresponding initial conditions $\hat{f}_{ic}$, and the corresponding boundary conditions $\hat{f}_{bc}$. All of which, except $\hat{f}_{bc}$, are static, while only the top and bottom boundary conditions are set dynamically based on the agents previous action using the function $\tau(a_{t-1})$. The weights ω of the neural network $NN(\omega, \beta)$ are assigned using a Glorot-Xavier initialization. This initializes the weights according to a Gaussian distribution with mean 0.0 and a specified variance (Glorot (2010)), while the bias β is initialized to zero for all neurons. The neural network used for this research consisted of 8 hidden layers each with 20 neurons. The coordinates used to train the network (X_{ic}, X_{bc}, X_D) are generated randomly in a custom function using a method known as latin hyper-cube sampling. This is a form of random sampling of coordinates known to reduce bias and variance (Jerome and Balesdent (2016)). These coordinates are freshly sampled for each step in the agent-environment interaction. As the PINN training occurs concurrently with the RL agent, this re-sampling allows the neural network greater exposure to varied examples of coordinates, allowing for better

generalization across the domain. The other inputs ($\hat{f}_{ic}$, $\hat{f}_{bc}$) are set by the PDE or the agent itself (for the HVAC boundaries). The number of coordinates used to constrain the solution for each case N_{ic}, N_{bc}, and N_D are shown in Table 4.3, along with a description of other parameters such as K and u.

Table 4.3: Parameter description used in the PINN.

Variable	Value	Units	Description
N_{ic}	100	NA	Initial Collocation Coordinates
N_{bc}	100	NA	Boundary Collocation Coordinates
N_D	10000	NA	Domain Collocation Coordinates
u	NA	$\frac{m}{s}$	Velocity
K	0.08	$\frac{m^2}{s}$	Thermal Diffusivity

Since the encoded PDE is arranged to equal zero, $\hat{f}_D$ is used to constrain the solution in the domain at the arbitrary number of points N_D. The PINN is able to utilize the new sensor information from each state to better inform its prediction. This is accomplished by appending the negative of the temperature values received by the sensors to the $\hat{f}_D$ array. In addition, the fixed sensor location coordinates are appended to the X_D array of collocation points. The negative of the sensor values is performed because $\hat{f}_D$ is on the right hand side of the encoded PDE. This effectively constrains the solution to the known imperfect observation in the domain. In addition, the observed temperature at the sensor locations are used in the source term, this generally constrains the PINN solution closer to the correct temperature in the simulation. Lastly, when training on the convection-diffusion environment, the flow velocity u is set as the agents chosen flow power to better inform its understanding of the state.

Once the neural network predicts the temperature at the initial conditions f_{ic}, at the boundary conditions f_{bc}, and in the domain T (lines 8, 9 , & 10), automatic-differentiation is used in Pytorch to compute the first and second derivatives of temperature T with respect to the coordinates and time (lines 11 & 12). The full PDE shown previously in equation 4.14 is then encoded in line 13 as f_D. The loss function, also previously shown in equation 4.15, is combined for each category of solution in line 14. Finally, the weights and biases of the neural network $NN(\omega, \beta)$ are adjusted using back-propagation and the Adam optimizer to minimize the loss function. This allows for the network to approximate a general solution for temperature throughout the domain. Two optimizer algorithms are used during training to minimize the loss: the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm and the Adaptive Moment Estimation (Adam) algorithm. The first is a quasi-Newton method which allows for a large initial step when minimizing the loss and is only used about once per environment episode. The second is a variation of stochastic gradient descent and acts as the primary optimizer in the PINN. The utilization of both algorithms is based on the original research by Raissi, Perdikaris

and Karniadakis (2019) for solving PDEs using a neural network. Lastly, the algorithm hyper-parameters are shown in Table 4.4.

Table 4.4: Optimizer Hyper-parameter values with corresponding algorithm.

Parameter	Value	Algorithm
Learning rate	0.1	L-BFGS
Tolerance	1×10^{-5}	L-BFGS
Iterations	10	L-BFGS
History Size	100	L-BFGS
Learning rate	1×10^{-4}	Adam
Tolerance	1×10^{-8}	Adam
Weight decay	0	Adam

4.2.3 PINN Implementation

The PINN package is designed to be highly configurable, simple to use, and computationally efficient. The `PINN` class instantiates a dynamic neural network architecture, which is specified as a list of layers, each with the number of neurons. This implementation uses Pytorch and inherits `torch.nn.Module`, which allows for the utilization of Pytorch attributes by the `PINN` class. A diagram showing the interactions between the `PINN` methods during training can be seen in Figure 4.5. In addition, a list describing each method and its functionality is provided below.

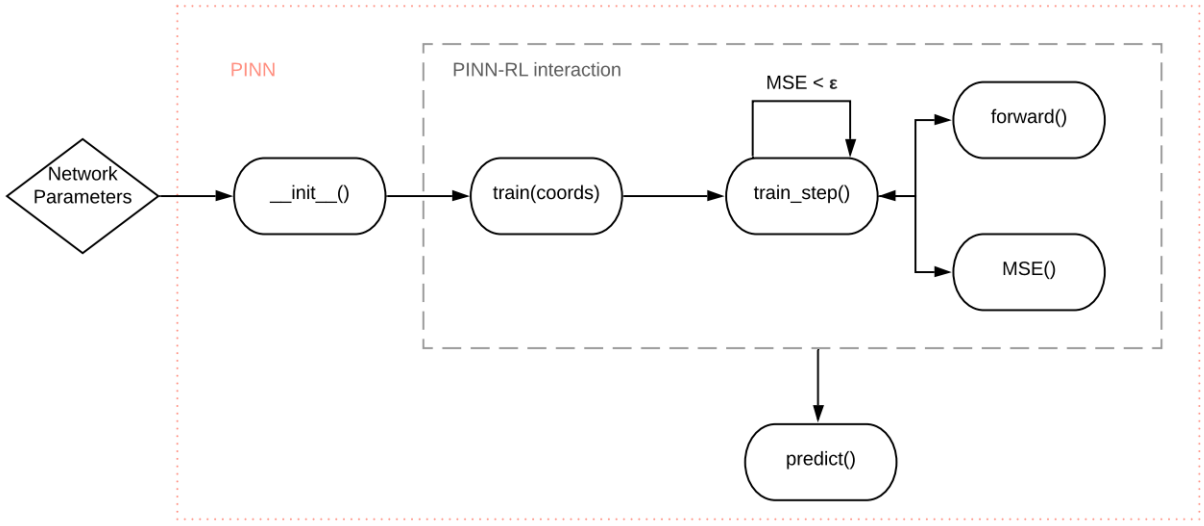

Figure 4.5: Interaction of the PINN methods during RL training.

- `_init_`: Instantiates neural network with received architecture and initializes weights and biases

- `train`: Declares optimizers and begins training loop, calls `train_step` until exit condition is met.
- `train_step`: Uses `forward` to predict on coordinates, performs automatic-differentation on resulting temperature prediction, and encodes the PDE into the loss function.
- `forward`: Performs a feed forward action through the neural network on the input coordinates.
- `MSE`: Receives f and $\hat{f}$ and computes the mean square error. Used primarily to simplify loss function.
- `predict`: Feeds forward on input coordinates through the network to output a matrix of temperature values throughout the domain. Processes output to be visualized as video or image if necessary.

4.3 Agent Composition

Initially, the PINN is implemented in a custom DQN agent based on pseudo-code from research by Mnih et al. (2015). This allowed for flexibility in how the PINN will interface with the agent neural network architecture. Subsequently, a wrapper package was implemented to allow for the PINN to interface with more complex RL algorithms from external libraries. This PINN wrapper was used with a policy-gradient based deep RL algorithm, known as REINFORCE, and A2C from the TensorForce library (Kuhnle, Schaarschmidt and Fricke (2017)). The remainder of this section will outline the design and implementation of the PINN-enabled DQN agent as well as the PINN wrapper for interfacing with more complex algorithms.

4.3.1 Physics-informed Deep Q-Learning

The algorithm was designed using standard Deep Q-Learning with experience replay, based on research by Mnih et al. (2015). The addition of the PINN layer in the neural network architecture is meant to encode prior physics understanding so that the agent will require less interactions to learn an optimal strategy. To ensure an equal comparison and efficient design, several criteria were considered in the design process:

- PINN-enabled agent must receive the same imperfect observation as the standard agent.
- Minimize increased computation by the addition of the PINN.
- Maximize consistency and accuracy of the PINN to ensure stable RL training.

The PINN can interface directly with the neural network within a deep RL agent. The output of the PINN is an extrapolation of the temperature within the environment, dependent only on the observed sensor values. Specifically, it is a matrix of temperature values for each coordinate in the domain. This can be thought of as a heat-map or image that can interface directly with a CNN for further feature extraction. This will ultimately allow for the development of an optimal physics-informed policy by the agent. In addition, the governing PDE is encoded into the PINN loss function as described in section 4.2.1 to provide the agent with a prior understanding of the environment. Finally, the PINN and the agent are trained concurrently with each step of the environment as shown in Figure 4.6. The modified deep Q-learning algorithm inspired by Mnih et al. (2015) is shown in Algorithm 2, where the primary adjustment is the utilization and training of the PINN.

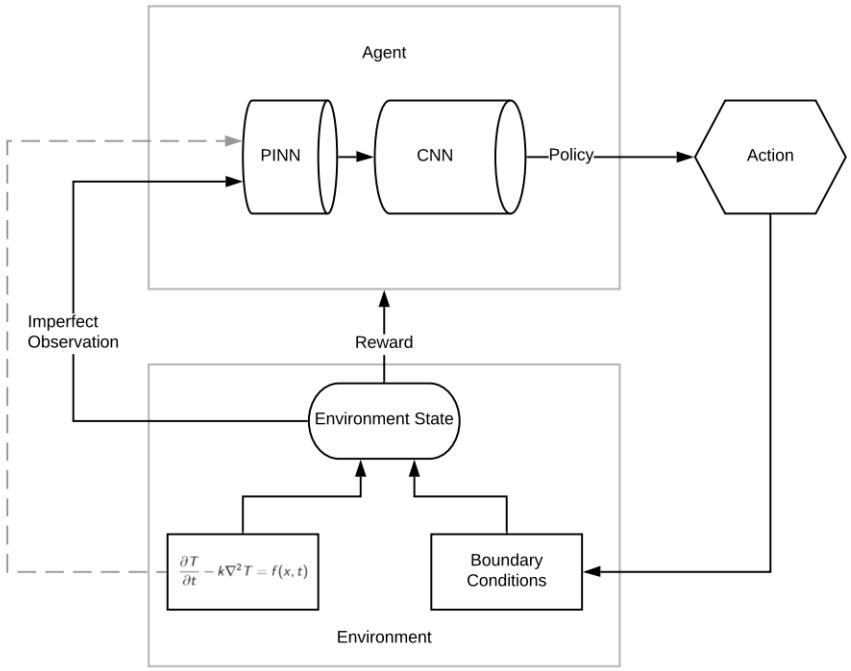

Figure 4.6: Physics-informed agent-environment interaction, depicting the environment governing-PDE encoded into the PINN.

Algorithm 2 Physics-informed deep Q-Learning with experience replay

Result: Trained agent with physics-informed policy π

17 Initialize replay memory D to capacity N

18 Initialize action function Q as $NN_{action}(\omega, \beta)$

19 Initialize target function $\hat{Q}$ as $NN_{target}(\hat{\omega}, \hat{\beta}) = NN_{action}(\omega, \beta)$

20 Initialize PINN as $\Psi = NN_{PDE}(\overline{\omega}, \overline{\beta})$

21 **for** *episode = 1, M* **do**

22 Initialize sequence $S_1 = x_1$

23 Extrapolate using PINN: $\phi_1 = \Psi(x_1)$

24 **for** *t = 1, T* **do**

25 With exploration probability ϵ select random action a_t
else select $a_t = argmax_a Q(\phi(S_t), a; \omega)$

26 Execute action a_t in environment, observe reward r_t, and sensor values x_{t+1}

27 Set HVAC boundaries according to agent previous action $\hat{f}_{bc} = \tau(a_{t-1})$

28 Randomly sample new coordinates within domain: X_{ic}, X_{bc}, X_D

29 **while** $\mathscr{L}_{nn} > \epsilon_{max}$ **do**

30 Train PINN: $\mathscr{L}_{nn} = \Psi(X_{ic}, X_{bc}, X_D)$

31 **end**

32 Extrapolate using PINN: $\phi_{t+1} = \Psi(x_{t+1})$

33 Store transition tuple $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

34 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

35 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \hat{\omega}), & else \end{cases}$

36 Perform gradient descent step on $(y_j - Q(\phi_j, a_j; \omega))^2$ with respect to ω

37 Every C steps set $\hat{Q} = Q$

38 **end**

39 **end**

The result of training using Algorithm 2 is an agent with a physics-informed policy capable of controlling the PDE-governed environment so to maximize occupant comfort. To achieve this, the replay memory, action network, target network, and PINN must be initialized for later training. The agent is then trained for a specified number of training episodes M. The environment is reset in line 22 and the PINN is used to extrapolate the initial sensor values in line 23 before the episode begins. The agent is then trained with each step in the environment until $T = 200$. The agent then chooses a random action according to probability ϵ, or the best known action according to the current policy produced by the action network Q. The action is executed in the environment and the agent is able to observe the reward r_t and the new sensor temperature values x_{t+1}. New training coordinates are then sampled (line 28) and the boundary coordinate values are updated according to the previous agent action (line 27). The PINN is then trained on the previously sampled coordinates until the loss $\mathscr{L}_{nn}$ is within some tolerance ϵ_{max}. This tolerance is one method to reduce increased computation due to the addition of the PINN.

Where a lower ϵ_{max} term will ensure more accurate and consistent predictions by the PINN, as well as more epochs to converge within the specified tolerance. The allowable training iterations within the PINN can also be adjusted to reduce computation or ensure consistency. Once trained, the PINN is applied to the sampled coordinates and the result is stored in a transition tuple for feature extraction by the CNN. The value estimate and target network are then updated according to the feedback based on the reward r_t (lines 35 & 36).

4.3.2 PINN-Agent Wrapper

The `PINN_agent` wrapper interfaces between the environment and the agent. This allows the PINN to train concurrently with the RL agent, while receiving the imperfect observation as input. `PINN_agent` receives the instantiated environment class so to utilize its methods during training. The wrapper also allows the PINN prediction to interface with the external agent. It is constructed to simply accomplish these tasks, while also including a number of helper methods to run an episode without training, render the environment for visualization, and to save the PINN model to disk for later use. A diagram depicting the internal interactions during training can be seen in Figure 4.7. A list describing each methods function is also provided.

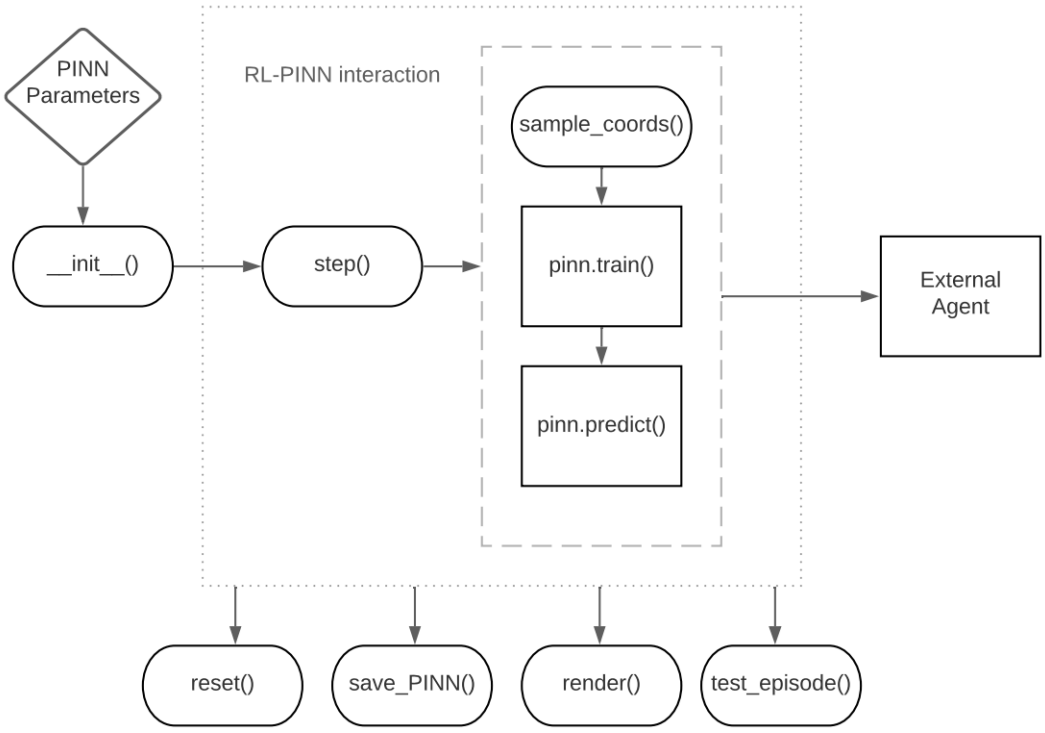

Figure 4.7: PINN-agent wrapper method interaction during RL training. The wrapper also calls methods from the PINN class, which is instantiated after initialization, as well as the external agent for further feature extraction.

- `_init_`: Instantiates a PINN and its parameters as internal attributes.
- `step`: Performs a step in the environment, calls `pinn.train` and `pinn.predict` to generate prediction, which is passed to the external agent.
- `sample_coords`: Uses latin hyper-cube sampling to generate new coordinates in each category of the domain.
- `pinn.train`: Trains the instantiated PINN until a terminal condition is met.
- `pinn.predict`: Predicts using the PINN on the coordinates in the domain.
- `reset`: Calls the environment reset method to re-initialize the physics simulation.
- `save_PINN`: Saves the PINN model to disk for later use.
- `render`: Calls the environment render method to visualize current state.
- `test_episode`: Runs an environment episode without training to evaluate and visualize agent performance.

Chapter 5

Experimental Results & Discussion

Three experiments were conducted to investigate the effect of the physics-informed architecture on the performance of the agent when training. Multiple agents were trained on the discrete diffusion, continuous diffusion, and multi-action continuous convection-diffusion scenarios. The averaged learning curves are compared between the physics-informed and standard agents for each case. Three RL algorithms were utilized: DQN, REINFORCE, and A2C, to address the increasingly complex control tasks. The trained agents were also tested on their respective scenarios as another metric of comparison. The remainder of this chapter will briefly present the results of each experiment in their own section as well as provide a discussion and critical analysis.

5.1 Diffusion with Discrete Action space

Multiple agents were trained utilizing the modified DQN algorithm described in section 4.3.1 on a discrete action space diffusion-governed environment. The learning curves for both the physics-informed and standard agents is shown in Figure 5.1. One will notice the physics-informed instance significantly outperforms the standard agent during training. Specifically, this is indicated by the improvement in sample efficiency, where the physics-informed agent converges to a higher reward in far less interactions with the PDE-governed environment. It can also be noted that the standard agent decreases in performance initially, which is likely due to the high initial exploration rate. Though this same exploration decay is used in both instances, the standard agent is unable to learn fast enough to overcome the added stability due to a high-degree of randomness. In addition, a summary of performance statistics is shown in Table 5.1. Convergence is defined as the point at which learning has changed less than 10% in the last 10 episodes. This experiment showed a 40 % reduction in episodes and a 32.4 % reduction in the time to converge. In addition, the test performance was increase by about 25.5 %.

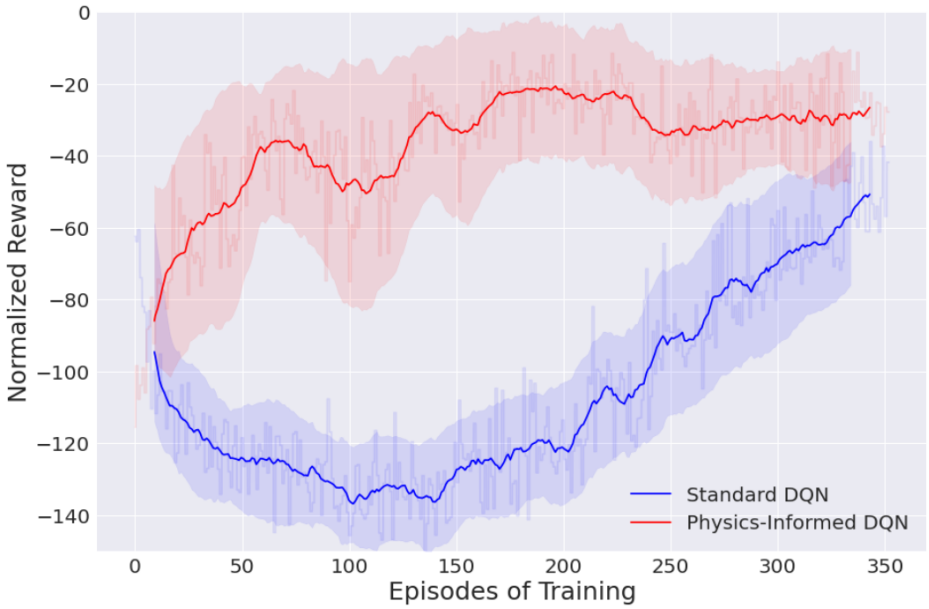

Figure 5.1: Learning curve of the standard agent compared to the PINN-enabled agent when training on the discrete action space diffusion environment.

Table 5.1: Summary of performance results for the discrete diffusion scenario.

Agent:	Standard	Physics-Informed	Difference (%)
Episodes to converge	400	240	-40.0
Time to converge	3.7 (hrs)	2.5 (hrs)	-32.4
Average training reward	-107.70	-36.70	65.9
Average test reward	-19.10	-14.23	25.5

5.2 Diffusion with Continuous Action space

The PINN-agent wrapper was used in combination with the REINFORCE algorithm from the Tensorforce library. A comparison of the physics-informed and standard agent reward curves during training is shown in Figure 5.2. Again, one will notice the significantly higher sample efficiency by the physics-informed agent. It is also worth noting that the physics-informed agent would typically start with a higher initial reward, as if it already had some pre-determined understanding of how not to behave. However, additional testing is necessary to isolate exactly why this is the case. One will also notice that the standard agent has a high standard deviation during much of its training. This demonstrates a significantly larger variability in training compared to the physics-informed agent. A summary of the performance statistics for each agent is shown in Table 5.2. The physics-informed agent converged in 64.3% less episodes and 17.5% quicker. Lastly, the performance during testing was very similar with only a 2.8% difference.

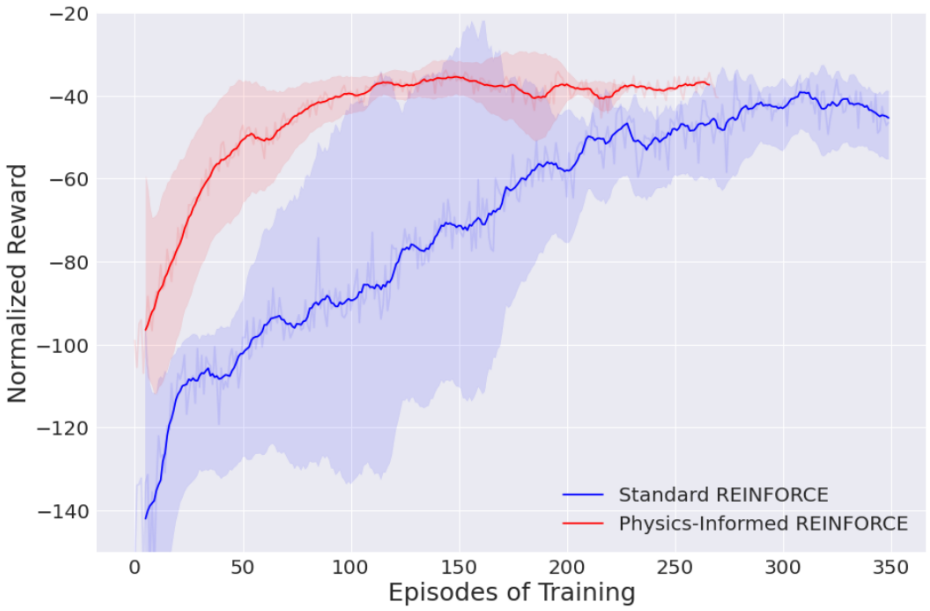

Figure 5.2: Learning curve with the standard agent compared to the physics-informed agent while training on the continuous action space diffusion environment.

Table 5.2: Summary of performance results for training with the continuous diffusion scenario.

Agent:	Standard	Physics-Informed	Difference (%)
Episodes to converge	292	104	-64.3
Time to converge	3.48 (hrs)	2.87 (hrs)	-17.5
Average training reward	-69.38	-45.20	34.9
Average test reward	-35.95	-36.98	-2.8

5.3 Convection with Continuous Action Space

Lastly, the A2C algorithm was trained to control the convection-diffusion environment. The averaged reward curves for the physics-informed and standard agents is shown in Figure 5.3. This experiment showed the highest disparity in performance between the physics-informed and standard agents. One instance of the standard agent was trained until near-similar performance to the physics-informed agent was reached, this is shown in Figure A.5. The drastic difference in performance may be due to the significant increase in complexity of the environment. This also reduced the episodes required to convergence by 92.5 % and the total time to converge by 88.7%. These summary metrics are shown in Table 5.3.

Table 5.3: Summary of performance results for the mutli-action continuous convection scenario.

Agent:	Standard	Physics-Informed	Difference (%)
Episodes to converge	322	24	-92.5
Time to converge	5.83 (hrs)	0.66 (hrs)	-88.7
Average training reward	-128.69	-92.99	27.7
Average test reward	-6.38	-5.60	12.2

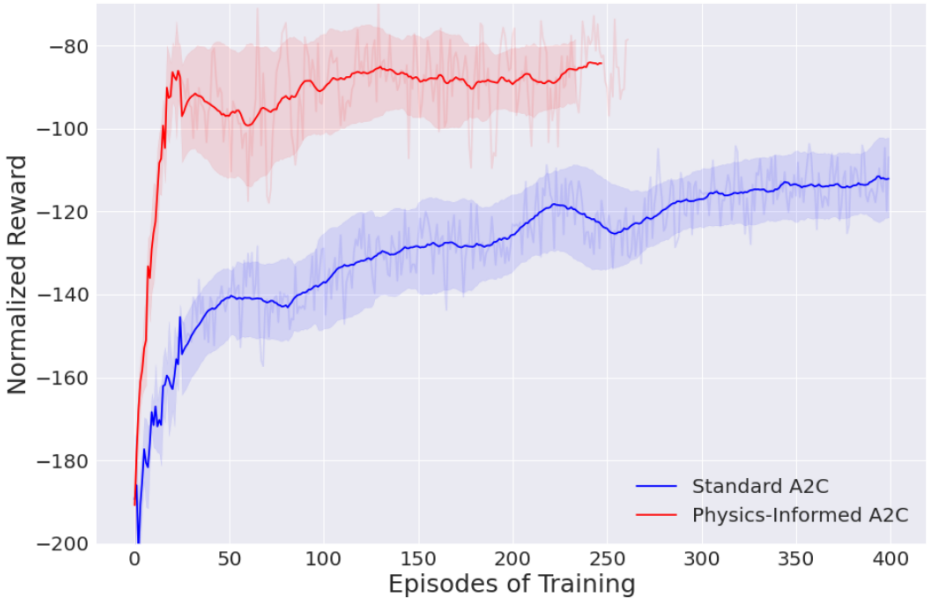

Figure 5.3: Learning curve comparing the standard agent to the PINN-enabled agent while training on the continuous action space convection-diffusion environment.

5.4 Discussion

One of the primary challenges when using RL techniques is the large number of environment interactions required for learning (Yu (2018)). This lack of sample efficiency towards learning complex control problems can render many tasks infeasible as they may require too much time or computational power to train an optimal policy. This work aims to address this issue for agents interacting with physics-governed environments by encoding the governing equations into the agent architecture. This is intended to integrate prior mathematical understanding of the environment into the agent design. This is achieved through the addition of a PINN layer, which interfaces with the typical feature extraction network present in a deep RL agent. The PINN layer receives the same imperfect observation and extrapolates upon this to provide a physics-governed prediction of the environment state. This may act as a regularization technique which, like in the PINN network itself, constrains the agent to take actions which interact optimally within physical laws, such as conservation of mass and momentum.

The primary results of a physics-informed technique improved sample efficiency and performance significantly, having the strongest effect on the most complex environment. The large disparity in performance on the most complex convection-diffusion scenario demonstrates that the PINN layer is likely working as intended. The increased complexity of this environment instance means it is inherently much more difficult to control. The results, shown in Figure 5.3, demonstrate the physics-informed agent has a remarkable advantage when learning to interact optimally with this environment compared to the standard agent. This finding could prove immensely useful for many complex control tasks, which operate in a physical environment. This work focused on investigating the application of this technique in a building management context. However, a physics-informed RL agent may be used to improve control and sample efficiency for many arenas of study in: robotics, fluid flow, chemical dispersion, and many other engineering disciplines.

Despite improved results for each experiment, it is important to acknowledge possible shortcomings. While the addition of the PINN layer seemed to be working as intended, this is not entirely certain. The exact cause of how and why the PINN layer increased performance is not isolated, this may be one avenue of study for future research. In addition, the lack of hyper-parameter optimization may effect or bias the results of learning. Ideally, one would compare the models using their optimal hyper-parameters, which would be used in a real-world control scenario. For example, its possible that the addition of the PINN layer may improve performance for one set of hyper-parameters but reduce performance for another set. This may also be worth investigating in a future study. Another avenue for future study might investigate utilizing this technique to control a building management simulation using software such as eQUEST (Hirsch (n.d.)). This would extend the complexity of the environment relative to the convection-diffusion scenario as well as provide long term control tasks lasting multiple months or seasons. In addition, using this simulation software would help in understanding how a physics-informed RL algorithm would be perform in a scenario which more closely reflects reality. Lastly, an important avenue for additional study is the exploration of the application of physics-informed RL algorithms for other physics-based control tasks. Robotics, flight, and chemical control are all potential applications which may yield promising results.

Chapter 6

Conclusions

This work proposes a novel technique to integrate prior mathematical understanding to improve RL sample efficiency and performance when interacting with a physics-based environment. This is achieved by the addition of a PINN layer, which interfaces with the RL agent neural network and encodes the environment's governing equation. Multiple experiments were conducted which investigate the performance of this method in the context of smart-building management. Prior surveys in this field have emphasized the increasing complexity required for HVAC control due to conflicting goals involving: occupant comfort, intermittent grid supply due to renewable energy sources, and managing energy consumption (Wang and Hong (2020)). These goals have motivated a significant increase in research for new techniques in HVAC control using deep RL. However, standard RL algorithms are time-consuming and data demanding during the training process. This is the primary motivation for this work, which aims to address both of these challenges as well as improve overall performance through the integration of prior physics understanding into the agent design.

6.1 Summary of Achievements

A novel "physics-informed" approach to RL is proposed. The central hypothesis of this work states that improved performance and sample efficiency can be achieved for a physics-based control problem through the addition of a PINN layer to RL agent design. This is intended to encode physical laws into the algorithm which provide the agent with a prior understanding of its environment. This hypothesis is confirmed through three experiments on scenarios with various levels of complexity. Three RL algorithms were trained: DQN, REINFORCE, and A2C on a discrete action space diffusion, continuous action space diffusion, and multi-action continuous convection-diffusion scenario. The physics-informed agent outperformed the standard agent in each experiment. In addition,

the largest disparity in performance occurred during training on the most complex convection-diffusion environment. This demonstrates a stronger advantage for the physics-informed agent when training on more challenging physics-based control tasks. In addition, a new highly configurable package, Gym-HVAC, is developed for the exploration of RL algorithms for various scenarios in HVAC control. This package contains the previously mentioned diffusion-governed case, which models the spread of heat throughout a room, as well as occupant comfort due to temperature. A second instance utilizes the convection-diffusion equation along with the Navier-Stokes equations to model heat transfer due to convection, diffusion, and airflow throughout the domain. The second control task is much more challenging due to the effect of the airflow on the heat transfer. Lastly, this work acts as a proof-of-concept for a physics-informed approach for other physics-based control problems in many engineering disciplines.

6.2 Limitations & Weaknesses

The limitations of a physics-informed approach fall into three primary categories:

- **Computational complexity**
  This physics-informed approach improves sample efficiency at the cost of additional computational complexity. This is due to the concurrent training of the PINN. In addition, depending on the encoded equation, may require multiple epochs to converge. This ultimately reduces the training speed significantly, compared to the standard agent. The results of this study indicate the reduced training speed is worth the improved sample efficiency, however, this may not be the case for other physics-based control problems. In addition to increased computational requirements, the physics-based approach is also significantly more memory intensive. This is because for each iteration, it is training and predicting with the PINN to produce multiple frames which are typically stored in the agents memory for experience replay. Considering trends in increasing computational power and memory resources, these challenges may not be an issue in future. However, it would likely be worth investigating methods to improve the algorithm to be less resource-intensive.

- **Model complexity**
  The addition of the PINN also increases the model complexity significantly. This added complexity is likely worth the improved performance, however, it still poses multiple challenges. RL algorithms are known for having many hyper-parameters, which can drastically affect training. Many RL hyper-parameters also effect one another, which causes significant challenges when performing hyper-parameter optimization. The addition of the PINN introduces a new layer of hyper-parameters, which complicate this challenge even further. While the intention of the physics-

informed approach is to encode prior physics understanding, it is not certain this is what is happening to improve the performance. The addition of the PINN layer introduces another level of complexity when interpreting why or how the agent is making decisions. While the output of the PINN at each step can be investigated, the algorithm becomes more like a black-box approach because it is uncertain how the PINN prediction is being used to inform its decision-making. Lastly, the physics-informed approach lacks an intuitive explanation. The idea of encoding physical laws into an agent make sense. However, the exact mechanism of the PINN layer is not easily understood.

- **Requires prior mathematical understanding**
  The PINN layer is encoded with a governing equation of the environment. This provides the agent with the underlying mathematical equation, by which it acquires prior understanding of its environment. Implying that this approach can only be used for control problems with well-understood mathematics. Developments in physics have identified countless phenomena governed by concisely-described equations. This of course is not true for every or even most control problems. This may limit the potential application of this approach to control tasks with previously defined governing mathematics.

6.3 Further Research

There are several possible avenues for further research related to this area. The primary three areas are highlighted as follows:

- **Physics-based control problems in other domains**
  An interesting avenue for further research would be in the application of physics-informed RL algorithms for various PDE and physics-based control problems. For example, deep RL has been applied to fluid control problems (Garnier et al. (2021)). Can a PINN encoded with the Navier-Stokes equations improve learning in this context? In addition, it would be interesting to investigate this application for tangentially encoded problems. For example, suppose an agent is encoded with a differential equation describing rocket motion. Would this then improve learning of other tangentially-related classical physics control problems, since it will provide understanding of the same laws of conservation of mass and momentum?

- **Apply physics-informed RL to simulation software or a real-world application**
  Utilizing physics-informed RL for control of more complex simulation software, such as eQUEST for building energy modeling, may yield valuable insights. This would

allow one to investigate algorithm performance when controlling for multiple factors in the environment such as humidity, temperature, lighting, and infiltration. This would introduce many real-world elements which are not taken into consideration during a purely physics-based simulation. This avenue for research may also provide insight into performance for long-term control scenarios lasting months or even between seasons. Lastly, investigating performance for this approach in a real-world application may offer insights into PDE-control as it relates to reality. Specifically, the physical phenomena is is well understood mathematically, however, reality is uncertain and contains many factors not under consideration in simulation.

- **Isolate causation and improve algorithm design**
  A final, likely more technical route for further investigation, is related to understanding the internal mechanisms of decision-making. The agent feature extraction network (CNN) interfaces directly with the PINN during training. However, it is not clear how the CNN is utilizing the PINN prediction to inform its decisions. In addition, several design choices may not be optimal. For example, is it better to insert the PINN prediction into the agent memory for experience replay, or to merely use the imperfect observation and to predict with the PINN as needed? Investigating similar questions may inform a more efficient and higher-performing physics-informed RL architecture.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems [Online]. Software available from tensorflow.org. Available from: `https://www.tensorflow.org/`.

Azuatalam, D., Lee, W.L., Nijs, F. de and Liebman, A., 2020. Reinforcement learning for whole-building hvac control and demand response. *Energy and ai* [Online], 2, p.100020. Available from: `https://doi.org/10.1016/j.egyai.2020.100020`.

Becker, R. and Vexler, B., 2007. Optimal control of the convection-diffusion equation using stabilized finite element methods. *Numerische mathematik* [Online], 106(3), p.349–367. Available from: `https://doi.org/10.1007/s00211-007-0067-0`.

Bellman, R.E. and Dreyfus, S.E., 1962. *Applied dynamic programming.* Princeton University Press.

Belus, V., Rabault, J., Viquerat, J., Che, Z., Hachem, E. and Reglade, U., 2019. Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film. *Aip advances* [Online], 9(12), p.125014. Available from: `https://doi.org/10.1063/1.5132378`.

Bi, Z., 2019. *Finite element analysis applications.* Academic Press.

Biewald, L., 2020. Experiment tracking with weights and biases [Online]. Software available from wandb.com. Available from: `https://www.wandb.com/`.

Blasco, Carlos, M.J.B.I.L.A., 2012. Modelling and pid control of hvac system according to energy efficiency and comfort criteria. *Sustainability in energy and buildings smart innovation, systems and technologies* [Online], p.365–374. Available from: `https://doi.org/10.1007/978-3-642-27509-8_31`.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. Openai gym. *Corr* [Online], abs/1606.01540. 1606.01540, Available from: http://arxiv.org/abs/1606.01540.

Campbell, J., 2015. Chapter 5 - solidification structure [Online]. In: J. Campbell, ed. *Complete casting handbook (second edition)*. Boston: Butterworth-Heinemann, pp.163–222. Second edition ed. Available from: https://doi.org/https://doi.org/10.1016/B978-0-444-63509-9.00005-4.

Chorin, A.J., 1968. Numerical solution of the navier-stokes equations. *Mathematics of computation* [Online], 22(104), p.745–745. Available from: https://doi.org/10.1090/s0025-5718-1968-0242392-2.

Demirel, Y., 2002. Chapter 8 - diffusion [Online]. In: Y. Demirel, ed. *Nonequilibrium thermodynamics*. Amsterdam: Elsevier Science, pp.206–233. Available from: https://doi.org/https://doi.org/10.1016/B978-044450886-7/50008-4.

Dermardiros, V., Bucking, S. and Athienitis, A.K., n.d. A simplified building controls environment with a reinforcement learning application. *Proceedings of building simulation 2019: 16th conference of ibpsa* [Online]. Available from: https://doi.org/10.26868/25222708.2019.211427.

Fan, D., Yang, L., Wang, Z., Triantafyllou, M.S. and Karniadakis, G.E., 2020. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the national academy of sciences* [Online], 117(42), p.26091–26098. Available from: https://doi.org/10.1073/pnas.2004939117.

Farahmand, A.m., Nabi, S., Grover, P. and Nikovski, D.N., 2016. Learning to control partial differential equations: Regularized fitted q-iteration approach. *2016 ieee 55th conference on decision and control (cdc)* [Online]. Available from: https://doi.org/10.1109/cdc.2016.7798966.

Garnier, P., Viquerat, J., Rabault, J., Larcher, A., Kuhnle, A. and Hachem, E., 2021. A review on deep reinforcement learning for fluid mechanics. *Computers fluids* [Online], p.104973. Available from: https://doi.org/10.1016/j.compfluid.2021.104973.

Geijtenbeek, T., Panne, M.V.D. and Stappen, A.F.V.D., 2013. Flexible muscle-based locomotion for bipedal creatures. *Acm transactions on graphics* [Online], 32(6), p.1–11. Available from: https://doi.org/10.1145/2508363.2508399.

Glorot, X., 2010. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp.249–256.

Goecks, V.G., Leal, P.B., White, T., Valasek, J. and Hartl, D.J., 2018. Control of

morphing wing shapes with deep reinforcement learning. *2018 aiaa information systems-aiaa infotech @ aerospace* [Online]. Available from: https://doi.org/10.2514/6.2018-2139.

Guo, Y., Cao, X., Liu, B. and Gao, M., 2020. Solving partial differential equations using deep learning and physical constraints. *Applied sciences* [Online], 10(17), p.5917. Available from: https://doi.org/10.3390/app10175917.

Hirsch, J., n.d. Available from: https://www.doe2.com/equest/.

Jerome, M. and Balesdent, M., 2016. *Reliability based approaches*, Woodhead Publishing/Elsevier, p.88–108.

Kalmár, Z., Szepesvári, C. and Lorincz, A., 1998. Modular reinforcement learning: An application to a real robot task. *Learning robots lecture notes in computer science* [Online], p.29–45. Available from: https://doi.org/10.1007/3-540-49240-2_3.

Karpathy, A., 2016. Deep reinforcement learning: Pong from pixels [Online]. Available from: http://karpathy.github.io/2016/05/31/rl/.

Kirby, R.C., 2004. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *Acm transactions on mathematical software* [Online], 30(4), pp.502–516. Available from: https://doi.org/10.1145/1039813.1039820.

Kirkwood, J., 2018. Three important equations. *Mathematical physics with partial differential equations* [Online], p.217–236. Available from: https://doi.org/10.1016/b978-0-12-814759-7.00005-3.

Kuhnle, A., Schaarschmidt, M. and Fricke, K., 2017. Tensorforce: a tensorflow library for applied reinforcement learning [Online]. Web page. Available from: https://github.com/tensorforce/tensorforce.

Lagaris, I., Likas, A. and Fotiadis, D., 1998. Artificial neural networks for solving ordinary and partial differential equations. *Ieee transactions on neural networks* [Online], 9(5), p.987–1000. Available from: https://doi.org/10.1109/72.712178.

Larson, M.G. and Bengzon, F., 2015. *The finite element method: Theory, implementation, and applications*. Springer Berlin.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2019. Continuous control with deep reinforcement learning. 1509.02971.

Lin, T.R., Penney, D., Pedram, M. and Chen, L., 2020. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. *2020 ieee international symposium on high performance computer architecture (hpca)* [Online]. Available from: https://doi.org/10.1109/hpca47549.2020.00018.

Liu, R., Nageotte, F., Zanne, P., Mathelin, M.D. and Dresp-Langley, B., 2021. Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. *Robotics* [Online], 10(1), p.22. Available from: `https://doi.org/10.3390/robotics10010022`.

Ma, P., Tian, Y., Pan, Z., Ren, B. and Manocha, D., 2018. Fluid directed rigid body control using deep reinforcement learning. *Acm transactions on graphics* [Online], 37(4), p.1–11. Available from: `https://doi.org/10.1145/3197517.3201334`.

McQuiston, F.C., Parker, J.D. and Spitler, J.D., 2018. *Heating, ventilating, and air conditioning: analysis and design*. Wiley India Pvt. Ltd.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature* [Online], 518(7540), pp.529–533. Available from: `http://dx.doi.org/10.1038/nature14236`.

Novati, G., Verma, S., Alexeev, D., Rossinelli, D., Rees, W. van and Koumoutsakos, P., 2016. Synchronised swimming of two fish.

Owhadi, H., 2015. Bayesian numerical homogenization. *Multiscale modeling & simulation* [Online], 13(3), pp.812–828. `https://doi.org/10.1137/140974596`, Available from: `https://doi.org/10.1137/140974596`.

O'Neill, Z., Li, Y. and Williams, K., 2016. Hvac control loop performance assessment: A critical review (1587-rp). *Science and technology for the built environment* [Online], 23(4), p.619–636. Available from: `https://doi.org/10.1080/23744731.2016.1239466`.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library [Online]. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds. *Advances in neural information processing systems 32*. Curran Associates, Inc., pp.8024–8035. Available from: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Rabault, J. and Kuhnle, A., 2019. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of fluids* [Online], 31(9), p.094105. Available from: `https://doi.org/10.1063/1.5116415`.

Raissi, M. and Karniadakis, G.E., 2017. Machine learning of linear differential equations

using gaussian processes. *Corr* [Online], abs/1701.02440. 1701.02440, Available from: http://arxiv.org/abs/1701.02440.

Raissi, M., Perdikaris, P. and Karniadakis, G., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of computational physics* [Online], 378, p.686–707. Available from: https://doi.org/10.1016/j.jcp.2018.10.045.

Raman, N.S., Devraj, A.M., Barooah, P. and Meyn, S.P., 2020. Reinforcement learning for control of building hvac systems. *2020 american control conference (acc)* [Online]. Available from: https://doi.org/10.23919/acc45564.2020.9147629.

Reda, D., Tao, T. and Panne, M.V.D., 2020. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. *Motion, interaction and games* [Online]. Available from: https://doi.org/10.1145/3424636.3426907.

Saenz-Aguirre, A., Zulueta, E., Fernandez-Gamiz, U., Lozano, J. and Lopez-Guede, J., 2019. Artificial neural network based reinforcement learning for wind turbine yaw control. *Energies* [Online], 12(3), p.436. Available from: https://doi.org/10.3390/en12030436.

Silver, D., 2019. Deepmind rl lecture series. Available from: https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning : an introduction*, Adaptive computation and machine learning series. Second edition. ed. Cambridge, Massachusetts: Bradford Books.

Tan, H. and Kawamura, K., 2011. A computational framework for integrating robotic exploration and human demonstration in imitation learning. *2011 ieee international conference on systems, man, and cybernetics* [Online]. Available from: https://doi.org/10.1109/icsmc.2011.6084053.

Verma, S., Novati, G. and Koumoutsakos, P., 2018. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the national academy of sciences* [Online], 115(23), p.5849–5854. Available from: https://doi.org/10.1073/pnas.1800923115.

Wang, Z. and Hong, T., 2020. Reinforcement learning for building controls: The opportunities and challenges. *Applied energy* [Online], 269, p.115036. Available from: https://doi.org/10.1016/j.apenergy.2020.115036.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning* [Online], p.5–32. Available from: https://doi.org/10.1007/978-1-4615-3618-5_2.

Xu, H., Zhang, W., Deng, J. and Rabault, J., 2020. Active flow control with rotating cylinders by an artificial neural network trained by deep reinforcement learning. *Journal of hydrodynamics* [Online], 32(2), p.254–258. Available from: `https://doi.org/10.1007/s42241-020-0027-z`.

Yamazaki, T., Yamakawa, Y., Kamimura, K. and Kurosu, S., 2011. Air-conditioning pid control system with adjustable reset to offset thermal loads upsets. *Advances in pid control* [Online]. Available from: `https://doi.org/10.5772/18818`.

Yu, Y., 2018. Towards sample efficient reinforcement learning. *Proceedings of the twenty-seventh international joint conference on artificial intelligence* [Online]. Available from: `https://doi.org/10.24963/ijcai.2018/820`.

Zill, D.G. and El-Iraki, A., 2018. *A first course in differential equations with modeling applications*. Cengage Learning.

Appendix A

Supplementary Results

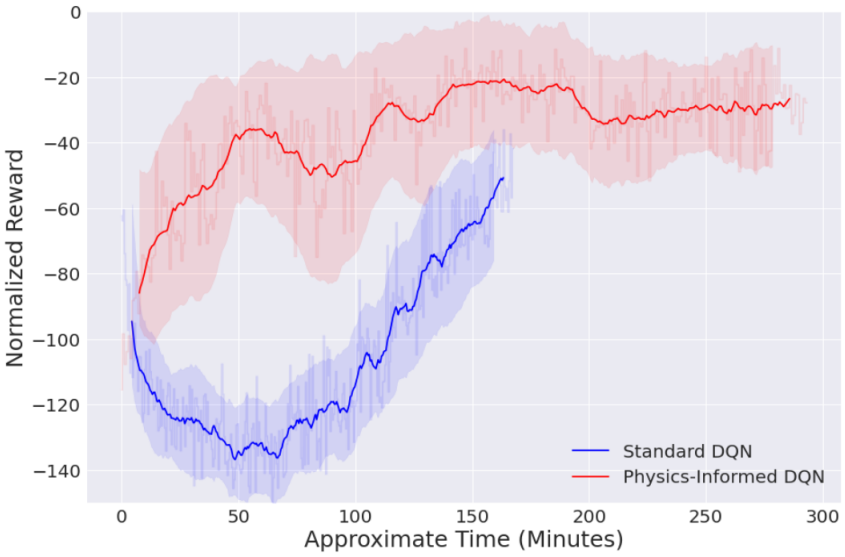

Figure A.1: Learning curve measure against training time of the standard agent compared to the PINN-enabled agent when training on the discrete action space diffusion environment useing Deep Q-learning.

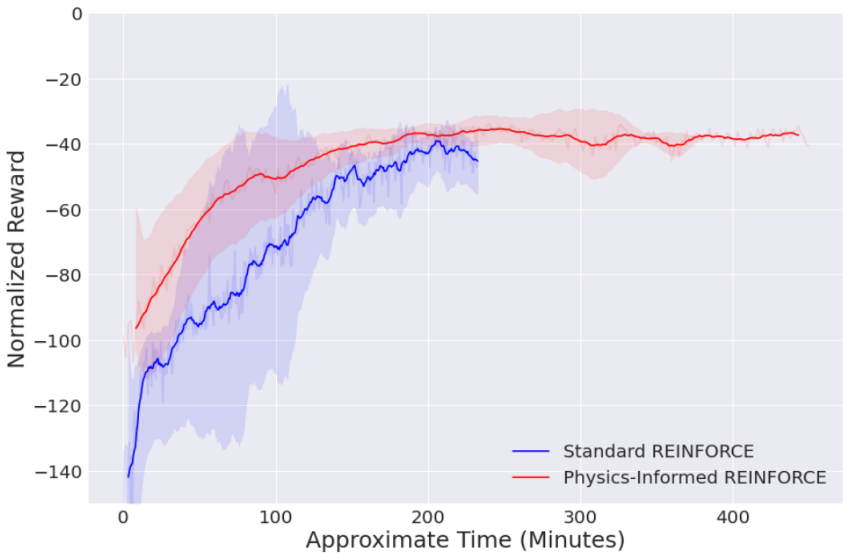

Figure A.2: Learning curve of the standard agent compared to the PINN-enabled agent when training on the discrete action space diffusion environment using REINFORCE plotted against approximate training time.

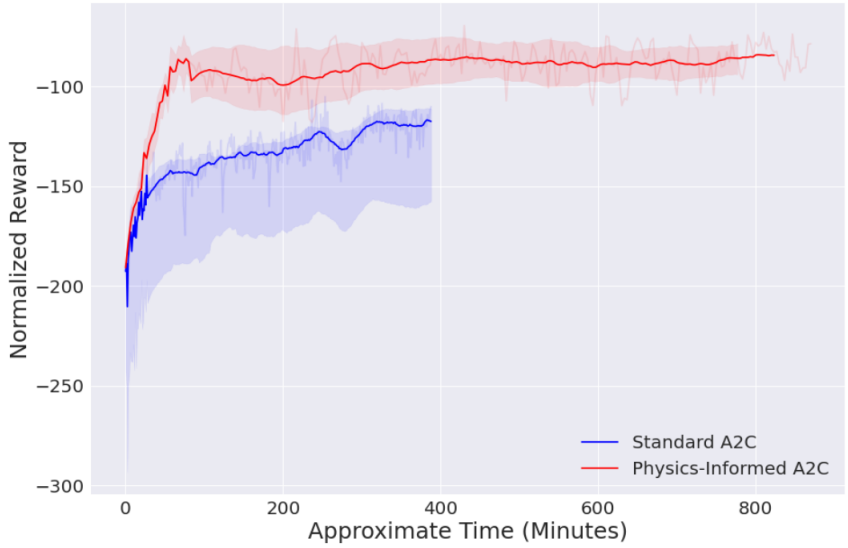

Figure A.3: Learning curve of the standard agent compared to the PINN-enabled agent when training on the discrete action space diffusion environment using A2C measured against training time.

Figure A.4: Physics-informed A2C compared to best standard training run for an extended session on the convection environment.

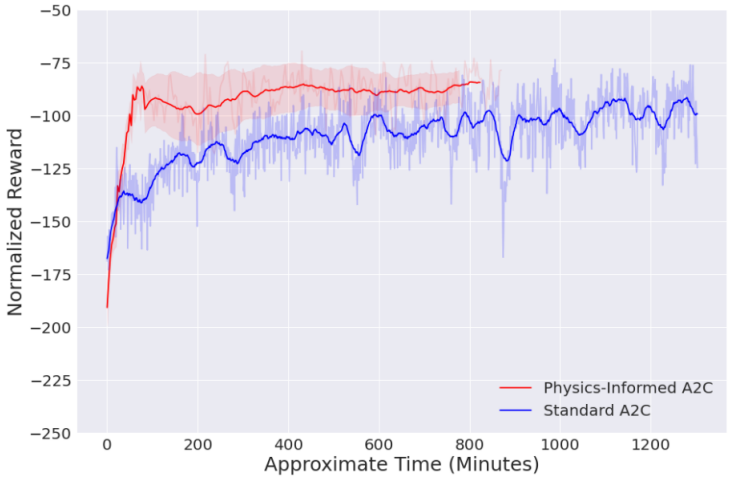

Figure A.5: Physics-informed A2C compared to best standard training run for an extended session on the convection environment plotted against approximate time.

Appendix B

Source Code

The full software packages and corresponding jupyter notebooks can be found at the following GitHub repository:
`https://github.com/itsjacobhere/Phyiscs-informed_Reinforcement_Learning`

This form must be attached to the dissertation as an appendix.

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student Jacob Turner

Academic Year or Project Title 2020-2021

Supervisor Vinay Namboodiri

Does your project involve people for the collection of data other than you and your supervisor(s)? YES / (NO)

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?* YES / NO

   This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO

   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?* YES / NO

   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO

   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature

of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO

Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Will you be offering any incentive to the participants?* YES / NO

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Will you be in a position of authority or influence over any of your participants?* YES / NO

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Will any of your participants be under the age of 16?* YES / NO

Parental consent is required for participants under the age of 16.

10. *Will any of your participants have an impairment that will limit Their understanding or communication?* YES / NO

Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO

All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Will you have a data management plan for all recorded data?* YES / NO

Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).