# NEURAL NETWORK APPROACHES FOR PARAMETERIZED OPTIMAL CONTROL

DEEPANSHU VERMA*, NICK WINOVICH†, LARS RUTHOTTO‡,
AND BART VAN BLOEMEN WAANDERS†

Abstract. We consider numerical approaches for deterministic, finite-dimensional optimal control problems whose dynamics depend on unknown or uncertain parameters. We seek to amortize the solution over a set of relevant parameters in an offline stage to enable rapid decision-making and be able to react to changes in the parameter in the online stage. To tackle the curse of dimensionality arising when the state and/or parameter are high-dimensional, we represent the policy using neural networks. We compare two training paradigms: First, our model-based approach leverages the dynamics and definition of the objective function to learn the value function of the parameterized optimal control problem and obtain the policy using a feedback form. Second, we use actor-critic reinforcement learning to approximate the policy in a data-driven way. Using an example involving a two-dimensional convection-diffusion equation, which features high-dimensional state and parameter spaces, we investigate the accuracy and efficiency of both training paradigms. While both paradigms lead to a reasonable approximation of the policy, the model-based approach is more accurate and considerably reduces the number of PDE solves.

## 1. Introduction

We are interested in deterministic, finite-time optimal control problems in which the dynamical system depends on parameters that are unknown or uncertain. Such problems arise in decision-making for complex systems and are prevalent across application domains, including physics, engineering, finance, robotics, economics, environmental sciences, and traffic flow [9, 45, 38, 5, 43, 48].

We aim to amortize the solution across multiple parameter uncertainties and obtain a policy that maps the current state and parameter to approximate optimal actions. While this introduces significant offline costs, the policy is fast to evaluate online which enables rapid decision-making.

Since an effective policy must depend on the current state of the system and the value of the parameter, amortization is subject to the curse of dimensionality (CoD). Traditional solution approaches become ineffective when the combined dimensions of the state and parameter exceed approximately four. Tackling the CoD leads to two critical challenges: First, one needs to determine policies using a high-dimensional function approximator. Motivated by their universal approximation properties, we consider neural networks. Second, identifying the relevant parts of the state space is critical to avoid the infeasibility of globally sampling high-dimensional spaces. Therefore, it is common to limit the search to parts of the state space likely to be visited when following optimal policies [28, 44].

2020 *Mathematics Subject Classification.* 35F21, 49M41, 68T07 .

*Key words and phrases.* Hamilton-Jacobi-Bellman equation, Reinforcement Learning, high dimensional optimal control, PDE constrained optimization, neural networks, policy optimization, actor-critic.

Given a class of function approximators, we focus primarily on numerical methods for learning effective policies and compare a model-based and a data-driven approach. Our model-based approach approximates the value function of the control problem from which the policy is obtained using a feedback form. The feedback form requires a model of the dynamical system and knowledge of the objective function. This enables incorporating the physics of the system and including known properties of the value function via the Hamilton-Jacobi-Bellman (HJB) equations into the training problem. Our data-driven approach uses the actor-critic reinforcement learning framework to decouple the networks for the policy (i.e., the actor) and the value function (i.e., the critic) and trains both networks purely from observations. Both approaches are intimately related through the underlying control theory, especially Pontryagin's maximum principle theory and HJB equations.

To compare the accuracy and efficiency of both approaches, we implement a model problem motivated by control of contaminant transport. Here, the state is governed by the two-dimensional advection-diffusion partial differential equation (PDE). The objective is to prevent a contaminant originating at a source location from reaching a target region by controlling the location of a sink. Uncertain parameters, in this case, consist of different scenarios for the source location and the velocity field. Since the state is infinite-dimesional, the problem is well-suited to create high-dimensional problem instances through discretization.

Our main contributions and observations are summarized as follows:

- We extend the optimal control approaches from [37] to address parametric control problems by amortizing the control policy over the parameters.
- We provide a direct comparison of the OC and RL approaches, including respective sub-optimalities. For the parameterized OC approach, this experiment increases the problem dimension from around 100 in [37] to about 1000 (see Section 6).
- We observe in our experiments that the OC approach achieves more accurate policies with fewer dynamical system simulations (20x less).

The remainder of the paper is organized as follows: Section 2 introduces the parametric OC problem and provides a brief background on OC theory that informs our approaches. Section 3 discusses related approaches for learning amortized policies. Section 4 presents our neural network approach for computing the value function. Section 5 revisits key RL concepts, laying the groundwork for employing Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradients (TD3) algorithms in policy approximation. Section 6 conducts a comprehensive numerical assessment of both approaches for PDE control problems. Finally, Section 6.4 provides a discussion of the results, highlighting the strengths and limitations of the proposed methodologies.

## 2. Parameterized Optimal Control Problem

We consider a family of deterministic, finite horizon optimal control problems of the form

$$\min_{\boldsymbol{u}\in U} J(t,\boldsymbol{x},\boldsymbol{u};\boldsymbol{y}) := \int_t^T L(s,\boldsymbol{z}(s),\boldsymbol{u}(s))\mathrm{d}s + G(\boldsymbol{z}(T)), \tag{2.1a}$$

subject to a nonlinear dynamical constraint

$$\mathrm{d}_s\boldsymbol{z}(s) = f(s,\boldsymbol{z}(s);\boldsymbol{y}) + g(s,\boldsymbol{z}(s);\boldsymbol{y})\,\boldsymbol{u}(s), \quad s\in(t,T]; \quad \boldsymbol{z}(t)=\boldsymbol{x}. \tag{2.1b}$$

These problems are parameterized by the vector $\boldsymbol{y}$ which is sampled from a distribution $\eta$ on the parameter space. For a given $\boldsymbol{y}$, the functions $f : [0,T]\times\mathbb{R}^d \to \mathbb{R}^d$ and $g : [0,T]\times\mathbb{R}^d \to \mathbb{R}^{d\times a}$ model the evolution of the states $\boldsymbol{z} : [0,T] \to \mathbb{R}^d$ in response to the

controls $\boldsymbol{u} : [0, T] \to U \subset \mathbb{R}^a$. The initial state of the system is denoted as $\boldsymbol{x} \in \mathbb{R}^d$, and the initial time is represented by $t \in [0, T]$. The function $L : [0, T] \times \mathbb{R}^d \times \mathbb{R}^a \to \mathbb{R}$ represents the running cost and $G : \mathbb{R}^d \to \mathbb{R}$ represents the terminal cost in the control objective eq. (2.1a). We assume that $f$, $L$, $G$, $U$ are sufficiently regular (see [50, Chapter 2] and [15, Sec. I.3, I.8-9] for a list of assumptions).

We aim to determine an optimal control policy amortized over the parameters $\boldsymbol{y} \sim \eta$ that minimizes the overall cost. In particular, given a specific parameter $\boldsymbol{y}$, we seek to find the value function

$$\Phi(t, \boldsymbol{x}; \boldsymbol{y}) = \inf_{\boldsymbol{u} \in U} J(t, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y}) \quad \text{s.t.} \quad \text{eq. (2.1b)}, \tag{2.2}$$

which represents the minimum cost-to-go for any initial state $(t, \boldsymbol{x})$. The control $\boldsymbol{u}^*$ for this minimum value is referred to as optimal control, and the corresponding trajectory $\boldsymbol{z}^*$ is called an optimal trajectory.

The problem eq. (2.1) is typically tackled using local solution methods. These methodologies involve minimizing the cost functional by eliminating the constraints while considering a fixed parameter $\boldsymbol{y}$. However, local solutions must be recomputed each time $\boldsymbol{y}$ changes, rendering the previously computed solution obsolete. This motivates devising a global approach where an amortized control policy approximates the optimal control for all parameters $\boldsymbol{y} \sim \eta$.

Subsequently, we will see that the value function $\Phi$ contains complete information about the optimal control, and we can straightforwardly deduce $\boldsymbol{u}^*$ from $\Phi$. Notably, this connection is not explicitly utilized by RL algorithms during policy approximation. This distinction between the HJB approach and RL is significant, as leveraging the underlying physics of the problem can lead to improved results as presented in Section 6.

We now introduce the Hamiltonian, $H : [0, T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$ of the system eq. (2.1), which plays a pivotal role in both OC theory and our numerical approach:

$$H(s, \boldsymbol{z}, \boldsymbol{p}; \boldsymbol{y}) = \sup_{\boldsymbol{u} \in U} \mathcal{H}(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{u}; \boldsymbol{y}). \tag{2.3}$$

Here $\boldsymbol{p} : [t_0, T] \to \mathbb{R}^d$ is referred to as the adjoint or costates of the system and

$$\mathcal{H}(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{u}; \boldsymbol{y}) = \boldsymbol{p} \cdot (f(s, \boldsymbol{z}; \boldsymbol{y}) + g(s, \boldsymbol{z}; \boldsymbol{y})\, \boldsymbol{u}) - L(s, \boldsymbol{z}, \boldsymbol{u}).$$

The Pontryagin maximum principle (PMP) provides a set of first-order necessary conditions and states that, see [15, Theorem I.6.3], the adjoint, $\boldsymbol{p}$, satisfies the adjoint equation. Furthermore, when the value function $\Phi$ is differentiable, the adjoint $\boldsymbol{p}$ and the optimal control $\boldsymbol{u}^*$ can be derived from $\Phi$ using [15, Theorem I.6.2]: for $s \in [t, T]$

$$\boldsymbol{p}(s) = \nabla_{\boldsymbol{z}} \Phi\big(s, \boldsymbol{z}^*(s); \boldsymbol{y}\big), \quad \text{and} \tag{2.4}$$

$$\boldsymbol{u}^*(s, \boldsymbol{z}^*(s), \nabla_{\boldsymbol{z}} \Phi\big(s, \boldsymbol{z}^*(s)\big); \boldsymbol{y}) \in \arg\max_{\boldsymbol{u} \in U} \mathcal{H}\big(s, \boldsymbol{z}^*(s), \nabla_{\boldsymbol{z}} \Phi\big(s, \boldsymbol{z}^*(s); \boldsymbol{y}\big), \boldsymbol{u}(s); \boldsymbol{y}\big). \tag{2.5}$$

Equation (2.5) characterizes optimal control in the feedback form and states that it can be readily computed at any space-time pair for a given $\boldsymbol{y}$ when the value function $\Phi$ and $\nabla_{\boldsymbol{z}} \Phi$ are available. For a thorough exploration of the aforementioned relationships, we refer the readers to [15, 28, 37].

The model-based approach exploits this fundamental relationship to approximate a $\Phi$ amortized across diverse parameters $\boldsymbol{y}$. This ensures a readily accessible approximate optimal control for each parameter setting. The data-driven approach, on the other hand, is agnostic to this relationship and hence suffers in terms of accuracy.

## 3. Related Work

This section reviews closely related approaches applying machine learning techniques to optimal control/policy training. Existing approaches can roughly be grouped based on two fundamental characteristics: (i) the type of function approximators used and (ii) whether training is driven primarily by the data itself or by learning a model. In this work, we restrict our attention to approaches that use neural networks (NNs) as function approximators and consider both model-based and data-driven training algorithms.

Model-based approaches can further be subcategorized into supervised and unsupervised learning approaches. The method in [35] adopts a supervised learning approach to learn a closed-form value function from optimal control trajectories. Other supervised methods, such as those in [13], approximate the value function and its gradient using tensor product decomposition with datasets generated via adaptive sampling from the PMP or by solving state-dependent Riccati equations. Similarly, [1] learns a feedback law from datasets generated by state-dependent Riccati equation solvers.

Some unsupervised, model-based approaches, such as the physics-informed neural networks (PINNs) method in [30, 34], integrate the cost functional into the standard PINNs loss for solving OC problems. However, these approaches are local methods and do not solve the HJB equation corresponding to the control problem, making them very different from our approaches.

Other techniques, such as deep-learning-based surrogate models [48, 32] and operator learning methods [46, 20], focus on achieving fast OC solution inference without intensive computations. In [12], an extended PINN approach utilizes Karush–Kuhn–Tucker (KKT) conditions as the loss function. Similarly, the surrogate modeling approach proposed in [31] reduces the computational cost of PDE solves to enable efficient PDE-constrained optimization under uncertainty. The approach employs neural operators to learn the mapping from uncertain parameters to the solution of the underlying PDE along with its derivatives. The significant acceleration of local solution methods in the above works makes surrogate modeling an attractive option to extend the policy-based methods in this work to higher-dimensional and more complex systems.

The existence and convergence of optimal feedback control when the value function is approximated by a neural network or a polynomial is shown in [25]. In this work, the loss function penalizes the violation of the control cost and the adjoint state from the PMP with the value function and its derivative, respectively, rather than directly penalizing the HJB equation. While this analysis is not directly applicable to our approach, it presents an interesting avenue for future research.

A recent development is the adjoint-oriented neural network method proposed in [49] for parametric OC problems. This method learns the states, adjoints, and control of the system through three different neural networks using a direct-adjoint looping method-type loss. While capable of handling parametric problems simultaneously with random sampling instead of spatial domain discretization, it faces challenges in the presence of changes in initial states or disturbances when following the optimal solution. Moreover, training separate networks introduces computational complexity, limiting the applicability of this approach to less complex problems.

Reinforcement learning (RL) has emerged as a powerful data-driven approach and shown success in diverse applications such as robotics, autonomous driving, games, and recommender systems [33, 41, 47, 42, 11, 21, 7]. Its non-intrusive and gradient-free nature make RL suitable for problems lacking well-defined models. However, RL often requires many

samples, which are computationally expensive in our setting. The process of hyperparameter tuning in RL algorithms is laborious and computationally expensive, hindering their applicability for parametric dynamics [36, 18, 22, 53, 2]. To overcome these limitations, there is a growing interest in effectively integrating system knowledge to overcome the constraints of data-driven methods for handling complex dynamics. In [51, 52], the authors investigate solving OC problems using actor-critic and policy gradient-type methods within the realm of RL. While successful for general OC problems, these methods have limited direct applicability to parametric problems. The main challenge arises from the added complexity introduced by parameters, in addition to the already expensive task of evolving the dynamics over time.

Works closely related to our OC approach include [37, 26, 28], which employ NNs to parameterize the value function and penalize the HJB equation satisfied by the value function. What distinguishes our framework from similar approaches is its consideration of higher-dimensional, parametric OC problems, with dimensionality arising from the states and the amortization of control policies across uncertainties. Moreover, amortization allows us to solve different problems at the same time that typically vary by parameters, making it valuable for various scientific applications, such as digital twins [10, 3]. Additionally, amortization enables a direct comparison with RL techniques.

## 4. Model-Based Approach

In this section, we present our model-driven approach, inspired by [37], for approximating the amortized value function of the parameterized OC problem eq. (2.1). The central concept revolves around utilizing an efficient function approximator to model the amortized value function $\Phi$ in eq. (2.2) and subsequently computing the control using the feedback form given by eq. (2.5). The learning adopts an unsupervised approach akin to RL. The loss function consists of the sum of the expected cost over different parameters, $\boldsymbol{y} \sim \eta$, driving the trajectories and penalty terms that enforce the HJB equations along the trajectories and at the final time.

### 4.1. Learning Problem.

Let $\Phi_{\boldsymbol{\theta}}$ denote the approximation of the value function $\Phi$ with parameters $\boldsymbol{\theta}$. Ideally, we aim for a choice of $\boldsymbol{\theta}$ where $\Phi_{\boldsymbol{\theta}}$ matches the value function of the corresponding control problem for every given $\boldsymbol{y}$. However, this problem suffers from the curse of dimensionality, so we resort to an approach that enforces this property in a subset of the space-time domain.

To learn the parameters in an unsupervised way (that is, no apriori data for $\Phi$ and optimal control trajectories), we approximately solve the minimization problem

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{y}\sim\eta}\left\{J(t,\boldsymbol{z},\boldsymbol{u};\boldsymbol{y}) + P_{\mathrm{HJB},\,\boldsymbol{y}}(\boldsymbol{z})\right\} \\ \text{s.t.}\quad &\mathrm{d}_s\boldsymbol{z} = f(s,\boldsymbol{z}(s);\boldsymbol{y}) + g(s,\boldsymbol{z}(s);\boldsymbol{y})\,\boldsymbol{u}(s), \quad s\in(t,T]; \quad \boldsymbol{z}(t)=\boldsymbol{x}. \end{aligned} \tag{4.1}$$

Here, $P_{\mathrm{HJB},\,\boldsymbol{y}}$ penalizes deviations from the HJB PDE satisfied by the value function for each parameter $\boldsymbol{y}$. In the learning process, the optimal control is computed through the feedback form provided in eq. (2.5), hence, incorporating the model information and introducing a dependence on $\boldsymbol{\theta}$ for both the control $\boldsymbol{u}$ and state $\boldsymbol{z}$.

Exploiting the fact that the value function satisfies the HJB PDE (see [15, Theorems I.5.1, I.6.1]), we guide the approximation of $\Phi$ using the penalty term $P_{\text{HJB},\boldsymbol{y}}$, defined as:

$$\begin{aligned} P_{\text{HJB},\boldsymbol{y}}(\boldsymbol{z}) =& \beta_1 \int_t^T |H(s,\boldsymbol{z},\nabla\Phi_{\boldsymbol{\theta}}(s,\boldsymbol{z}(s);\boldsymbol{y});\boldsymbol{y}) - \partial_s\Phi_{\boldsymbol{\theta}}(s,\boldsymbol{z}(s);\boldsymbol{y})|\mathrm{d}s \\ &+ \beta_2|G(\boldsymbol{z}(T)) - \Phi_{\boldsymbol{\theta}}(T,\boldsymbol{z}(T);\boldsymbol{y})| + \beta_3|\nabla_{\boldsymbol{z}}G(\boldsymbol{z}(T)) - \nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}}(T,\boldsymbol{z}(T);\boldsymbol{y})|, \end{aligned} \tag{4.2}$$

where the relative influence of each term is controlled by the components of $\beta = (\beta_1, \beta_2, \beta_3) \in \mathbb{R}^3_+$. The first component on the right-hand side is the cumulative residual of the HJB PDE satisfied by the value function $\Phi$; the other two components are the terminal time condition and the terminal gradient condition, which are both satisfied by the value function as well.

We further make the following assumption for our model-driven approach.

*Assumption* 4.1. There exists a closed form solution to eq. (2.5), which allows us to write the control $\boldsymbol{u}$ as a function of $\nabla_{\boldsymbol{z}}\Phi$, explicitly, i.e.,

$$\boldsymbol{u}^*(s,\boldsymbol{z}^*(s),\nabla_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*(s);\boldsymbol{y}\big);\boldsymbol{y}) = \arg\max_{\boldsymbol{u}\in U} \mathcal{H}\big(s,\boldsymbol{z}^*(s),\nabla_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*(s);\boldsymbol{y}\big),\boldsymbol{u}(s);\boldsymbol{y}\big), \tag{4.3}$$

A closed-form solution for the optimal control exists in a wide variety of OC problems [29, 6, 24]. While we do not explicitly demonstrate it in this work, it is worth noting that this assumption can be relaxed to include implicitly defined functions as long as they can be efficiently obtained. This flexibility allows for the modeling of more general convex running costs and enhances the applicability of our approach to a wider range of problems.

Equations (4.1) and (4.3) provide a framework for obtaining the optimal control and trajectory based on the value function $\Phi$, subject to certain smoothness assumptions, by outlining the necessary conditions for optimality. Once a good approximation of the value function $\Phi$ is computed, this framework can be applied to any initial data and parameters, also allowing for adaptability to perturbations in the system.

4.2. **Function Value Approximation.** In principle, one could employ any high-dimensional function approximator to parameterize the value function. Yet, due to the universal approximation properties inherent in NNs, we opt for using an NN to parameterize the value function. Designing an effective neural network architecture is essential for various learning tasks, and it remains an active area of research. In our approach, we treat this as a modular component, providing flexibility. Our framework can seamlessly integrate with any scalar-valued neural network that accepts inputs in $\mathbb{R}^{1+d+d_{\boldsymbol{y}}}$, with $d_{\boldsymbol{y}}$ denoting the number of parameters and possesses at least one continuous derivative concerning its first $d+1$ inputs, to allow computations of $\nabla\Phi$.

In our experiments, we use a residual neural network given by

$$\Phi_{\boldsymbol{\theta}}(\boldsymbol{h}_0) = \boldsymbol{w}^\top \mathrm{NN}(\boldsymbol{h}_0;\boldsymbol{\theta}_{\mathrm{NN}}), \tag{4.4}$$

with weights $\boldsymbol{\theta}$ containing $\boldsymbol{w} \in \mathbb{R}^m$ and $\boldsymbol{\theta}_{\mathrm{NN}} \in \mathbb{R}^p$ and inputs $\boldsymbol{h}_0 = (t, \boldsymbol{z}(t); \boldsymbol{y}) \in \mathbb{R}^{1+d+d_{\boldsymbol{y}}}$ corresponding to time-space, and $\mathrm{NN}(\boldsymbol{h}_0;\boldsymbol{\theta}_{\mathrm{NN}})\colon \mathbb{R}^{1+d+d_{\boldsymbol{y}}} \to \mathbb{R}^m$ is a residual neural network (ResNet) [17]

$$\begin{aligned} \boldsymbol{h}_1 &= \sigma(\boldsymbol{K}_0\boldsymbol{h}_0 + \boldsymbol{b}_0) \\ \boldsymbol{h}_{i+2} &= \boldsymbol{h}_{i+1} + \sigma(\boldsymbol{K}_{\mathrm{i}+1}\boldsymbol{h}_{\mathrm{i}+1} + \boldsymbol{b}_{\mathrm{i}+1}), \quad 0 \le \mathrm{i} \le \mathrm{M}-2 \\ \mathrm{NN}(\boldsymbol{h}_0;\boldsymbol{\theta}_{\mathrm{NN}}) &= \boldsymbol{h}_M + \sigma(\boldsymbol{K}_{\mathrm{M}}\boldsymbol{h}_{\mathrm{M}} + \boldsymbol{b}_{\mathrm{M}}), \end{aligned} \tag{4.5}$$

with neural network weights $\boldsymbol{\theta}_{\mathrm{NN}}=(\boldsymbol{K}_0,\dots,\boldsymbol{K}_M,\boldsymbol{b}_0,\dots,\boldsymbol{b}_M)$ where $\boldsymbol{b}_i \in \mathbb{R}^m$ $\forall i$, $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(1+d+d_{\boldsymbol{y}})}$, and $\{\boldsymbol{K}_1,\dots,\boldsymbol{K}_M\} \in \mathbb{R}^{m\times m}$ with $M$ being the depth of the network. We

use the element-wise nonlinearity $\sigma(x) = \log(\exp(x) + \exp(-x))$, which is the antiderivative of the hyperbolic tangent, i.e., $\sigma'(x) = \tanh(x)$. For the experiments, we use $m = 64$ nodes per layer and a network depth of $M = 4$.

4.3. **Numerical Implementation.** We tackle the control problem in eq. (4.1) using the discretize-then-optimize approach. We begin by discretizing the constraints and then optimize. To do so, we first sample parameters $\boldsymbol{y} \sim \eta$ and then use a time discretization of $N+1$ equidistant time points $t = s_0, \ldots, s_N = T$ with step size $\Delta s = (T-t)/N$ to eliminate the constraints in eq. (2.1b). The specific discretization details for each experiment vary and are discussed in their respective sections. This yields a state trajectory starting at $\boldsymbol{z}_0 = \boldsymbol{x}$ via

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i + f(s_i, \boldsymbol{z}_{i+1}; \boldsymbol{y})\Delta s + g(s_i, \boldsymbol{z}_{i+1}; \boldsymbol{y})\, \boldsymbol{u}_i\, \Delta s, \quad i = 0, \ldots, N-1, \tag{4.6}$$

where $\boldsymbol{z}_i = \boldsymbol{z}(s_i)$ and $\boldsymbol{u}_i = \boldsymbol{u}^*(s_i, \boldsymbol{z}_i, \nabla_z \Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i; \boldsymbol{y}))$ is the optimal control obtained from the feedback form, that is, from eq. (4.3), computed by equating $\nabla_{\boldsymbol{u}} \mathcal{H} = 0$. Finally, we approximate the objective functional via

$$J(s_k, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y}) = \Delta s \sum_{i=k}^{N} L(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i) + G(\boldsymbol{z}_N), \tag{4.7}$$

and approximate the penalty term $P_{\text{HJB},\boldsymbol{y}}$ in a similar manner at the same time. Approximating $P_{\text{HJB},\boldsymbol{y}}$ requires one network and gradient evaluation per time step, which is not very expensive compared to the PDE solves.

In principle, any stochastic approximation approach can be used to solve the above optimization problem. Here, we use Adam [23] and sample a minibatch of trajectories originating in i.i.d. samples from $\eta$ with an initial learning rate of 0.075. During training, the learning rate gradually diminishes with an exponential decay rate of 0.975 until it stabilizes at 0.0025. For each training iteration, we randomly select a batch of 20 problem parameters from the distribution $\eta$. Following the HJB framework, we evaluate controls for each problem and evolve the systems accordingly. Subsequently, we compute the losses, using them to update the network weights, as illustrated in Algorithm 1.

## 5. Data-Driven Approach

We now consider an alternative, data-driven approach to the problem: reinforcement learning (RL), which provides a general framework for approximating policies through a series of interactions with an environment. More precisely, an RL environment specifies all possible system states, a description of admissible actions, and a model governing the evolution of the system space when an action is performed. In RL language, the controls are referred to as the actions. The environment is paramount for RL algorithms, and its proper definition is critical. In this section, we provide the RL formulation for the learning problem introduced in Section 4.1. We then give a brief overview of the actor-critic models we will use for comparison with the proposed model-based approach. Our presentation follows [8, 44].

Let $\pi_{\boldsymbol{\theta a}}$ denote the policy, parameterized by parameters $\boldsymbol{\theta}_{\boldsymbol{a}}$, which is a mapping from the state space to the action space. The policy can be either deterministic, denoted as $\boldsymbol{u}_i = \pi_{\boldsymbol{\theta a}}(\boldsymbol{z}_i)$, or stochastic, indicated as $\boldsymbol{u}_i \sim \pi_{\boldsymbol{\theta a}}(\cdot|\boldsymbol{z}_t)$. Using eq. (4.6), a policy generates a sequence of states and actions, $(\boldsymbol{z}_i, \boldsymbol{u}_i)_{i=0}^N$, referred to as an episode.

To evaluate the performance of a policy, it is necessary to create functions step() and reset() to govern the problem dynamics and the interaction between the agent and the

**Algorithm 1** Model-based training approach

**Require:** Number of training problems P, problem parameter distribution $\eta$, number of time steps $N$, initial network weights $\boldsymbol{\theta}$, Hamiltonian $H$, and loss weighting factors $(\beta_1, \beta_2, \beta_3)$

**Initial Setup for Training**
*Derive Feedback Form Expression, using eq.* (4.3)
Feedback_Form : $(-\nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}},\, f,\, g) \mapsto \boldsymbol{u}$

*Assemble Randomized Problems*
Sample P parameters $\{\boldsymbol{y}_i\}_{i=1}^{\mathrm{P}} \sim \eta$
Store `discretization` for $f$ and $g$ corresponding to $\{\boldsymbol{y}_i\}_{i=1}^{\mathrm{P}}$

**Training Iteration**
*Sample Batch of Problems*
Retrieve `discretized` $f$ and $g$ corresponding to $\boldsymbol{y}_p$ with $p \sim \mathrm{Uniform}(\{1, \ldots, \mathrm{P}\})$

*Assess HJB Control Performance*
Set $s_0 = 0$ , $\boldsymbol{z}_0 = \boldsymbol{x}$ , $J = 0$, and $P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}} = 0$
**while** $i \leq N$ **do**
    *Approximate Control using Feedback Form*
    Evaluate network $\Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i, \boldsymbol{y}_p)$ and gradient $\nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i; \boldsymbol{y}_p)$
    $\boldsymbol{u}_i = \mathrm{Feedback\_Form}(-\nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i; \boldsymbol{y}_p),\, f,\, g)$

    *Evolve System Dynamics eq.* (2.1b) *in Time*
    $\boldsymbol{z}_{i+1} \leftarrow \mathrm{Time\_Integrator}(\boldsymbol{z}_i, \boldsymbol{u}_i; \boldsymbol{y}_p)$
    $s_{i+1} \leftarrow s_i + \Delta s$

    *Update Intermediate Losses*
    $J \leftarrow J + L(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i)\Delta s$
    $P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}} \leftarrow P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}} + \beta_1 |H(s_i, \boldsymbol{z}_i, \nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i; \boldsymbol{y}_p); \boldsymbol{y}_p) - \partial_s \Phi_{\boldsymbol{\theta}}(s_i, \boldsymbol{z}_i, \boldsymbol{y}_p)|\Delta s$
**end while**
*Update Final Losses*
$J \leftarrow J + G(\boldsymbol{z}_N)$
$P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}} \leftarrow P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}} + \beta_2 |G(\boldsymbol{z}_N - \Phi_{\boldsymbol{\theta}}(s_N, \boldsymbol{z}_N, \boldsymbol{y}_p)| + \beta_3 |\nabla_{\boldsymbol{z}} G(\boldsymbol{z}_N) - \nabla_{\boldsymbol{z}}\Phi_{\boldsymbol{\theta}}(s_N, \boldsymbol{z}_N, \boldsymbol{y}_p)|$

*Update Networks Weights*
$\theta \leftarrow \mathrm{Optimizer}(J + P_{\mathrm{HJB},\boldsymbol{y}_{\mathrm{p}}})$

environment. Aiming to address the optimization problem introduced in Section 2, these functions operate as follows

- *reset()*: This function initializes or resets the environment to its initial state, $\boldsymbol{z}_0$, preparing it for the new episode.
- *step()*: Given an action $\boldsymbol{u}_i$ as input, determined by a policy, it returns the following:
  - The next state $\boldsymbol{z}_{i+1}$ using the dynamics described in eq. (4.6).

– A scalar feedback signal representing the objective function, known as the reward,

$$r_i = \begin{cases} \Delta s\, L(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i), & i < N \\ \Delta s\, L(s_N, \boldsymbol{z}_N, \boldsymbol{u}_N) + G(\boldsymbol{z}_N), & i = N \end{cases}.$$

– The termination status indicates the terminal time $T$ has been reached or the current episode has been terminated.

Specific environment details for the problems considered are provided in Section 6.

The goal is to identify an optimal policy in order to minimize the cumulative reward, called return[1], given by the value function

$$\Psi(\boldsymbol{\theta_a}) \;=\; \mathbb{E}_{\pi_{\boldsymbol{\theta_a}}, \boldsymbol{y}\sim\eta}\left\{\sum_{i=0}^{N} r_i\right\} \;=\; \mathbb{E}_{\pi_{\boldsymbol{\theta_a}}, \boldsymbol{y}\sim\eta}\left\{J(s_0, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y})\right\}. \tag{5.1}$$

When the policy is deterministic, the expectation over $\pi_{\boldsymbol{\theta_a}}$ in eq. (5.1) is omitted, making the return equivalent to the objective function in the model-based approach; see eq. (4.7).

*Remark* 5.1. When starting from the initial state $\boldsymbol{z}_0$ and following the optimal policy $\pi^*_{\boldsymbol{\theta_a}}$, we recover the original value function $\Phi$ in eq. (2.2).

In our work, we consider the policy stochastic as they provide additional flexibility, robustness, and adaptability that can be advantageous in complex and uncertain environments.

Now that we have established a loss function representative of the original optimization problem, we must also review the expression for the gradients with respect to $\boldsymbol{\theta_a}$ for updating these parameters. This expression is provided by the following foundational result from the theory of RL.

**Theorem 5.2.** *[44, Policy Gradient Theorem, Chapter 13] The gradient of the cumulative reward objective function eq. (5.1) can be expressed in terms of the policy gradients as follows:*

$$\nabla_{\boldsymbol{\theta_a}}\Psi(\boldsymbol{\theta_a}) \;=\; \mathbb{E}_{\pi_{\boldsymbol{\theta_u}}, \boldsymbol{y}\sim\eta}\left\{\sum_{i=0}^{N} \nabla_{\boldsymbol{\theta_a}} \log \pi_{\boldsymbol{\theta_a}}(\boldsymbol{u}_i|\boldsymbol{z}_i) J(s_i, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y})\right\}. \tag{5.2}$$

While the direct use of Theorem 5.2 for policy optimization can work effectively in some cases, the practical performance often suffers from high variance in the gradient estimate, and thus produce slow learning. To reduce variance effects and increase stability, many policy-based RL algorithms subtract the cumulative reward by a baseline, $b(\boldsymbol{s})$:

$$\nabla_{\boldsymbol{\theta_a}}\Psi(\boldsymbol{\theta_a}) \;=\; \mathbb{E}_{\pi_{\boldsymbol{\theta_a}}, \boldsymbol{y}\sim\eta}\left\{\sum_{i=0}^{N} \left(J(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i; \boldsymbol{y}) - b(\boldsymbol{z}_i)\right) \nabla_{\boldsymbol{\theta_a}} \log \pi_{\boldsymbol{\theta_a}}(\boldsymbol{u}_i|\boldsymbol{z}_i)\right\}. \tag{5.3}$$

Intuitively, making the cumulative reward smaller by subtracting it with a baseline will make smaller gradients, and thus smaller and more stable updates.

5.1. **Actor-Critic Models.** Selecting an effective baseline poses a challenge, but the state-value function is a suitable choice, making both the value function and policy learnable. This creates a two-component RL framework known as an actor-critic model:

- The actor $\pi_{\boldsymbol{\theta_a}}$ determines actions based on the current state (i.e., defines the policy).
- The critic $\Psi_{\boldsymbol{\theta_c}}$, parameterizing the baseline with parameters $\boldsymbol{\theta_c}$, assesses the approximate value of the current state (i.e., provides value function estimates).

[1]Conventional RL literature assumes that the objective function is being maximized; for the purposes of aligning with the optimal control problem setup from previous sections, we instead aim to minimize the loss (which can be interpreted as the negative reward in standard RL literature).

and both are parameterized by neural networks.

Now that the value function is also made learnable, that means we must provide an objective for the critic network to learn the parameters $\boldsymbol{\theta_c}$. Since the role of the critic is to provide a baseline estimate for the return, a natural choice for this objective is the expected value of the squared error:

$$P_{\text{critic}}(\boldsymbol{\theta_c}) \;=\; \sum_{i=0}^{N} \left|\Psi_{\boldsymbol{\theta_c}}(\boldsymbol{z}_i) \;-\; J(s_i, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y})\right|^2 . \tag{5.4}$$

In summary, actor-critic models aim to approximately solve the following minimization problem, where $\beta_5 \in \mathbb{R}_+$ controls the relative contribution of critic:

$$\min_{\boldsymbol{\theta_a}, \boldsymbol{\theta_c}} \; \mathbb{E}_{\pi_{\boldsymbol{\theta_a}}, \boldsymbol{y}\sim\eta} \left\{ J(s_0, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y}) + \beta_5 \, P_{\text{critic}}(\boldsymbol{\theta_c}) \right\} .$$

There are several actor-critic algorithms available, and in this work, we focus on two prominent, state-of-the-art algorithms: Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradients (TD3). The PPO and TD3 algorithms have been applied to a range of practical applications and consistently provides performance competitive with other state-of-the-art methods [27, 4, 40]. PPO and TD3 are both advanced versions of the actor-critic framework that address critical aspects regarding stability, mitigating overestimation bias, and improving exploration. PPO uses the concept of trust regions to control the optimization step-size and prevent the agent from drifting too far away from the current policy. TD3 uses delayed updates to stabilize training. This is done by maintaining "twin" copies of the actor and critic models that are updated less frequently than the primary models. These delayed updates provide a layer of robustness to transient, unanticipated events that have a tendency to destabilize RL training procedures. OpenAI's 'Spinning Up' documentation provides brief technical overviews of PPO[2] and TD3[3]. For more in-depth explanations and implementation specifics, we direct readers to [39, 16].

5.2. **RL Network Architecture.** Since the observation data for our experiments is spatially structured on a two-dimensional grid, we employ convolutional network architectures to process the system state at each time step. Both the actor and critic networks share the same network architecture with the exception of the final layer which is adapted to the specific format of the actions and value outputs, respectively. As illustrated in Figure 1, the networks consist of three convolutional layers with $3 \times 3$ kernels each followed by $2 \times 2$ max-pooling layers; the series of convolutional layers is then followed by two dense layers, and a final linear layer is used to produce the position/variance in the case of the actor network and the value prediction in the case of the critic network. All layers are equipped with hyperbolic tangent activation functions, and orthogonal initialization is used for both the convolutional and dense weights.

We train the networks using a pair of Adam optimizers, one used for the actor network and the other used for the critic and apply an exponential decay to the learning rates, which are both initially set to $10^{-4}$. The observation states are also normalized using an exponential moving average estimate for the mean and variance of the input data in each batch.

It is essential to highlight the difference in network architectures employed for the OC and RL approaches. The OC approach necessitates an architecture with at least one continuous derivative with respect to the network inputs. In contrast, the RL approach only requires

[2] https://spinningup.openai.com/en/latest/algorithms/ppo.html
[3] https://spinningup.openai.com/en/latest/algorithms/td3.html

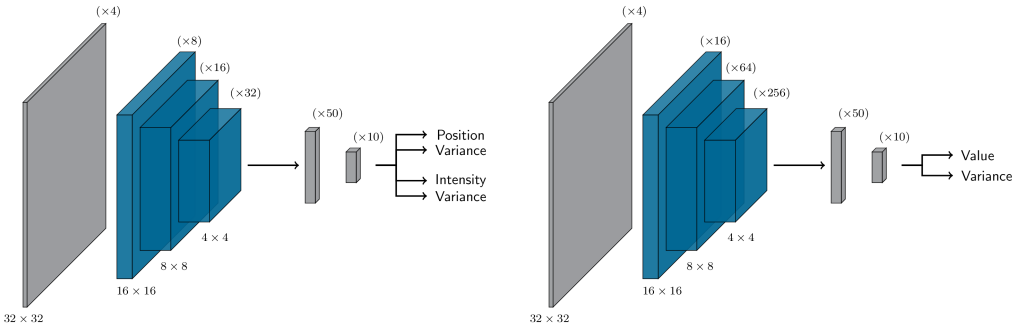


FIGURE 1. Network architectures for the actor (left) and critic (right) components of the RL models. Both networks receive input arrays with input containing the values of $\boldsymbol{z}(s)$, $\boldsymbol{y}$, and $s$. Convolutional and max-pooling layers (blue) process the data received from the PDE environment to extract features. These features are then flattened and passed to dense layers (grey) to form the position, variance, and value predictions.

policies that are differentiable with respect to the network weights. Because of this, RL is compatible with most off-the-shelf architectures, while architecture choices for the OC approach are slightly more restricted.

## 6. Numerical Results

In this section we compare the model-based and data-driven training approaches for a control problem involving the advection diffusion equation. Since the ground-truth solutions for these problems are not available, we use conventional, gradient-based solutions as baselines to approximate the optimal solutions for fixed parameters in a test set. All models were trained using 10 cores of an Intel Xeon Platinum 8176 CPU at 2.10GHz.

In our initial experiments, we trained all models using the same network architecture; however, we found the OC and RL approaches required distinct design choices to attain the best results. When using a convolutional architecture for the OC training procedure, the gradient signals appeared to be too weak, and models failed to train from the start. For the RL models, replacing convolutional layers with a residual network led to a noticeable drop in performance that remained after several iterations of hyperparameter tuning. To provide a fair assessment, we present numerical results using the architecture that yields the best performance for each approach.

### 6.1. Advection-Diffusion Problem Formulation.

For our numerical tests, we consider the solution $a(s, x_1, x_2)$ to an advection-diffusion PDE defined on a square domain $\Omega = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ at time $s$ between $t = 0$ and $T = 0.75$. The control problem is based on a continuous control vector $\boldsymbol{u}(s)$, and the goal is to prevent positive values of the solution $a$ from occurring in a target region $\Omega_{\text{targ}} = \{(x_1, x_2) \in \Omega : x_1 > 0.75\}$ at the final time. A motivating contaminant control example is illustrated in Figure 2 where $a$ represents the contaminant concentration (in red) that originates from a source location determined by the vector $\boldsymbol{y}$. For our first experiment, the parameters $\boldsymbol{y}$ determine the source location while the velocity field remains fixed; we then extend the parameters to include information specifying the phase of the advecting velocity in addition to the source location. The goal

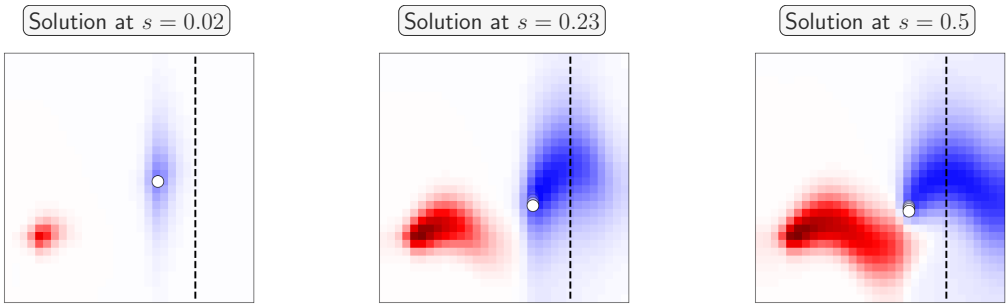


FIGURE 2. Example evolution of advection-diffusion system.

is to prevent contamination from reaching the target subdomain $\Omega_{\text{targ}}$, which is shown to the right of the vertical dashed line. To achieve this, an optimal sequence of locations and magnitudes for the sink (in blue) must be found, both of which will depend strongly on the parameter $\boldsymbol{y}$ under consideration.

For a given $\boldsymbol{y} \sim \eta$, the objective function for the problem is given explicitly by:

$$\frac{1}{2}\int_0^T |\boldsymbol{u}(s)|^2 \mathrm{d}s + \rho \int_{\Omega_{\text{targ}}} |\max\{a(T, x_1, x_2), 0\}| \, \mathrm{d}x_1 \, \mathrm{d}x_2. \tag{6.1}$$

The primary dynamics from eq. (2.1b) are prescribed by the advection-diffusion PDE

$$\partial_s a + \operatorname{div}(\kappa \nabla a) + \varepsilon_{\boldsymbol{y}} \nabla a + \varphi_{\boldsymbol{y}} \;=\; u_1 \cdot Q(\alpha) \tag{6.2}$$

with initial condition $a(0, x_1, x_2) = 0$ and homogeneous Neumann boundary conditions. In addition to this PDE, the system dynamics include an equation for tracking the sink's position, $\alpha$, based on the selected controls.

The controls $\boldsymbol{u}(s) = [u_1(s), u_2(s)]^\top$ influence system dynamics through the sink term '$Q$' on the right-hand-side of eq. (6.2). The value $u_1$ determines the magnitude of the sink, and $u_2$ controls the sink's velocity. The velocity is used to update the sink position $\alpha$ as the system evolves; this update is prescribed by the following equation, and it is included as the final component in the system dynamics:

$$\mathrm{d}_s \alpha(s) \;= u_2(s), \quad 0 \le s < T, \quad \text{with} \quad \alpha(0) = 0.5.$$

Once $\alpha$ is computed based on the values of $u_2$, we define the sink term $Q$ as follows:

$$Q(x_1, x_2, \alpha(s)) \;=\; 25 \cdot \exp\left(-|x_1 - 0.6|/0.025 - |x_2 - \alpha(s)|/0.15\right).$$

The problems are randomized by sampling parameters $\boldsymbol{y} \sim \eta$ that affect the initial state and dynamics of the system. Each parameter consists of three components $\boldsymbol{y} = [\boldsymbol{y}_{x_1}, \boldsymbol{y}_{x_2}, \boldsymbol{y}_v]$. The parameters $\boldsymbol{y}_{x_1}$ and $\boldsymbol{y}_{x_2}$ denote the $x_1$ and $x_2$ coordinates of the source term $\varphi_{\boldsymbol{y}}$. The source term is then defined by:

$$\varphi_{\boldsymbol{y}}(x_1, x_2) \;=\; (c/\sigma_s) \exp\left(-(|x_1 - \boldsymbol{y}_{x_1}| + |x_2 - \boldsymbol{y}_{x_2}|)/\sigma_s\right) \tag{6.3}$$

The remaining parameter, $\boldsymbol{y}_v$, adjusts the phase of the velocity field, $\varepsilon_{\boldsymbol{y}}$, as described below.

For our first family of problems, we consider a fixed velocity field with randomized source locations. We refer to this as the *horizontal setup* and use the following parameter values:

$$\varepsilon_{\boldsymbol{y}}(x_1, x_2) = [25, 0]^\top, \quad c = 5, \quad \sigma_s = 0.01, \quad \text{and} \quad \kappa = 0.008.$$

We then consider a new family of problems, the *sinusoidal setup*, which incorporates a more complex velocity field and extends problem randomization to include an additional parameter, $\boldsymbol{y}_v$, that determines the phase of the advecting velocity field:

$$\varepsilon_{\boldsymbol{y}}(x_1, x_2) \;=\; \begin{bmatrix} (1+x_1)\cdot\sqrt{0.9^2 - (0.75\cdot\cos(1.1-x_1)\cdot\sin(4\pi(x_1-\boldsymbol{y}_v)))^2} \\ -0.9\cdot\cos(1.1-x_1)\cdot\sin(4\pi(x_1-\boldsymbol{y}_v)) \end{bmatrix} \tag{6.4}$$

For the sinusoidal setup, we also adjust the source magnitude to $c = 0.5$ and change the source size to $\sigma_s = 0.025$ to produce more natural flows for the modified velocity field.

We sample the components of $\boldsymbol{y}$ from the uniform distributions

$$\boldsymbol{y}_{x_1} \sim \mathrm{Unif}(0.1, 0.25) \quad , \quad \boldsymbol{y}_{x_2} \sim \mathrm{Unif}(0.2, 0.8) \quad , \quad \boldsymbol{y}_v \sim \mathrm{Unif}(-0.425, 0.0)$$

so that the source locations are varied in the horizontal and vertical directions across the left side of the domain. For the sinusoidal setup, the velocity field directs concentration near the source upward for values of $\boldsymbol{y}_v \approx -0.425$ and directs downward for $\boldsymbol{y}_v \approx 0.0$.

Finally, to balance the importance of the running cost and terminal cost terms in the objective function, we choose a scaling factor of $\rho = 40$ in eq. (6.1). This is generally necessary since the scale of the action space does not always match that of the state space, and one of the loss components can dominate the other if left unweighted.

6.1.1. *Discretization.* Since the continuous formulation results in an infinite-dimensional OC problem, we cannot apply numerical methods to the problem described above directly. Instead we consider an approximate formulation arising from the spatial discretization of eq. (6.2) using finite elements.

Denoting the discretized, FEM representation of $a(s, x_1, x_2)$ by $\boldsymbol{a}(s)$, the state vector for the problem is given by:

$$\boldsymbol{z}(s) \;=\; [\boldsymbol{a}(s)\,,\, \alpha(s)]^\top .$$

The dynamics for the $\boldsymbol{a}$ component of the state vector are defined by the linear systems associated with the FEM formulation of eq. (6.2). By construction, the state dynamics for $\alpha$ are simply $\mathrm{d}_s\alpha(s) = u_2(s)$.

For the spatial discretization, we have selected linear, triangular elements with nodes associated with a uniform $32 \times 32$ grid. The FEM representation for the concentration $\boldsymbol{a}$ is passed to the neural networks by evaluating pointwise on the grid locations $\{(x_1^i, x_2^j)\}_{i,j=1}^{32}$ with $x_1^i = i \cdot \Delta x_1$, $x_2^j = j \cdot \Delta x_2$, and $\Delta x_1 = \Delta x_2 = 1/31$. We further discretize the PDE in time via implicit Euler with a step count of $N = 25$ and step size $\Delta s = 0.02$ to obtain the fully discretized control objective

$$J(0, \boldsymbol{z}, \boldsymbol{u}; \boldsymbol{y}) := \frac{1}{2}\sum_{i=0}^{25} |\boldsymbol{u}_i|^2 \Delta s \;+\; \rho \sum_{i,j=1}^{32} 1_{\Omega_{\mathrm{targ}}}(x_1^i, x_2^j)\, |\max\{\boldsymbol{a}(T), 0\}|\, \Delta x_1 \Delta x_2. \tag{6.5}$$

6.1.2. *Feedback Form for HJB.* To apply the proposed model-based procedure, we first need to derive the feedback form for the optimal control. This expression depends on the specific PDE under consideration and provides a natural way of incorporating system knowledge into the training procedure. As stated in eq. (2.5), the optimal control is a maximizer for the Hamiltonian evaluated at $\nabla_{\boldsymbol{z}}\Phi$. By taking the gradient of the Hamiltonian and setting the resulting expression to zero, the feedback form for both problem setups is given by:

$$\boldsymbol{u}(s) = \left[-\boldsymbol{Q}^T \nabla_{\boldsymbol{a}}\Phi(s),\, -\nabla_\alpha \Phi(s, \boldsymbol{z}(s); \boldsymbol{y})\right]^\top ,$$

where $\boldsymbol{Q}$ is the finite element matrix associated with the sink term '$Q$' in eq. (6.2).

6.1.3. *Parallel Implementation of RL environments.* The FEM calculations used to simulate the PDE environment present a significant computational bottleneck in the training procedure. Fortunately, both the PPO and TD3 algorithms can be trained using interactions with multiple, distinct environments in parallel. This is possible since both algorithms decouple the actor weights used during optimization from the network weights used in the simulations[4]. In particular, since both algorithms update policy weights based on interactions performed by decoupled policies, it is possible to run multiple episodes in parallel before calculating losses and applying gradient updates. To take advantage of this, we employ OpenMPI and MPI for Python to parallelize the interactions of the actor network with several different environment realizations simultaneously. The environments are initialized in parallel, observation data is gathered to the root node and passed to the actor/critic networks as a single batch, and the proposed actions are then broadcast back to the respective environments. We repeat this process until all environments have completed a full episode. After all episodes are complete, we compute the associated returns, group transitions from all environments into minibatches, and perform gradient updates.

6.2. **Hyperparameter Selection.** Determining effective hyperparameters, such as optimization parameters and network architectures, is critical when training neural networks. In our experience, the selection of hyperparameters often depends on the specific problem at hand as well as the learning algorithm being used. Developing well-founded mathematical principles for parameter selection in these models represents a significant area for future research, but it exceeds the scope of this paper. Here, we provide our insights into hyperparameter selection for the two approaches considered in this paper.

The proposed HJB approach requires tuning only a small number of parameters: the networks' width and depth, the learning rate and decay schedule, the batch size, and three HJB penalty weights $(\beta_1, \beta_2, \beta_3)$. Since the $\beta$ parameters determine the weight of the HJB constraints in the loss function, these choices can result in varying outcomes.

- $\beta_1$ weights the HJB residual; if set too high, the model underprioritizes reaching the target, leading to suboptimal solutions.
- $\beta_2$ weights the final condition on $\Phi$; if set too high, the model overprioritizes fitting $\Phi$ at time $T$ rather than $\nabla_{\boldsymbol{z}}\Phi$, which is more relevant to the dynamics.
- $\beta_3$ weights the gradient term; if set too high, the model overprioritizes reaching the target with less flexibility to choosing trajectories for optimizing $L$. If $\beta_3$ is too low, the model may fail to adequately reach the target, increasing suboptimality.

The RL models considered in this work have a much larger set of hyperparameters and require a substantial number of system solves to identify the correct training settings. In addition to the parameters explicitly described in the original papers, there are a number of subtle implementation details required for the data-driven RL procedures to work in practice [14, 19]. In particular, we found that a series of scaling factors needed to be introduced to adjust the magnitude of the rewards and actions for the RL agents. Gradient clipping was also critical to the performance of the RL models, and an extensive series of experiments was performed to identify suitable clipping and scaling levels. When these parameters are not set correctly, the models' performance will often deteriorate midway through training, and the procedure must be restarted.

For the experiments considered in this work, we found the hyperparameter tuning process for the RL models required significantly more time than the HJB approach. This was in

[4]PPO decouples weights through an '*old*' policy, and TD3 uses a related technique using a '*target*' policy.

part due to the larger number of parameters for the RL algorithms (and the need to implement new clipping and scaling operations after tuning alone was deemed insufficient). The effects of the RL parameters were also less transparent, and interactions between different parameters further complicated the tuning process. For example, the appropriate level of gradient clipping was strongly influenced by the scaling operations, but it also needed to be balanced with the learning rates for the actor and critic. Finding an effective combination of settings for these parameters was non-intuitive and required dozens of training attempts. For the HJB approach, each parameter is directly associated with a specific component of the loss. The values of each loss term can easily be monitored during training, and we adjusted the parameters to ensure no term is under-represented or over-represented in the overall loss. This results in a straight-forward procedure for tuning HJB hyperparameters, which is noticeably faster than the RL tuning procedure and requires significantly fewer PDE solves.

6.3. **Baseline via Gradient-Based Solutions.** To understand how the proposed amortized procedure compares with more conventional approaches, we now review how gradient-based methods handle this class of control problems. These methods apply to a single problem realization (i.e., a fixed value of $\boldsymbol{y}$) and rely on gradient information, namely $\nabla_{\boldsymbol{u}} J$, to identify an optimal control. For this work, we have implemented a gradient-based solver using FEM matrices extracted from FEniCS and automatic differentiation provided by PyTorch to retrieve the necessary gradients. We then use the L-BFGS algorithm to search the control space for an optimal solution. We use the solutions obtained from this gradient-based search as proxies for the true optimal solutions (which are intractable for the problems considered in this work).

Here, it is important to note that the control obtained from the gradient-based method is specific to the given parameter. While the control may perform well for nearby parameters, in general, changes in the parameter require another solve. In many cases, the original control does not even provide an effective initialization point for new problems (since the optimal control can change considerably when system dynamics are modified). Because of this, the initialization procedure is another important and non-trivial aspect of gradient-based methods. Even though these methods are designed to solve a single problem, unlike amortized approaches that target a number of problems at once, the gradient-based search can fail to converge to an optimal control without proper initialization. In order to evaluate the nonlinear impact of uncertain parameters, we conducted a linear regression analysis between the parameters $\boldsymbol{y}$ and the corresponding optimal controls obtained using L-BFGS. The resulting average $R^2$ (coefficient of determination) score of 0.6 suggests a significant nonlinear influence.

6.4. **Assessment of HJB and RL Performance.** For the first set of experiments, we consider the horizontal velocity setup. The RL and HJB models are tasked with positioning the sink in locations that block the concentration from entering the target region while using as little movement as possible. To evaluate each model's performance, we assess the accuracy of the models on a set of fixed validation problems throughout training. For the validation set, we have selected parameter values $\boldsymbol{y}_{x_1} \in \{0.125, 0.225\}$ and $\boldsymbol{y}_{x_2} \in \{0.25, 0.4, 0.5, 0.6, 0.75\}$. This corresponds to 10 realizations of the horizontal setup with source locations spread across the left side of the domain. For HJB training, we used $P = 200$ example problems with values of $\boldsymbol{y}_{x_1}$ and $\boldsymbol{y}_{x_2}$ randomly sampled from the intervals $[0.1, 0.25]$ and $[0.2, 0.8]$, respectively. We compare the performance of the RL and HJB models with solutions obtained using the gradient-based method from Section 6.3 as a baseline

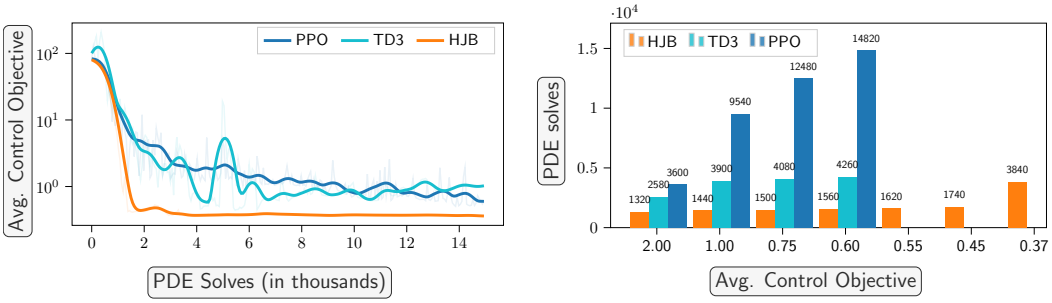


FIGURE 3. Horizontal problem setup: (left) Validation loss during training and (right) number of PDE solves required for different target accuracies of control objective.

for near-optimal performance; the resulting suboptimality levels for both experiments are reported in Figure 5.

The results for the horizontal setup are summarized in Figure 3 where we observe that the HJB framework yields several, noticeable improvements over both RL models. The most clear, and practically significant, distinction between the HJB and RL training procedures is the rapid drop in the HJB validation loss at the very start of training. This is perhaps to be expected thanks to the gradient information exposed by the HJB training formulation, while the RL models are initially reliant on an uninformed, trial-and-error state of exploration. For practical applications, this is a key feature of the HJB training procedure; the model is able to identify effective control policies using far fewer forward-model PDE solves than its RL counterparts. By relying on fewer forward solves, the HJB approach has the potential to be applied to more computationally expensive simulations where performing the tens of thousands of queries required by RL models is simply impractical.

To assess how well the model-based approach framework extends to more complicated systems, we now turn to evaluating its performance on the sinusoidal setup. This setup introduces more complex system dynamics which incorporate non-uniform velocity fields for advection. We also add another layer of randomization by varying the phase of the velocity field according to an additional problem parameter $\boldsymbol{y}_v$. This variability in the velocity field is designed to reflect uncertain ambient conditions (such as weather) which may need to be taken into account when developing control strategies for practical applications. This form of randomization influences how systems evolve in time and will help us gauge how well the model-based and RL approaches are able to amortize over problems with substantially different dynamics.

For the sinusoidal setup, we define the validation set using parameter values selected from $\boldsymbol{y}_{x_1} \in \{0.125, 0.225\}$, $\boldsymbol{y}_{x_2} \in \{0.25, 0.4, 0.5, 0.6, 0.75\}$, and $\boldsymbol{y}_v \in \{-0.35, -0.2125, -0.1\}$. This yields 30 problem realizations with source locations spread across the left side of the domain and velocity fields which direct concentration near these source terms upward, horizontally, as well as downward. For HJB training, we increased the number of example problems to $P = 400$, and sampled values of $\boldsymbol{y}_v$ randomly from the interval $[-0.425, 0.0]$.

The results for the sinusoidal setup are depicted in Figure 4, where we again observe a clear advantage in data efficiency for the HJB model. The HJB model achieves an average control objective of 0.15 using only 780 PDE solves. However, both RL models fail to achieve an average control objective below 0.25 even with the full 15,000 solves.

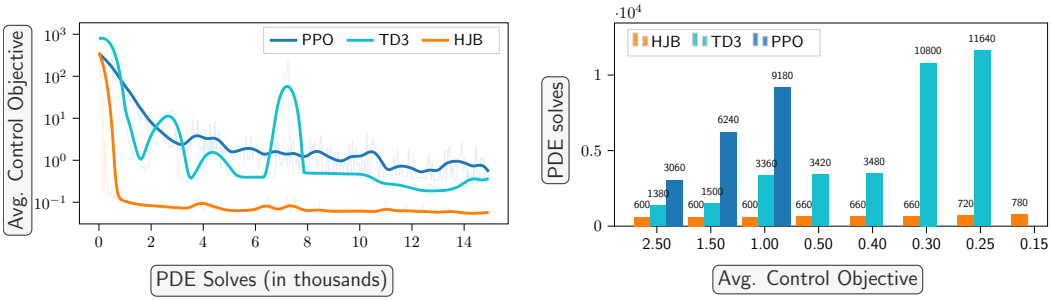


FIGURE 4. Sinusoidal problem setup: (left) Validation loss during training and (right) number of PDE solves required for different target accuracies of control objective.

In Figure 5, we also present the suboptimality of HJB and RL approaches compared to the baseline, as discussed in Section 6.3. We observe that the HJB model achieves much lower suboptimality than the RL models with significantly fewer PDE solves. This difference is particularly pronounced in the sinusoidal setup, where the HJB approach can achieve a suboptimality of 0.03 after 4740 PDE solves, while both RL approaches fail to reach a suboptimality below 0.20.

This faster convergence can be partially attributed to the training algorithms used in RL. While both RL models succeed in preventing positive concentrations from reaching the target region, they do so with unnecessary movement. The RL policies often exhibit bias in initial movement in one direction (e.g., drifting downward at the start of a simulation) and then correct with a sharp reversal in direction if needed. These inefficiencies result in much larger running costs for the RL models and prevent them from reaching the control objective levels attained by the HJB approach.

## 7. DISCUSSION

We consider deterministic, finite-time optimal control problems with unknown or uncertain parameters and aim to approximate optimal policies for all parameters with neural networks. We compare two training paradigms. On the one hand, our model-based approach parameterizes the value function using a neural network and derives the policy from the feedback form. This approach incorporates the system's physics by penalizing the Hamilton-Jacobi-Bellman equation satisfied by the value function during training. On the other hand, the data-driven approach adopts state-of-the-art actor-critic RL frameworks and trains separate networks for the policy (actor) and the value function (critic). Both networks are trained exclusively from observational data. Despite drawing inspiration from control theory, particularly Dynamic Programming, both approaches exhibit distinct behaviors in practical applications.

Although we strived to use almost identical learning problems in both paradigms, we had to allow for some differences to account for their assumptions. Most importantly, our model-based approach had to use a simpler network architecture to ensure differentiability with respect to the inputs needed to apply the feedback form. While non-differentiable off-the-shelf CNN architectures worked well for the data-driven approach, the RL approach failed to achieve adequate results using the simpler architecture.

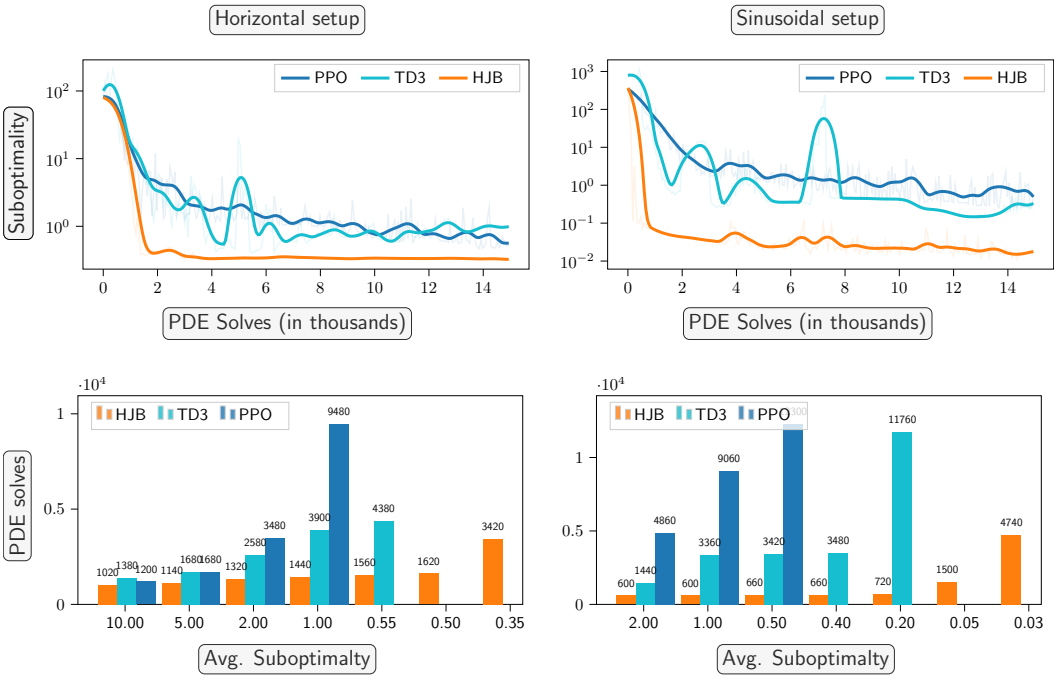


FIGURE 5. Suboptimality, relative to the baseline, on validation problems for the horizontal (left column) and sinusoidal (right column) problem setups.

We apply both approaches to a model problem inspired by contaminant containment to assess accuracy and efficiency. In this scenario, the system's state is governed by a two-dimensional advection-diffusion PDE. The objective is to prevent the contaminant from reaching a target region by controlling the location and intensity of a sink, with the parameter reflecting various scenarios for the source locations and velocity fields.

While both approaches follow an unsupervised training regime, our numerical results reveal that the model-based approach approximates a reasonable policy with much lower sub-optimality when compared to a gradient-based baseline. Notably, it achieves this with far fewer PDE solves than RL (see Figure 5). The superior performance of the model-based approach can be attributed to the incorporation of physics and system derivatives guided by control theory. In contrast, RL lacks awareness of this information and demands a larger number of PDE solutions. During RL training, many models showed promising learning for several thousand PDE solves but eventually diverged and were unable to recover. This phenomenon never occurred in the model-based approach.

Despite requiring a large number of PDE solves to learn the value function, RL algorithms are widely applicable due to their non-intrusive and gradient-free characteristics. This makes them particularly suitable for experimenting with new problems that lack well-defined models or face challenges in efficiently computing feedback forms and system derivatives.

The model-based approach also benefits from a more straightforward hyperparameter setup. Compared to RL approaches, it requires fewer hyperparameters, including the number of time steps $N$, the network width $m$, ResNet depth (the number of layers) $M$, and the multipliers $\beta_1$, $\beta_2$, $\beta_3$. Effective hyperparameters identified for one problem setup could

be seamlessly applied to other problems without modification. Conversely, RL training procedures proved sensitive to hyperparameter choices, often necessitating re-tuning between problems for convergence. This again emphasizes the data efficiency of the model-based approach.

As part of our future work, we plan to extend our model-based approach to tackle more complex problems, including those that lack well-defined models or closed-form solutions for the feedback form. We also intend to adapt our methods to handle changes in parameters over time. Additionally, we will investigate how both smooth and non-smooth dependencies of states and controls on uncertain parameters affect outcomes. While this work currently focuses on examples with smooth parameter dependencies, addressing non-smooth dependencies will require revisiting the derivations for the model-based approach and potentially modifying neural network architectures for both methods to address these challenges.

Furthermore, we aim to explore the scalability of our approaches by using increasingly finer discretizations and higher-dimensional parameter spaces. To achieve this, we will need to efficiently implement the model-based approach to handle the extensive memory requirements for solving PDEs when using dense solvers in PyTorch. Another interesting avenue for exploration could be the investigation of the convergence behavior of the model-based approach.

## Acknowledgments

LR's and DV's work was supported in part by NSF awards DMS 1751636 and DMS 2038118, AFOSR grant FA9550-20-1-0372, and US DOE Office of Advanced Scientific Computing Research Field Work Proposal 20-023231. NW's and BV's work was supported in part by Sandia National Laboratories which is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525, and US Department of Energy, and Office of Advanced Scientific Computing Research, Field Work Proposal 20-023231.

## References

[1] Giacomo Albi, Sara Bicego, and Dante Kalise. Gradient-augmented supervised learning of optimal feedback laws using state-dependent riccati equations. *IEEE Control Systems Letters*, 6:836–841, 2022.

[2] Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021.

[3] Harbir Antil. Mathematical opportunities in digital twins (math-dt). *arXiv preprint arXiv:2402.10326*, 2024.

[4] Omur Aydogmus and Musa Yilmaz. Comparative analysis of reinforcement learning algorithms for bipedal robot locomotion. *IEEE Access*, 2023.

[5] Francesco Ballarin, Elena Faggiano, Andrea Manzoni, Alfio Maria Quarteroni, Gianluigi Rozza, Sonia Ippolito, Carlo Antona, and Roberto Scrofani. Numerical modeling of hemodynamics scenarios of patient-specific coronary artery bypass grafts. *Biomechanics and Modeling in Mechanobiology*, 16:1373–1399, 2017.

[6] Somil Bansal and Claire J. Tomlin. DeepReach: A deep learning approach to high-dimensional reachability. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824, 2021.

[7] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

[8] Dimitri P. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Optimization and Computation Series. Athena Scientific, Belmont, MA, [2019] ©2019. Second printing with editorial revisions.

[9] Rodrigo C. Carlson, Ioannis Papamichail, Markos Papageorgiou, and Albert Messmer. Optimal motorway traffic flow control involving variable speed limits and ramp metering. *Transportation Science*, 44(2):238–253, 2010.

[10] Souvik Chakraborty, Sondipon Adhikari, and Ranjan Ganguli. The role of surrogate models in the development of digital twins of dynamic systems. *Applied Mathematical Modelling*, 90:662–681, 2021.

[11] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. Top-k off-policy correction for a reinforce recommender system. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2018.

[12] Nicola Demo, Maria Strazzullo, and Gianluigi Rozza. An extended physics informed neural network for preliminary analysis of parametric optimal control problems. *arXiv:2110.13530*, 2021.

[13] Sergey Dolgov, Dante Kalise, and Luca Saluzzi. Data-driven tensor train gradient cross approximation for hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 45(5):A2153–A2184, 2023.

[14] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[15] Wendell H. Fleming and H. Mete Soner. *Controlled Markov Processes and Viscosity Solutions*, volume 25 of *Stochastic Modelling and Applied Probability*. Springer, New York, second edition, 2006.

[16] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

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

[19] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*, 2022.

[20] Rakhoon Hwang, Jae Yong Lee, Jin Young Shin, and Hyung Ju Hwang. Solving PDE-constrained control problems using operator learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4504–4512, 2022.

[21] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4, 2019.

[22] Alex Irpan. Deep reinforcement learning doesn't work yet. 2018.

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

[24] Karl Kunisch and Daniel Walter. Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation. *ESAIM: Control, Optimisation and Calculus of Variations*, 27, 2021.

[25] Karl Kunisch and Daniel Walter. Optimal feedback control of dynamical systems via value-function approximation. *Comptes Rendus. Mécanique*, 351(S1):535–571, 2023.

[26] Karl Kunisch and Daniel Walter. Optimal feedback control of dynamical systems via value-function approximation. *arXiv:2302.13122*, 2023.

[27] Thomas Nakken Larsen, Halvor Ødegård Teigen, Torkel Laache, Damiano Varagnolo, and Adil Rasheed. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, 8:738113, 2021.

[28] Xingjian Li, Deepanshu Verma, and Lars Ruthotto. A neural network approach for stochastic optimal control. *arXiv:2209.13104*, 2022.

[29] Victor G Lopez, Frank L Lewis, Yan Wan, Edgar N Sanchez, and Lingling Fan. Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors. *IEEE Transactions on Automatic Control (TAC)*, 65(5):1911–1923, 2019.

[30] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

[31] Dingcheng Luo, Thomas O'Leary-Roseberry, Peng Chen, and Omar Ghattas. Efficient pde-constrained optimization under high-dimensional uncertainty using derivative-informed neural operators, 2023.

[32] Kjetil O Lye, Siddhartha Mishra, Deep Ray, and Praveen Chandrashekar. Iterative surrogate model optimization (ISMO): an active learning algorithm for PDE constrained optimization with deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 374:113575, 2021.

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

[34] Saviz Mowlavi and Saleh Nabi. Optimal control of PDEs using physics-informed neural networks. *Journal of Computational Physics*, page 111731, 2022.

[35] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high dimensional Hamilton-Jacobi-Bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.

[36] Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.

[37] Derek Onken, Levon Nurbekyan, Xingjian Li, Samy Wu Fung, Stanley Osher, and Lars Ruthotto. A neural network approach for high-dimensional optimal control. *arXiv:2104.03270*, 2021.

[38] Gianluigi Rozza, Andrea Manzoni, and Federico Negri. Reduction strategies for PDE-constrained optimization problems in haemodynamics. In *Proceedings of the 6th European Congress on Computational Methods in Applied Sciences and Engineering*, number CONF, pages 1748–1769. Vienna Technical University, 2012.

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

[40] Hou Shengren, Edgar Mauricio Salazar, Pedro P Vergara, and Peter Palensky. Performance comparison of deep rl algorithms for energy systems optimal scheduling. In *2022 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE, 2022.

[41] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[42] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140 – 1144, 2018.

[43] Maria Strazzullo, Francesco Ballarin, Renzo Mosetti, and Gianluigi Rozza. Model reduction for parametrized optimal control problems in environmental marine sciences and engineering. *SIAM Journal on Scientific Computing*, 40(4):B1055–B1079, 2018.

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

[45] Fredi Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods and Applications*, volume 112 of *Graduate Studies in Mathematics*. 2010.

[46] Sifan Wang, Mohamed Aziz Bhouri, and Paris Perdikaris. Fast PDE-constrained optimization via self-supervised operator learning. *arXiv:2110.13297*, 2021.

[47] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Joshua B. Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, 2017.

[48] Mengfei Xu, Shufang Song, Xuxiang Sun, Wengang Chen, and Weiwei Zhang. Machine learning for adjoint vector in aerodynamic shape optimization. *Acta Mechanica Sinica*, pages 1–17, 2021.

[49] Peng-Heng Yin, Guangqiang Xiao, Keju Tang, and Chao Yang. AONN: An adjoint-oriented neural network method for all-at-once solutions of parametric optimal control problems. *arXiv:2302.02076*, 2023.

[50] Jiongmin Yong and Xun Yu Zhou. *Stochastic controls*, volume 43 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 1999. Hamiltonian systems and HJB equations.

[51] Mo Zhou, Jiequn Han, and Jianfeng Lu. Actor-critic method for high dimensional static Hamilton–Jacobi–Bellman partial differential equations based on neural networks. *SIAM Journal on Scientific Computing*, 43(6):A4043–A4066, jan 2021.

[52] Mo Zhou and Jianfeng Lu. A policy gradient framework for stochastic optimal control problems with global convergence guarantee. *arXiv:2302.05816*, 2023.

[53] Hao Dong Zihan Ding. Challenges of reinforcement learning. In Shanghang Zhang Hao Dong, Zihan Ding, editor, *Deep Reinforcement Learning: Fundamentals, Research, and Applications*, chapter 7, pages 249–272. Springer Nature, 2020. http://www.deepreinforcementlearningbook.org.

*Department of Mathematics, Emory University, Atlanta, GA
*Email address*: dverma4@emory.edu

†Departments of Mathematics and Computer Science, Emory University, Atlanta, GA
*Email address*: lruthotto@emory.edu

‡Sandia National Laboratories, Albuquerque, NM
*Email address*: {nwinovi, bartv}@sandia.gov