DATA-DRIVEN HAMILTONIAN FOR NEURAL OPTIMAL CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) has emerged as a prominent data-driven approach to optimal control, but often requires substantial amounts of data and provides limited guarantees when encountering out-of-distribution state-action pairs. In this paper, we propose an alternative approach grounded in the well-established Hamilton-Jacobi framework in optimal control theory. Rather than requiring an explicit analytical model of the system dynamics to evaluate the Hamiltonian, we replace this with a data-driven Hamiltonian (DDH), which we generalize to a broad class of optimal control problems. In contrast to standard RL approaches, the DDH provides a principled mechanism, requiring only minimal prior system information such as a Lipschitz constant, for quantifying and accounting for uncertainty in regions not covered by the data, enabling the derivation of controllers with performance guarantees. Additional prior system knowledge, when available, can be readily incorporated to improve performance. For high-dimensional systems, we further introduce a physics-informed neural approximation of the value function, trained using a novel procedure that generates worst-case fictitious rollouts evaluated through the DDH. On an LQR problem, the DDH reaches near-optimal cost on every dataset we test, including random data on which offline RL methods remain far from optimal, and needs far fewer samples than online RL. On a nonlinear lunar lander benchmark, the DDH achieves an optimality gap to a trajectory-optimization reference more than $3\times$ smaller than CQL and FQL, and matches the performance of SAC with approximately $37\times$ fewer samples. These results establish the DDH as a promising framework for data-driven optimal control, particularly in settings where available data are limited in quantity or quality.

1 INTRODUCTION

While optimal control is important across a wide range of applications, including autonomous driving, aeronautics, robotics, HVAC regulation, and chemical process control, the dynamics of such complex systems are often challenging to model accurately. Reinforcement learning (RL) provides a prominent approach to control under unknown dynamics, as it can learn policies through interactions with the system without requiring an explicit dynamics model. However, RL is often sample inefficient, requiring substantial interaction data to learn an effective policy. Moreover, learned policies can perform poorly when encountering states and actions that are insufficiently represented in the training distribution, despite extensive efforts to mitigate such distribution shift (Fujimoto et al., 2019; Kumar et al., 2020; Yu et al., 2020; Kang et al., 2022). This limitation is particularly relevant in offline RL, where the performance still remains strongly dependent on the quality and coverage of the available data (Park et al., 2024). At the same time, even when constructing an accurate analytical model is difficult, coarse prior information about the system—such as smoothness properties or bounds on the dynamics—is often readily available. Standard model-free RL formulations, however, provide limited mechanisms for directly incorporating such information to improve data-efficiency, out-of-distribution behavior, or performance guarantees.

To address these limitations of existing approaches, we propose an alternative data-driven approach for optimal control problems. Our approach builds on the Hamilton-Jacobi-Bellman (HJB) framework for continuous-time optimal control, where the value function, representing the optimal cost-to-go, is characterized as the solution of the HJB partial differential equation (PDE) (Bardi et al., 1997). The optimal policy is then derived from this value function. Standard solution methods for

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the HJB-PDE require an explicit model of the system dynamics to evaluate the Hamiltonian, whose optimization objective depends directly on the dynamics. Thus, when the dynamics are unknown, the Hamiltonian cannot be evaluated directly, limiting the applicability of the HJB framework to data-driven control. Previous work addressed this challenge for reachability problems, a particular class of optimal control problems concerned with reaching or avoiding sets of states, by deriving a direct data-driven conservative bound on the Hamiltonian called the Data-Driven Hamiltonian (DDH) (Choi et al., 2025). The DDH quantifies the worst-case uncertainty consistent with the available dataset through a min–max formulation, using only minimal prior system information such as a verified Lipschitz constant of the vector field. Through this formulation, DDH values at unseen state-action pairs provide conservative bounds on the unknown true Hamiltonian. This conservative Hamiltonian can then be incorporated into the HJB-PDE, producing an estimated safe set from which the reachability task is guaranteed to be achievable.

We first extend the DDH to a broader class of optimal control problems by incorporating *running costs*, which are common in RL problems but absent from reachability problems. To scale our approach to high-dimensional systems, we further incorporate the DDH into the learning of neural value functions that approximate solutions to the HJB-PDE. Crucially, the PDE residual, which serves as the primary training signal for these value functions (Karniadakis et al., 2021), can be evaluated at arbitrary states using the DDH. This enables dense supervision throughout the entire state space, including regions not covered by the available data, and thereby facilitates learning a reliable value function. This stands in sharp contrast to standard model-free deep RL, where Bellman-based training signals are evaluated only on sampled transitions, and are thus fundamentally tied to states and actions represented in the collected experience or offline dataset. Similarly, rollouts in model-based RL methods (Janner et al., 2019; Yu et al., 2020), which use learned dynamics models, are typically initialized from observed states and kept short to limit the effect of out-of-distribution model error.

Next, to further improve the accuracy of the learned solution, we propose a novel algorithm that augments the PDE residual loss with an additional supervised training signal, analogous to Monte Carlo targets commonly used in RL to improve the reliability of value function learning (Sutton & Barto, 2018). By exploiting the min–max formulation underlying the DDH, we generate long-horizon fictitious rollouts that capture the effect of worst-case dynamics uncertainty consistent with the available data, and use the resulting trajectories to construct empirical value targets for direct supervision. This idea is inspired by MPC-guided value-function training in Feng et al. (2025); crucially, however, our fictitious rollouts require neither an explicit dynamics model nor access to a simulator and can be initialized from arbitrary states in the state space.

We validate our approach on a two-dimensional LQR problem and compare the DDH's performance to a variety of RL algorithms. We find that the DDH performs robustly across datasets of different qualities, especially outperforming the offline RL algorithms on suboptimal demonstrations. This supports our intuition that the DDH is well-suited to tasks where the state-action distribution in the dataset differs substantially from the one the optimal policy visits. With a dataset consisting of random trajectories, the DDH is within 0.1% of the optimal cost, whereas the offline RL methods CQL (Kumar et al., 2020), IQL (Kostrikov et al., 2021), and FQL (Park et al., 2025) remain 13%, 26%, and 61% worse than optimal. Against online RL, SAC (Haarnoja et al., 2018a) and PPO (Schulman et al., 2017), the DDH matches their best performance on every dataset with at least $25\times$ and $100\times$ fewer samples, respectively. We also evaluate the neural DDH on a 6-D lunar lander system. We find that the neural DDH reduces the optimality gap to a trajectory-optimization reference by more than $3\times$ compared to CQL and FQL, and matches the performance of SAC with approximately $37\times$ and PPO with $2{,}300\times$ less data.

Related Work. The most relevant related work is referenced above, and a detailed overview of the related areas is given in Section A.1.

Notations. $\|\cdot\|$ is the l_2 norm and $|\cdot|$ denotes an absolute value. $\mathrm{B}(\mathbf{c}, r) = \{\mathbf{v} \mid \|\mathbf{v} - \mathbf{c}\| \leq r\}$ denotes an l_2 ball centered at $\mathbf{c}$ with radius r. $\mathrm{Rect}(\mathbf{c}, \mathbf{r}) = \{\mathbf{v} \mid |v_j - c_j| \leq r_j, \forall j = 1, \cdots, n\}$ denotes a hyperrectangle centered at $\mathbf{c}$ with radii $\mathbf{r} \in \mathbb{R}^n$. The superscript i denotes the index of a datapoint and the subscript j denotes the j-th element of a vector. Vectors are written in bold.

2 Problem Formulation and Background

System Definition and Assumptions. We consider nonlinear system dynamics

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

with state $\mathbf{x}(t) \in \mathbb{R}^{n_x}$, control $\mathbf{u}(t) \in \mathcal{U}(\mathbf{x}(t)) \subseteq \mathbb{R}^{n_u}$, and initial state $\mathbf{x}_0$ at time t_0, and where $\mathcal{U}(\mathbf{x})$ is a state-dependent set of admissible controls. The vector field f is assumed to be Lipschitz continuous in the state. While the dynamics f itself is deemed unknown, we assume that (i) a dataset of trajectories from this uncertain system is given as $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}, \mathbf{v}^{(i)})\}_{i=1}^{N}$, where $\mathbf{v}^{(i)} := f(\mathbf{x}^{(i)}, \mathbf{u}^{(i)})$ denotes "state velocity" and is known exactly, and (ii) we have access to some high-level information about the system that allows us to bound the uncertainty on the dynamics at a given state, for example the state Lipschitz constant L^x of f: $\|f(\mathbf{x}, \mathbf{u}) - f(\mathbf{x}', \mathbf{u})\| \leq L^x \|\mathbf{x} - \mathbf{x}'\|$. We also define the control Lipschitz constant L^u of f as $\|f(\mathbf{x}, \mathbf{u}) - f(\mathbf{x}, \mathbf{u}')\| \leq L^u \|\mathbf{u} - \mathbf{u}'\|$. Knowing or estimating such constants requires far less modeling effort than identifying f, although it makes our method not fully model-free. We will use L^x and L^u as our running example throughout this paper of a known property of the system that allows us to bound the uncertainty, but we provide several other examples in Section A.3.

Optimal Control Problem. We consider a continuous-time optimal control problem with a running cost. We start at time t_0 and end at time t_f, where $t_f \in \mathbb{R} \cup \{\infty\}$. $\mathcal{U}_{t_0}^{t_f}$ denotes the set of control signals from time t_0 to t_f. The cost function we optimize over is

$$J(\mathbf{x}_0, \mathbf{u}(\cdot); t_0, t_f) = \int_{t_0}^{t_f} e^{-\gamma(t-t_0)} c(\mathbf{x}(t), \mathbf{u}(t)) dt + e^{-\gamma(t_f - t_0)} c_f\left(\mathbf{x}(t_f)\right) \tag{1}$$

with discount factor $\gamma \geq 0$. We will refer to the function c as the running cost and c_f as the terminal cost. Our optimal control problem starting from state $\mathbf{x}_0$ at time t_0 is written as $\inf_{\mathbf{u}(\cdot) \in \mathcal{U}_{t_0}^{t_f}} J(\mathbf{x}_0, \mathbf{u}(\cdot); t_0, t_f)$. Our goal is to approximately solve this problem given access to our dataset $\mathcal{D}$ of state transitions.

Hamilton-Jacobi-Bellman Equation. The Hamilton-Jacobi-Bellman (HJB) framework solves optimal control problems through dynamic programming (Evans, 2022). We first define our value function $V(\mathbf{x}, t)$ as the optimal cost starting from state $\mathbf{x}$ at time t: $V(\mathbf{x}, t) = \inf_{\mathbf{u} \in \mathcal{U}_t^{t_f}} J(\mathbf{x}, \mathbf{u}(\cdot); t, t_f)$. Applying the principle of dynamic programming yields the HJB-PDE with terminal condition

$$\frac{\partial V}{\partial t}(\mathbf{x}, t) = -\left(H(\mathbf{x}, \mathbf{p}) - \gamma V(\mathbf{x}, t)\right); \quad V(\mathbf{x}, t_f) = c_f(\mathbf{x}), \tag{2}$$

where $\mathbf{p} = \frac{\partial}{\partial \mathbf{x}} V(\mathbf{x}, t)$ and the Hamiltonian $H(\mathbf{x}, \mathbf{p})$ is

$$H(\mathbf{x}, \mathbf{p}) = \min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \mathbf{p}^\top f(\mathbf{x}, \mathbf{u}) + c(\mathbf{x}, \mathbf{u}). \tag{3}$$

The optimal control at state $\mathbf{x}$ is given by $\mathbf{u}^* = \arg\min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \mathbf{p}^\top f(\mathbf{x}, \mathbf{u}) + c(\mathbf{x}, \mathbf{u})$. More formally, the value function is the unique viscosity solution to the HJB-PDE (Bardi et al., 1997)[1]. For an infinite-time problem with discounting and no terminal cost, the static value function arrived at as $t_f \to \infty$ is uniquely defined by the static discounted HJB-PDE: $H(\mathbf{x}, \mathbf{p}) - \gamma V(\mathbf{x}) = 0$.

To solve Equation (3) directly, we require knowledge of the dynamics f. As a result, the Hamiltonian is the key entity we will need to replace when our dynamics are unknown.

Data-Driven Hamiltonian. The previous work of Choi et al. (2025) focused on bounding the Hamiltonian for reachability problems, which concern driving a system to reach or avoid particular sets of states and have zero running cost. It introduced the Data-Driven Hamiltonian, which provides a conservative bound on the Hamiltonian in the form of

$$\min_{i \in \{1, \ldots, N\}} \max_{\mathbf{v} \in \mathcal{E}(\mathbf{x}; \mathbf{x}^{(i)})} \mathbf{p}^\top \mathbf{v}, \tag{4}$$

where $\mathcal{E}(\mathbf{x}; \mathbf{x}^{(i)})$ must include the true state velocity at state $\mathbf{x}$ under control $\mathbf{u}^{(i)}$. Our approach generalizes this definition of the Data-Driven Hamiltonian and uncertainty set.

[1]When we refer to a potentially non-differentiable function V as a solution to a PDE, we interpret it as a weak solution. This notion is formalized through the theory of viscosity solutions (Bardi et al., 1997).

3 APPROACH: THE DATA-DRIVEN HAMILTONIAN FOR OPTIMAL CONTROL

Extending Choi et al. (2025), we approach our problem by upper-bounding our Hamiltonian. This conservative estimate of the Hamiltonian results in a conservative value function that upper-bounds the true value function. We will loosely refer to any function $\widehat{H}(\mathbf{x}, \mathbf{p})$ that provides an upper bound on the Hamiltonian from the data, without requiring the exact system dynamics, as a data-driven Hamiltonian (DDH).

We introduce one form that such an upper bound can take. We start with the concept of an *uncertainty set* $\mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D})$. This represents a set of possible state velocities that could occur at state $\mathbf{x}$ with control $\mathbf{u}$. The only requirement of this set is that it must include the true state velocity $f(\mathbf{x}, \mathbf{u}) \in \mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D})$. For instance, given a Lipschitz constant on the state L^x, control L^u, and a single datapoint $(\mathbf{x}^0, \mathbf{u}^0, \mathbf{v}^0)$ we could define $\mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D}) = \mathrm{B}(\mathbf{v}^0, L^x \left\|\mathbf{x} - \mathbf{x}^0\right\| + L^u \left\|\mathbf{u} - \mathbf{u}^0\right\|)$. We also introduce a search set $\mathcal{U}_{\text{search}}(\mathbf{x}) \subseteq \mathcal{U}(\mathbf{x})$, which represents the set of controls under consideration. We can then define a generalization of Equation (4) (Choi et al., 2025):

Data-Driven Hamiltonian (DDH):
$$\widehat{H}(\mathbf{x}, \mathbf{p}) := \min_{\mathbf{u} \in \mathcal{U}_{\text{search}}(\mathbf{x})} \max_{\mathbf{v} \in \mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D})} \mathbf{p}^\top \mathbf{v} + c(\mathbf{x}, \mathbf{u}) \tag{5}$$

The corresponding control will be the optimal $\mathbf{u}$ for this problem. Any uncertainty set from Choi et al. (2025) that bounds $\mathbf{p}^\top \mathbf{v}$ can be applied here since $c(\mathbf{x}, \mathbf{u})$ does not depend on $\mathbf{v}$. We see that the DDH upper-bounds the Hamiltonian since

$$\widehat{H}(\mathbf{x}, \mathbf{p}) \geq \min_{\mathbf{u} \in \mathcal{U}_{\text{search}}(\mathbf{x})} \mathbf{p}^\top f(\mathbf{x}, \mathbf{u}) + c(\mathbf{x}, \mathbf{u}) \geq \min_{\mathbf{u} \in \mathcal{U}} \mathbf{p}^\top f(\mathbf{x}, \mathbf{u}) + c(\mathbf{x}, \mathbf{u}) = H(\mathbf{x}, \mathbf{p}),$$

where the first inequality holds since $f(\mathbf{x}, \mathbf{u}) \in \mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D})$, and the second holds since $\mathcal{U}_{\text{search}}(\mathbf{x}) \subseteq \mathcal{U}$. If we use the DDH in place of the Hamiltonian when solving for a value function, the resulting value function is guaranteed to be a conservative bound on the true value function, $\widehat{V}(\mathbf{x}, t) \geq V(\mathbf{x}, t)$, following a similar argument to that in Choi et al. (2025). We will instantiate several example uncertainty and search sets in the case where we know L^x and L^u, as defined in Section 2. For examples with other system information, see Section A.3.

State-Only. To recover a similar DDH to that in Choi et al. (2025) we can define a DDH from the ith datapoint as $\widehat{H}_i$ with $\mathcal{U}_{\text{search}} = \{\mathbf{u}^{(i)}\}$ and $\mathcal{E}^{(i)}(\mathbf{x}, \mathbf{u}^{(i)}; \mathcal{D}) = \mathrm{B}\left(\mathbf{v}^{(i)}, L^x \left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|\right)$ and then take the minimum over the bound from each datapoint as in Equation (6) (see Section A.2). We will refer to this as the *state-only DDH* . This results in a computationally tractable DDH since $\widehat{H}_i$ maximizes a linear function over a ball, which has a closed-form solution.

State-Control. One limitation of the state-only DDH is that the control obtained from the DDH must be a control already observed in the dataset from the definition of its search set. This can lead to a chattering control signal and jittery movement. We can search outside these observed controls and provide a tighter bound by introducing the following *state-control DDH* , with $\widehat{H}_i$ defined by $\mathcal{U}^{(i)}_{\text{search}} = \mathcal{U}$ and $\mathcal{E}^{(i)}(\mathbf{x}, \mathbf{u}; \mathcal{D}) = \mathrm{B}\left(\mathbf{v}^{(i)}, L^x \left\|\mathbf{x} - \mathbf{x}^{(i)}\right\| + L^u \left\|\mathbf{u} - \mathbf{u}^{(i)}\right\|\right)$. Here we now search over the whole control space, making use of L^u to bound the behavior of the system on arbitrary controls. This is a strict tightening on the state-only DDH , since we have the same uncertainty set but search over a larger search set. If each $\widehat{H}_i$ is expensive to compute, we can choose to only apply this tightening to a subset of the datapoints. One heuristic to choose these datapoints is via the datapoints that achieve the top k bounds in the state-only DDH . We will refer to the DDH where we first compute the state-only DDH then tighten only its top k bounds as the *k-tightened DDH* .

3.1 NEURAL DATA-DRIVEN HAMILTONIAN-BASED VALUE FUNCTION

Grid-based solvers for the HJB-PDE scale exponentially with the state dimension, so for high-dimensional systems, we use a neural network to approximate the solution of the PDE. This resembles minimizing a Bellman residual, except that the residual comes from differentiating the network rather than from a sampled next state. Because the DDH is defined at every state, this residual can be evaluated anywhere in the state space, not only at states in the dataset, which is a key advantage over most model-free RL methods. We start by introducing a parameterized approximation of the value function, the solution to the HJB-PDE, $V_\theta(\mathbf{x}, t)$. Since we want the learned value function to

Algorithm 1 Neural DDH training (static, discounted)

Require: dataset $\mathcal{D}$, sensitivity bounds, running cost c, discount rate γ, weight λ_2, refresh period T_{sup}, rollout horizon τ, step Δt

initialize θ; $\mathcal{D}_{\text{sup}} \leftarrow \emptyset$
for $j = 1, 2, \ldots$ **do**
  sample $\mathcal{D}_{\text{PDE}}$ uniformly in $\mathcal{X}$
  if $j \bmod T_{\text{sup}} = 0$ **then**
    sample start states $\mathbf{x}_0$ in a supervision region $\mathcal{X}_{\text{sup}} \subseteq \mathcal{X}$
    roll out for $\tau/\Delta t$ steps, stopping early if the next state leaves $\mathcal{X}$: set $\mathbf{p} = \nabla_{\mathbf{x}} V_\theta(\mathbf{x})$, take $(\mathbf{u}^*, \mathbf{v}^*)$ from Equation (5), step $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}^* \Delta t$
    $\mathcal{D}_{\text{sup}} \leftarrow \{(\mathbf{x}_0, v_{\text{sup}})\}$, with $v_{\text{sup}} = \sum_{k<K} e^{-\gamma k \Delta t} c(\mathbf{x}_k, \mathbf{u}_k^*) \Delta t + e^{-\gamma K \Delta t} V_\theta(\mathbf{x}_K)$, where $\mathbf{x}_K$ is the final state (a fixed exit value is used if the rollout leaves $\mathcal{X}$)
  end if
  $\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}_{\text{PDE}}|} \sum_{\mathcal{D}_{\text{PDE}}} \left| \widehat{H}(\mathbf{x}, \nabla_{\mathbf{x}} V_\theta) - \gamma V_\theta(\mathbf{x}) \right| + \frac{\lambda_2}{|\mathcal{D}_{\text{sup}}|} \sum_{\mathcal{D}_{\text{sup}}} \left| V_\theta(\mathbf{x}_0) - v_{\text{sup}} \right|$
  update θ with a gradient step on $\mathcal{L}(\theta)$
end for

be a solution to the HJB-PDE, it needs to satisfy both the terminal condition and PDE given in Equation (2). This motivates two components of the loss function: a PDE loss and a terminal condition loss inspired by Bansal & Tomlin (2021):

$$\mathcal{L}_{\text{PDE}}(\mathbf{x}, t) = \left| \frac{\partial V}{\partial t}(\mathbf{x}, t) + \widehat{H}(\mathbf{x}, \mathbf{p}) - \gamma V(\mathbf{x}, t) \right|, \qquad \mathcal{L}_{\text{terminal}}(\mathbf{x}) = |V(\mathbf{x}, t_f) - c_f(\mathbf{x})|$$

The PDE loss enforces only local consistency; thus, as in one-step temporal-difference learning, errors can propagate slowly. Feng et al. (2025) addressed this in reachability by supervising the network with returns from MPC rollouts, which require a dynamics model. We instead propose an approach to generate fictitious rollouts using the worst-case dynamics implicit in our DDH. We will work with the form of the DDH in Equation (5). Let

$$\mathbf{u}^*(\mathbf{x}, \mathbf{p}), \mathbf{v}^*(\mathbf{x}, \mathbf{p}) = \underset{\mathbf{u} \in \mathcal{U}_{\text{search}}(\mathbf{x})}{\arg\min} \; \underset{\mathbf{v} \in \mathcal{E}(\mathbf{x}, \mathbf{u}; \mathcal{D})}{\arg\max} \; \mathbf{p}^\top \mathbf{v} + c(\mathbf{x}, \mathbf{u}).$$

We can then define adversarial dynamics that depend only on the dataset and V_θ: $f_{\text{adversarial}}(\mathbf{x}, \mathbf{u}^*(\mathbf{x}, \mathbf{p})) = \mathbf{v}^*(\mathbf{x}, \mathbf{p})$, where $\mathbf{p} = \nabla_{\mathbf{x}} V_\theta(\mathbf{x}, t)$. This corresponds to the fictitious dynamics introduced in the proof of Theorem 1 in Choi et al. (2025), resulting from a differential game where the leader chooses the control $\mathbf{u}^*$ and an adversarial follower chooses the worst-case realization of the dynamics $\mathbf{v}^*$ subject to the uncertainty set resulting from the choice of $\mathbf{u}^*$. We can then perform rollouts of the system using these adversarial dynamics despite not knowing the dynamics of the true system. Let $v_{\text{sup}}(\mathbf{x}, t)$ be the discounted accumulated cost obtained from a rollout starting at state $\mathbf{x}$ at time t under the adversarial dynamics, plus the discounted value from V_θ at the rollout's final state. For V_θ to be consistent we would like $V_\theta(\mathbf{x}, t) = v_{\text{sup}}(\mathbf{x}, t)$. Formally, since $f_{\text{adversarial}}$ corresponds to the adversarial dynamics from Choi et al. (2025), if V_θ perfectly represented the desired $\widehat{V}$ then $v_{\text{sup}}(\mathbf{x}, t) = \widehat{V}(\mathbf{x}, t) \geq V(\mathbf{x}, t)$. We define the supervision loss as $\mathcal{L}_{\text{rollout}}(\mathbf{x}, t) = |V_\theta(\mathbf{x}, t) - v_{\text{sup}}(\mathbf{x}, t)|$. This can be thought of as policy evaluation in a policy-iteration algorithm, as it evaluates the current policy and pulls the value function towards that value.

During training we sample datasets of state-time pairs for each of our losses, which we will denote as $\mathcal{D}_{\text{PDE}}$, $\mathcal{D}_{\text{term}}$, and $\mathcal{D}_{\text{sup}}$. As in Bansal & Tomlin (2021), a curriculum learning approach can be applied to the choice of t in the samples. Altogether, our empirical loss is

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}_{\text{PDE}}|} \sum_{\mathbf{x}_i, t_i \in \mathcal{D}_{\text{PDE}}} \mathcal{L}_{\text{PDE}}(\mathbf{x}, t) + \frac{\lambda_1}{|\mathcal{D}_{\text{term}}|} \sum_{\mathbf{x}_i \in \mathcal{D}_{\text{term}}} \mathcal{L}_{\text{terminal}}(\mathbf{x}) + \frac{\lambda_2}{|\mathcal{D}_{\text{sup}}|} \sum_{\mathbf{x}_i, t_i \in \mathcal{D}_{\text{sup}}} \mathcal{L}_{\text{rollout}}(\mathbf{x}, t)$$

with λ_1, λ_2 weights we can set to prioritize different losses. Since the rollouts are comparatively expensive to compute, the dataset $\mathcal{D}_{\text{sup}}$ can be fixed for a number of iterations before being refreshed. We can also learn an infinite-horizon static value function $V_\theta(\mathbf{x})$ when the discount factor $\gamma > 0$. The rollout supervision loss remains unchanged (just losing its dependence on time), while the PDE loss becomes $\mathcal{L}_{\text{PDE}}(\mathbf{x}) = |\widehat{H}(\mathbf{x}, \mathbf{p}) - \gamma V(\mathbf{x})|$ encouraging satisfaction of the static HJB equation. In

practice, we observe that learning this static value function leads to a more stable training problem, likely due to its contraction property. Algorithm 1 summarizes the training procedure. Note that our learning process can also be applied to the original reachability DDH in Choi et al. (2025).

4 Experimental Results

In this section, we investigate how the DDH compares to existing data-driven optimal control techniques. We begin by validating the DDH on a simple LQR problem. Through this case study, we are able to explore how the *type* of data the DDH has affects its performance, and how this differs from offline RL methods: the DDH appears to benefit from random (less optimal) demonstrations, whereas offline RL algorithms suffer from that randomness. We also evaluate the neural DDH on a lunar lander problem and see substantial advantages in data-efficiency over model-free RL methods.

4.1 Verification on an LQR Problem

We first validate the DDH on a simple two-dimensional linear system with quadratic cost. The dynamics are $\dot{\mathbf{x}} = A\mathbf{x}+B\mathbf{u}$ with $A = \left[\begin{smallmatrix}0 & 1\\ 0 & 0\end{smallmatrix}\right]$, $B = I$, state and control bounds $\mathcal{X} = [-2, 2]^2$ and $\mathcal{U} = [-4, 4]^2$, and finite-horizon LQR cost $J(\mathbf{x}_0, \mathbf{u}(\cdot)) = \int_0^T (\mathbf{x}^\top Q\mathbf{x}+\mathbf{u}^\top R\mathbf{u})\, dt+\mathbf{x}(T)^\top Q_f\mathbf{x}(T)$ with $Q = R = Q_f = I$, horizon $T = 2\,\text{s}$. When simulating the system, we use an Euler approximation with step size $\Delta t = 0.01\,\text{s}$. As the optimal reference, we apply the controller $\mathbf{u} = -K\mathbf{x}$ with K a constant gain from the finite-horizon Riccati differential equation solution. It achieves $J^\star = 2.96$ on a fixed set of $N_{\text{eval}} = 1{,}000$ initial conditions (ICs). A random policy attains ~ 39 on the same set. A disjoint set of 500 validation ICs is used for all checkpoint and hyperparameter selection.

DDH Variants. To isolate the effects of the uncertainty set, tightening, and function approximation, we consider seven DDH variants. Six variants come from the spherical and rectangular uncertainty sets from the L^x and $L^{x,\text{io}}$ Lipschitz constants respectively (see Section A.3) without tightening and with tightening ($k = 1$ or $k = 10$). The final variant uses the neural DDH, with a rectangular uncertainty set from $L^{x,\text{io}}$ and tightening of $k = 10$. For this system, $L^{x,\text{io}} = |A|$ and $L^{u,\text{io}} = |B|$ entrywise, so the rectangular set encodes the model up to the sign of each entry, while the spherical set uses only the scalar constants $L^x = 1$ and, with tightening, $L^u = 1$. For this low-dimensional study the neural DDH is trained with the PDE and terminal losses only. The full per-variant results are in Table 1, while Figure 6 shows the four most informative (both in Section A.5).

Datasets. We train all methods on four datasets: `optimal`, `noisy optimal`, `random trajectories`, and `uniform random`. For the `optimal` dataset, the known optimal controller was applied to generate trajectories. For the `noisy optimal` dataset, the same controller was applied but with additive Gaussian noise. For the `random trajectories` dataset, a uniformly random control drawn from the control bounds was applied to generate trajectories. For `uniform random`, states and controls were uniformly randomly sampled, and then their dynamics were evaluated (no trajectories were rolled out). Each trajectory dataset contains $100{,}000$ transitions (500 trajectories of 200 steps at $\Delta t = 0.01$), and every result at dataset size n uses the first n transitions, so each larger dataset is a strict superset of the smaller ones.

Baseline Algorithms. We compare the DDH against conservative Q-learning (CQL) (Kumar et al., 2020), implicit Q-learning (IQL) (Kostrikov et al., 2021), and flow Q-learning (FQL) (Park et al., 2025), three popular offline RL methods. We chose to compare against offline methods since they most closely match the problem setting we introduce for the DDH, where we have access only to a fixed offline dataset. As a reference for sample efficiency, we also train the online RL methods PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018a) directly on the true system ($\gamma = 0.995$, single seed). Since they collect their own data, they are not a like-for-like comparison. See Section A.4 for details on how the baselines were trained.

Results. Figure 1 shows each method's cost as the dataset grows. At $n = 20{,}000$ the DDH (rectangular, $k = 10$) is within 0.2% of $J^\star$ on every dataset, while on `random trajectories` CQL, IQL, and FQL remain 13%, 26%, and 61% above it. The two families favor opposite datasets. The DDH is within 1.5% of $J^\star$ from $n = 75$ on `uniform random` (within 4% in each of five dataset re-draws, Section A.4) and within 1% from $n = 1{,}000$ on `random trajectories`, but needs $n \approx 5{,}000$ to come within 2% on `optimal`, whose datapoints concentrate along optimal trajecto-

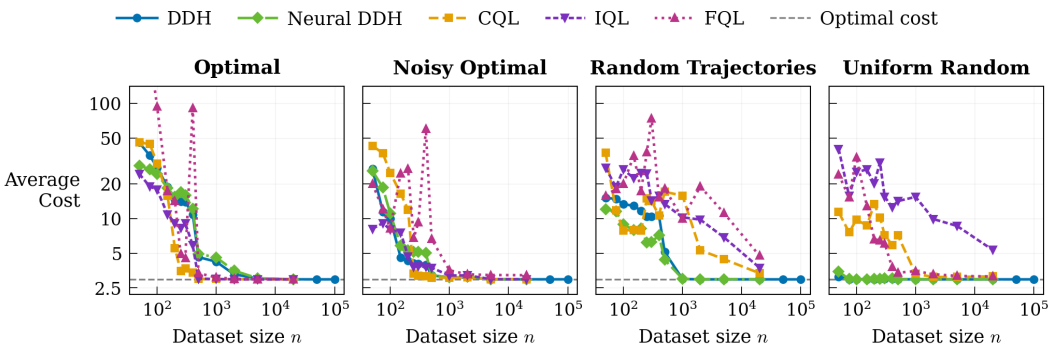


Figure 1: **The DDH and offline RL are data-efficient on opposite datasets.** Average rollout cost over 1,000 test initial conditions versus dataset size n (both log scale); dashed line: optimal reference (from the Riccati equation). FQL's behavior-cloning weight α is selected once per dataset on the validation set at $n = 20{,}000$ and reused for all n; two FQL points on `optimal` exceed the axis. The DDH improves fastest on datasets that cover the state and control space broadly, while offline RL improves fastest on expert data.

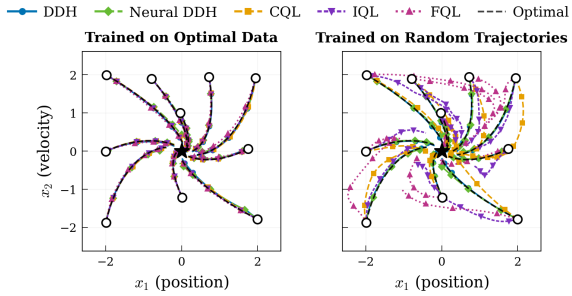


Figure 2: **Data quality affects offline RL's behavior but not the DDH's.** Ten trajectories using policies trained on `optimal` data (left) and `random trajectories` (right), $n = 20{,}000$. On optimal data all methods perform well. On random data, the DDH matches the optimal trajectory while the other algorithms drift away.

ries. CQL is the reverse: within 1% of $J^\star$ by $n = 500$ on `optimal`, but not converging on `random trajectories` by $n = 20{,}000$. IQL and FQL show the same pattern (Figures 1 and 2).

One possible explanation lies in how each method uses its data. The DDH uses each datapoint as evidence of how the system responds near that state and control, regardless of whether the recorded action was optimal. Even if an action was suboptimal in that state, it might be useful for neighboring states, and the DDH allows for this extrapolation in a structured way. As a result, while both families benefit from state coverage, the DDH benefits more from coverage of the control space. CQL, IQL, and FQL instead tend to keep their policies closer to the recorded actions (through a value penalty on unseen actions, advantage-weighted regression onto dataset actions, and a behavior-cloning term, respectively), which helps when those actions are optimal and hurts when they are random.

Among the DDH variants at $n = 500$ (Figure 6), the rectangular uncertainty set and tightening each lower the cost on most datasets, and the neural DDH is comparable to the grid solution. Even the spherical set, which uses only scalar Lipschitz constants, outperforms every offline RL baseline on `random trajectories` from $n = 500$ (Table 2). Online RL is far less data-efficient (Figure 3a): after 50k (SAC) and 526k (PPO) environment steps, neither reaches the 2.98 the DDH attains from 1,000 random transitions.

4.2 Lunar Lander Case Study

We evaluate the neural DDH against a variety of baselines on a 6-D lunar lander system. As in Inoue et al. (2026), we model the lunar lander with a 6-D state that includes the lunar lander's 2-D position (x, y), its rotation angle (θ), and their corresponding first derivatives $(v_x, v_y,$ and $\omega)$. The control is 3-D and consists of the main thruster and the right and left side thrusters (u_1, u_2, and u_3). The detailed dynamics, a diagram, and derivations of the associated Lipschitz constants can be found in Section A.7. We solve an infinite-time discounted optimal control problem with $\gamma = 0.33$, known equilibrium thrust $\mathbf{u}_0$, and running cost

$$c(\mathbf{x}, \mathbf{u}) = 1 - e^{-\sqrt{\mathbf{x}^\top Q \mathbf{x}}} + (\mathbf{u} - \mathbf{u}_0)^\top R(\mathbf{u} - \mathbf{u}_0), \quad Q = 0.01 \cdot \mathrm{diag}(1, 1, 1, 1, 10, 2), \quad R = 0.3I.$$

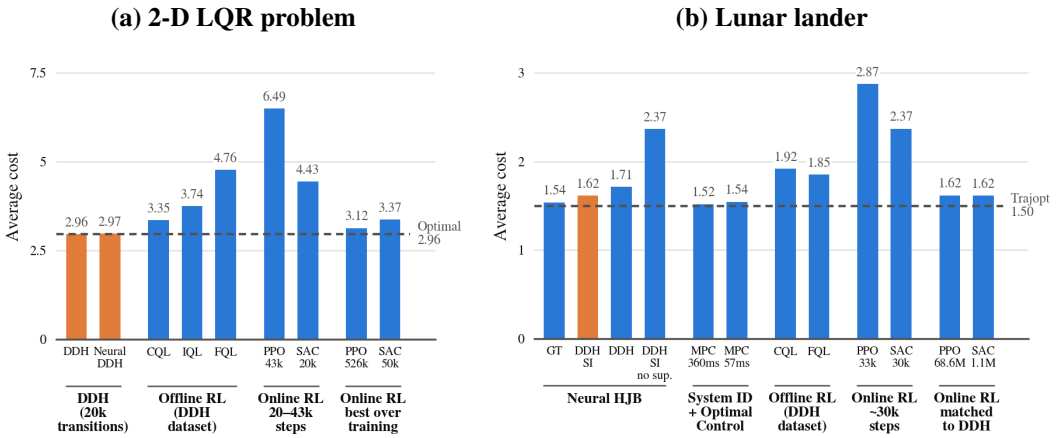


Figure 3: **The DDH outperforms model-free RL.** (a) Average cost on 1k initial conditions; offline methods use the same 20k transitions, and online RL interacts with the true system. The last group shows PPO and SAC at their best checkpoints, the end of training (526k and 50k steps). Online RL is single-seed; the CQL and FQL gaps hold across three seeds (Section A.4). (b) Average cost on the lunar lander evaluated on 20k initial conditions from the supervision region. Note that the training dataset was drawn from the full region, of which the supervision region constitutes roughly 4.8%. We evaluate on this smaller region so that more rollouts have the potential to be kept within the state space. The dashed line is a trajectory-optimization reference computed with the true dynamics. The DDH has more than a $3\times$ lower optimality gap than the offline RL algorithms, and SAC and PPO need roughly $37\times$ and $2{,}300\times$ more interaction to match the DDH's performance. Fitting a dynamics model and then applying MPC performs better, but must solve a trajectory optimization online at every re-planning step and lacks the built-in guarantees and principled out-of-distribution degradation of the DDH. We show two versions of MPC with different average runtimes.

The state space is $\mathcal{X} = \text{Rect}(\mathbf{0}, [15, 12, 15, 12, \pi, 3\pi])$ over $(x, v_x, y, v_y, \theta, \omega)$, and we evaluate on initial conditions drawn from an inner supervision region $\mathcal{X}_{\text{sup}} = \text{Rect}(\mathbf{0}, [7.5, 6, 7.5, 6, \pi, 7.1])$ from which we also draw the initial conditions for the DDH's supervision loss.

We apply the neural DDH to this problem with known sensitivity matrix $L^{x,\text{io}}$ (see Section A.3) and we further reduce the uncertainty set by incorporating the knowledge that three of the six states are single integrators of the other states (e.g., $\dot{x} = v_x$). Slices of the value function are visualized in Section A.8 and training details are in Section A.9. We compare against several representative offline RL algorithms (CQL and FQL), online RL algorithms (PPO and SAC), and a model-based RL baseline consisting of a system identification step where we fit a neural network model of the dynamics using supervised learning and then a control step where we apply MPC (Tables 6 and 7). The neural DDH and offline RL algorithms are trained on the same dataset, drawn uniformly at random from the state and control space (like the `uniform random` dataset we used for the LQR problem). In addition, we implemented a trajectory-optimization baseline that has access to the true dynamics. We treat the performance of the trajectory optimizer as an oracle that we can then compare each method against. We define each method's optimality gap as its average cost minus that of the trajectory optimizer. We evaluate all methods by rolling out 15 s long trajectories from a set of $20{,}000$ random initial conditions and compare their average costs.

Figure 3b compares the DDH to these baselines. We observe that the DDH substantially outperforms the offline RL algorithms, reducing the optimality gap by 67% compared to FQL and 72% compared to CQL. The model-based baseline, whose learned dynamics model uses no structural prior, outperforms the DDH. It must, however, solve an optimization problem online at every re-planning step, whereas the DDH controller evaluates the value gradient and selects among recorded controls. It also lacks the DDH's guarantee and its well-characterized degradation away from the data. Online RL given a comparable budget of about 30k environment steps performs substantially worse; matching the performance of the DDH takes SAC 1.1M steps and PPO 69M steps, $37\times$ and $2{,}300\times$ as much data respectively. Figure 3b also presents two ablations on the DDH: (i) removing the knowledge of the single integrator states, and (ii) removing the fictitious rollout-based supervision loss. We see that the single integrators clearly improve the DDH, but are less essential than

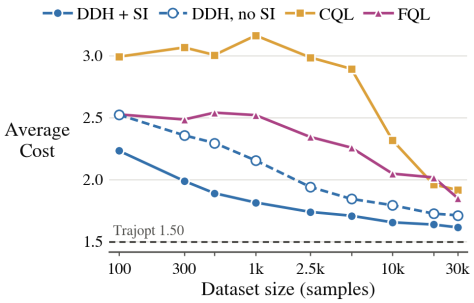


Figure 4: **The neural DDH is substantially more data-efficient than both CQL and FQL.** Average cost of the neural DDH, CQL, and FQL as the dataset grows, with and without knowledge of the single-integrator states $x, y,$ and θ for the DDH. CQL and FQL show the best cost over 3 training seeds. With just 1,000 samples, the DDH with known single integrators outperforms CQL and FQL on 30,000.

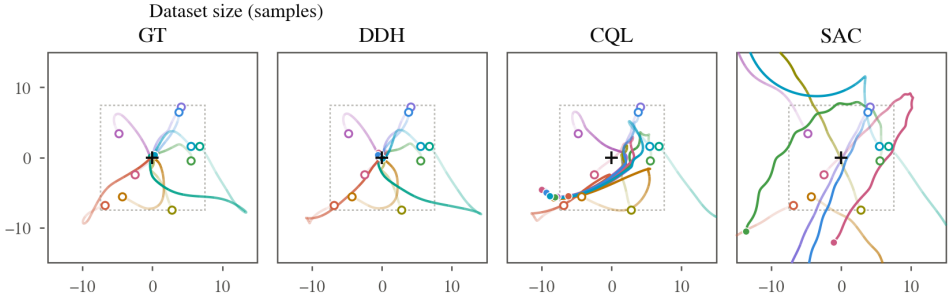


Figure 5: **The DDH's trajectories qualitatively outperform CQL and SAC.** The plots show a set of trajectories rolled out by (i) our neural approach with the true dynamics, (ii) the neural DDH, (iii) CQL, and (iv) SAC. CQL uses the same dataset as the DDH, while SAC uses the same number of samples (30k) but gathered through its online approach. The DDH drives the lander towards the origin, while CQL keeps the lunar lander within bounds but does not drive it to the origin, and SAC fails to keep the lunar lander within bounds.

supervision, without which the DDH drastically deteriorates. Figure 4 breaks down data-efficiency against CQL and FQL: with only 1,000 samples, the DDH outperforms CQL and FQL trained on all 30,000. The trajectories in Figure 5 reflect these costs.

5 Conclusion and Future Work

In this work, we extended the data-driven Hamiltonian (DDH) to a broader class of optimal control problems with running costs, and introduced a neural approximation to the solution to the HJB-PDE we call the neural DDH. The neural DDH greatly benefits from a novel approach we introduce to generate worst-case fictitious rollouts for its training. In comparison to most model-free RL algorithms, which only train on states in their dataset, the neural DDH is able to train its network across the entire state space by using the DDH to handle previously unobserved state-action pairs.

The DDH offers an alternative to the conservative penalties and heuristic uncertainty measures that RL typically applies to unseen states and actions: a worst-case bound computed from known properties of the system. Because this bound is defined everywhere, the value function can be trained everywhere, and prior knowledge can be directly incorporated. We expect this to be useful in robotics and aerospace, wherever some physical structure is known but a full model is not. We also hypothesize a core difference in what kind of dataset is best for the different methods: the DDH benefits most from broad coverage of states and actions, while offline RL benefits most from optimal demonstrations. The DDH is therefore a natural fit when consistent and high-quality expert demonstrations are hard to collect.

The DDH has several limitations. It requires enough system information to create a valid uncertainty set on the dynamics, and a cost function that can be evaluated at any state and action, rather than only on logged transitions. With only minimal information, such as a single Lipschitz constant, the uncertainty set can grow prohibitively large, especially in high-dimensional spaces. Overestimating the bound degrades cost gradually (Section A.6). Some directions for future work include extending offline reinforcement learning algorithms (e.g., CQL) to make use of some system information, bounds that combine information across several datapoints, pairing the DDH with a nominal dynamics model with a known maximum error, and extending the approach to accommodate stochastic dynamics.

Reproducibility Statement

A GitHub repository with the implementation of the neural DDH is available at [Redacted for anonymity], and the forks of the baselines we used for comparison can be found at [Redacted for anonymity]. See the supplementary materials, particularly Section A.4 and Section A.9 for details on training and implementation not found in the main body of the paper.

AI Use Statement

In this work, we used generative AI tools to support implementing the baseline methods for comparison (via the use of Claude Code during that implementation). We have not used generative AI tools for generating synthetic data sets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, or interpreting results.

Additionally, we used generative AI tools for creating visualizations while debugging and for creating and editing some miscellaneous areas in our software (again via Claude Code) and writing out some tables in latex. We have also used generative AI tools to cross-check numerical claims against our experimental results and to suggest edits to the abstract, captions, and experimental sections; all such text was reviewed, verified, and finalized by the authors.

We have reviewed all AI-assisted work. We have thoroughly tested our implementation and checked all data used to produce the resulting figures. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Mohammad Bajelani, Christopher A Strong, Claire J Tomlin, Jason J Choi, and Klaske van Heusden. Robust direct data-driven hamiltonian for safe set computation under measurement noise and disturbances. *arXiv preprint arXiv:2606.17292*, 2026.

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

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *IEEE Conference on Decision and Control (CDC)*, 2017.

Martino Bardi, Italo Capuzzo Dolcetta, et al. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, volume 12. Springer, 1997.

Julian Berberich, Johannes Köhler, Matthias A Müller, and Frank Allgöwer. Data-driven model predictive control with stability and robustness guarantees. *IEEE transactions on automatic control*, 66(4):1702–1717, 2020.

Jason J Choi, Christopher A Strong, Koushil Sreenath, Namhoon Cho, and Claire J Tomlin. Data-driven hamiltonian for direct construction of safe set from trajectory data. In *2025 IEEE 64th Conference on Decision and Control (CDC)*, pp. 2951–2958. IEEE, 2025.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Jeremy Coulson, John Lygeros, and Florian Dörfler. Data-enabled predictive control: In the shallows of the deepc. In *2019 18th European control conference (ECC)*, pp. 307–312. IEEE, 2019.

Lawrence C Evans. *Partial differential equations*, volume 19. American mathematical society, 2022.

Zeyuan Feng, Le Qiu, and Somil Bansal. Bridging model predictive control and deep learning for scalable reachability analysis. *arXiv preprint arXiv:2505.03830*, 2025.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Tsukasa Inoue, Sajjad Keshtkar, and Hirohisa Kojima. Lunar lander trajectory control using model predictive on–off thruster control. *Advances in Space Research*, 2026.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Katie Kang, Paula Gradu, Jason J Choi, Michael Janner, Claire Tomlin, and Sergey Levine. Lyapunov density models: Constraining distribution shift in learning-based control. In *International Conference on Machine Learning*, pp. 10708–10733. PMLR, 2022.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

Yankai Li and Mo Chen. HJRNO: Hamilton-jacobi reachability with neural operators. *arXiv preprint arXiv:2504.19989*, 2025.

Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline rl? *Advances in Neural Information Processing Systems*, 37:79029–79056, 2024.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction (second edition)*. MIT Press, 2018.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in neural information processing systems*, 33:14129–14142, 2020.

A Appendix

A.1 Related Work

There are a variety of data-driven approaches for controlling systems with unknown dynamics. Reinforcement learning is one of the richest fields concerning the optimal control of unknown systems. In model-free RL, online approaches, such as proximal policy optimization (PPO) (Schulman et al., 2017) and soft actor-critic (SAC) (Haarnoja et al., 2018a;b), explore the state-action space while learning state or state-action value functions. Offline RL approaches rely on a fixed pre-collected dataset of system transitions. Conservative Q-Learning (CQL) (Kumar et al., 2020) learns a state-action value function while penalizing out-of-distribution (OOD) actions to avoid an overoptimistic value function on OOD data. Implicit Q-Learning (IQL) (Kostrikov et al., 2021) avoids querying OOD actions altogether by fitting the value function with expectile regression over dataset actions. Flow Q-Learning (FQL) (Park et al., 2025), a more recent approach which learns a state-action value function, a one-step policy, and a flow-matching model of the behavior policy, effectively incorporates information from near-optimal demonstrations from the dataset. However, in all of these approaches, the value function supervision is fundamentally done at observed state transitions, leaving the behavior outside the support of the collected data only indirectly constrained.

Other offline methods, such as AWR (Peng et al., 2019) and TD3+BC (Fujimoto & Gu, 2021), similarly regularize the policy toward the data; we compare against CQL, IQL, and FQL as widely used representatives with strong reported performance.

Model-based RL tackles the same problem, and typically consists of (i) a model-learning stage, where an explicit dynamics model is learned, and (ii) the development of a controller using this model (Janner et al., 2019; Chua et al., 2018). However, model-based RL can also suffer from OOD issues when the dynamics model is used outside of its training data, particularly when restricted to an offline dataset. Model-Based Offline Policy Optimization (MOPO) (Yu et al., 2020) attempts to mitigate this by penalizing state-action pairs with higher uncertainty.

There is also a body of work in direct data-driven model predictive control (MPC). However, the majority of this work focuses on linear systems in order to leverage Willems' fundamental lemma (Berberich et al., 2020; Coulson et al., 2019), which restricts its generalizability.

The approach that we build on, the Data-Driven Hamiltonian (DDH), proposed by Choi et al. (2025) and extended by Bajelani et al. (2026), provides a conservative bound on the Hamiltonian, which allows Hamilton-Jacobi Reachability problems (Bansal et al., 2017) to be solved for systems whose dynamics are unknown. We directly extend this approach to the broader class of optimal control problems involving running costs, as well as to higher-dimensional systems, while retaining its key insights that (i) the Hamiltonian is where the system dynamics enter, and (ii) data and minimal system information can be used to bound this Hamiltonian. Finally, recent advances in physics-informed machine learning are applied to solve Hamilton-Jacobi reachability problems (Bansal & Tomlin, 2021; Feng et al., 2025; Li & Chen, 2025), where a model-based Hamiltonian is used to evaluate the PDE residual.

A.2 Modular Combination of DDHs

Given $\{\widehat{H}_i(\mathbf{x}, \mathbf{p})\}_{i=1}^n$, where $\widehat{H}_i$ are each valid DDHs, we can define the combined DDH as the minimum of the individual DDHs, since each DDH individually upper-bounds the Hamiltonian:

$$\widehat{H}(\mathbf{x}, \mathbf{p}) = \min_i \left\{ \widehat{H}_i(\mathbf{x}, \mathbf{p}) \right\} \geq H(\mathbf{x}, \mathbf{p}). \tag{6}$$

For example, if you had a nominal model of your system dynamics $\hat{f}$ with a known maximum error norm $e_{\max}(\mathbf{x}, \mathbf{u})$, then you could have one DDH based on this nominal model with $\mathcal{E}(\mathbf{x}, \mathbf{u}) = \mathrm{B}(\hat{f}(\mathbf{x}, \mathbf{u}), e_{\max}(\mathbf{x}, \mathbf{u}))$ and another from a dataset of observations. The combined DDH would then apply the upper bound derived from the dataset when close to the data, and the upper bound from the nominal model when farther from the data. When combined together in this way, the optimal control and the corresponding worst-case value of $\dot{\mathbf{x}}$ for the combined Hamiltonian will be the $\arg\min$ and $\arg\max$ of $\widehat{H}_{i^*}$ where i^* is the index of the minimum $\widehat{H}_i$.

A.3 Additional Lipschitz Constants and Uncertainty Sets

We consider four notions of sensitivity; the first is our running example. Each encodes a different amount of knowledge about f and gives a differently shaped uncertainty set. The uniform constant (i) treats all directions the same and gives a sphere. The input-wise vector (ii) allows a different sensitivity per state coordinate, shrinking the sphere when the state displacement lies in low-sensitivity coordinates. The output-wise vector (iii) bounds each velocity component separately, giving a rectangle. The sensitivity matrix (iv) combines both and is the tightest: entry L^{io}_{ij} bounds how much state j can change velocity component i, so zero entries in the Jacobian become zero-width directions of the set. For the double integrator only one entry is nonzero, which is why the rectangular variants win in Figure 6.

(i) a uniform Lipschitz constant L^x, satisfying

$$\|f(\mathbf{x},\mathbf{u}) - f(\mathbf{x}',\mathbf{u})\| \le L^x \|\mathbf{x}-\mathbf{x}'\|, \tag{7}$$

(ii) an input-element-wise constant vector $L^{x,\text{in}}$ satisfying

$$\|f(\mathbf{x},\mathbf{u}) - f(\mathbf{x}',\mathbf{u})\| \le \sum_{j=1}^{n_x} L^{x,\text{in}}_j |x_j - x'_j| \tag{8}$$

(iii) an output-element-wise constant vector $L^{x,\text{out}}$ satisfying for $i=1,\cdots,n_x$,

$$|f_i(\mathbf{x},\mathbf{u}) - f_i(\mathbf{x}',\mathbf{u})| \le L^{x,\text{out}}_i \|\mathbf{x}-\mathbf{x}'\| \tag{9}$$

(iv) an $n_x \times n_x$ sensitivity matrix $L^{x,\text{io}}$ satisfying for $i=1,\cdots,n_x$,

$$|f_i(\mathbf{x},\mathbf{u}) - f_i(\mathbf{x}',\mathbf{u})| \le \sum_{j=1}^{n_x} L^{\text{io}}_{ij} |x_j - x'_j| \tag{10}$$

for all $\mathbf{x},\mathbf{x}' \in \mathcal{X}, \mathbf{u} \in \mathcal{U}$. We define $L^u, L^{u,\text{in}}, L^{u,\text{out}}$, and $L^{u,\text{io}}$ analogously with respect to the control.

These lead to uncertainty sets:

$$\begin{aligned}
L^x &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u}^{(i)};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{B}\left(\mathbf{v}^{(i)}, L^x \left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|\right)\\
L^{x,\text{in}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u}^{(i)};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{B}\left(\mathbf{v}^{(i)}, (L^{\text{in}})^\top |\mathbf{x}-\mathbf{x}^{(i)}|\right)\\
L^{x,\text{out}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u}^{(i)};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{Rect}\left(\mathbf{v}^{(i)}, L^{x,\text{out}} \left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|\right)\\
L^{x,\text{io}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u}^{(i)};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{Rect}\left(\mathbf{v}^{(i)}, L^{x,\text{io}} |\mathbf{x}-\mathbf{x}^{(i)}|\right)
\end{aligned}$$

Searching over arbitrary controls, as in the state-control DDH , adds the corresponding control sensitivity. Writing $f(\mathbf{x},\mathbf{u}) - \mathbf{v}^{(i)} = \big(f(\mathbf{x},\mathbf{u}) - f(\mathbf{x}^{(i)},\mathbf{u})\big) + \big(f(\mathbf{x}^{(i)},\mathbf{u}) - f(\mathbf{x}^{(i)},\mathbf{u}^{(i)})\big)$ and applying the triangle inequality gives

$$\begin{aligned}
L^x, L^u &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{B}\left(\mathbf{v}^{(i)}, L^x \left\|\mathbf{x}-\mathbf{x}^{(i)}\right\| + L^u \left\|\mathbf{u}-\mathbf{u}^{(i)}\right\|\right)\\
L^{x,\text{in}}, L^{u,\text{in}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{B}\left(\mathbf{v}^{(i)}, (L^{\text{in}})^\top |\mathbf{x}-\mathbf{x}^{(i)}| + {L^{u,\text{in}}}^\top |\mathbf{u}-\mathbf{u}^{(i)}|\right)\\
L^{x,\text{out}}, L^{u,\text{out}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{Rect}\left(\mathbf{v}^{(i)}, L^{x,\text{out}} \left\|\mathbf{x}-\mathbf{x}^{(i)}\right\| + L^{u,\text{out}} \left\|\mathbf{u}-\mathbf{u}^{(i)}\right\|\right)\\
L^{x,\text{io}}, L^{u,\text{io}} &: \mathcal{E}^{(i)}(\mathbf{x},\mathbf{u};\mathbf{x}^{(i)},\mathbf{u}^{(i)}) = \text{Rect}\left(\mathbf{v}^{(i)}, L^{x,\text{io}} |\mathbf{x}-\mathbf{x}^{(i)}| + L^{u,\text{io}} |\mathbf{u}-\mathbf{u}^{(i)}|\right)
\end{aligned}$$

Setting $\mathbf{u} = \mathbf{u}^{(i)}$ recovers the sets above; the tightened rectangular variants in our experiments use the set in the final row.

A.4 LQR Baseline Training Details

Datasets. `noisy optimal` uses $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, 0.5^2 I)$. Rewards are the negated running cost times Δt, with the negated terminal cost added at the last step of each trajectory; actions are normalized to $[-1,1]$ for training and unnormalized for evaluation. The baselines maximize a discounted return

($\gamma = 0.995$) while evaluation uses the undiscounted finite-horizon cost; this mismatch costs little, since CQL comes within 1% of $J^\star$ on `optimal`.

Evaluation and epoch selection. Every method is scored by mean rollout cost from the same $1{,}000$ fixed test initial conditions under the true dynamics ($T = 2\,\text{s}$, $\Delta t = 0.01$). The DDH variants are deterministic given a dataset. For the offline-RL baselines we hold out 500 validation initial conditions (disjoint from the test set) and report test cost at the validation-best checkpoint at every dataset size.

CQL. Actor and twin critics are 3-hidden-layer $\times 256$ ReLU MLPs. Policy learning rate 10^{-4}, critic 3×10^{-4}, batch size 256, discount 0.995, target-network rate 0.005, deterministic backup. The entropy coefficient is auto-tuned to target entropy -2; the conservative penalty uses the Lagrangian formulation with target action gap 5.0 and 10 sampled random actions; no behavior-cloning warmup. 50 epochs ($50{,}000$ gradient steps), seed 10.

FQL. We use the official JAX implementation with its default architecture and optimizer (4-hidden-layer $\times 512$ networks, learning rate 3×10^{-4}, batch size 256, 10 flow steps), discount 0.995, $250{,}000$ gradient steps, seed 0. Following Park et al. (2025), the behavior-cloning coefficient α is tuned per dataset: we sweep $\alpha \in \{0.001, 0.01, 0.1, 1, 10\}$ and select on the validation initial conditions, yielding 10 (`optimal`), 1 (`noisy optimal`), 0.01 (`random trajectories`), and 0.001 (`uniform random`), decreasing monotonically with the action quality of the data. FQL trains for more gradient steps than CQL; both are reported at their validation-best checkpoint.

IQL. We use the CORL implementation, with its network depth and learning-rate schedule changed to match CQL: 3-hidden-layer $\times 256$ networks, expectile $\tau = 0.7$, inverse temperature $\beta = 3.0$, discount 0.995, 50 epochs of $1{,}000$ gradient steps, with actions normalized as for CQL. As for the other baselines, we report test cost at the checkpoint with the best cost on the validation initial conditions.

PPO and SAC. Stable-Baselines3 implementations trained on the true system (forward Euler, $\Delta t = 0.01$, $T = 2\,\text{s}$, actions in $[-4, 4]$, discount 0.995), one seed each. We evaluate the deterministic policy at saved environment-step checkpoints and report the one saved at the $20{,}000$-step milestone ($20{,}000$ steps for SAC; $43{,}008$ for PPO, whose training rounds each checkpoint interval up to whole $2{,}048$-step rollouts) and the final one ($50{,}000$ for SAC, $526{,}336$ for PPO), which is also each run's lowest cost.

Robustness checks. Rerunning CQL (seeds 10–12) and FQL (seeds 0–2) at three representative cells gives, on `random trajectories` at $n = 20{,}000$, 3.34 ± 0.01 for CQL and 4.77 ± 0.26 for FQL (mean $\pm$ SEM over seeds); the reported 13% and 61% gaps are the seed means. CQL is within 1% of $J^\star$ at $n = 500$ on `optimal` in every seed, and on `uniform random` at $n = 20{,}000$ CQL and FQL reach 3.16 ± 0.01 and 3.13 ± 0.01. The small-n uniform result is likewise not an artifact of the dataset draw: across five independent re-draws of the `uniform random` dataset, grid DDH (IO, $k = 10$) attains 3.03 ± 0.04 at $n = 50$ and 3.00 ± 0.04 at $n = 75$ (mean $\pm$ std over draws, vs $J^\star = 2.96$), within 4% of optimal in every draw.

A.5 DDH Variants on the LQR Problem

Table 1: All DDH variants at $n = 500$: average cost over the $1{,}000$ test initial conditions. Bold = best per column.

Variant	`optimal`	`noisy optimal`	`random traj.`	`uniform random`
Riccati Optimal	2.96	2.96	2.96	2.96
Sphere	4.67	3.50	7.06	3.74
Sphere + $k = 1$	4.67	3.49	6.15	3.45
Sphere + $k = 10$	4.67	3.49	6.09	3.22
IO	4.73	3.25	6.45	3.19
IO + $k = 1$	4.62	3.22	5.29	3.05
IO + $k = 10$	**4.62**	3.21	5.11	**2.97**
Neural DDH ($k = 10$)	4.93	**3.19**	**4.38**	2.98

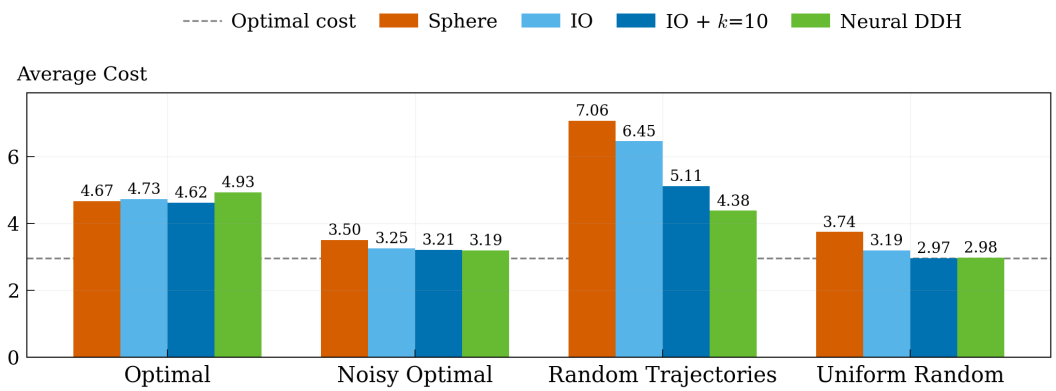


Figure 6: **Each component of the DDH contributes to controller quality.** Average rollout cost over the 1,000 test initial conditions at $n = 500$; dashed line: optimal reference. The rectangular (IO) uncertainty set and tightening each lower the cost on most datasets, and the neural DDH stays close to the grid solution. All variants are in Table 1.

Table 2 tests the minimal-information setting, in which the spherical DDH uses only the scalar constants $L^x = L^u = 1$ instead of the entrywise structure of A and B. On `random trajectories` it outperforms every offline RL baseline at every tested size from $n = 500$. On `uniform random` it does so up to $n = 300$ and at $n = 20{,}000$, while FQL is ahead from $n = 400$ and CQL from $n = 1{,}000$, through $n = 5{,}000$. The rectangular set closes most of the remaining gap to $J^\star$, for example 3.11 vs 5.11 on `uniform random` at $n = 50$.

Table 2: Minimal-information DDH: average cost over the 1,000 test initial conditions. Sphere variants use only scalar Lipschitz constants.

Dataset	n	Sphere	Sphere k=10	IO k=10	CQL	IQL	FQL
`random traj.`	500	7.06	6.09	5.11	17.05	13.36	18.29
	1,000	3.52	3.26	2.98	15.80	10.24	9.96
	20,000	3.11	3.03	2.96	3.35	3.74	4.76
`uniform random`	50	5.11	4.10	3.11	11.39	40.23	24.16
	500	3.74	3.22	2.97	7.14	14.16	3.38
	20,000	3.11	3.02	2.96	3.14	5.34	3.14

A.6 Sensitivity to Misspecification of the Lipschitz Bound

The DDH assumes a valid sensitivity bound, so we test what happens when it is overestimated. We scale $L^{x,\mathrm{io}}$ and $L^{u,\mathrm{io}}$ by 2 and 4, re-solve the value function (IO, $k = 10$, $n = 500$), and evaluate on the 1,000 test ICs (Table 3). Cost rises gradually: by 4 to 34% at $2\times$ and 10 to 61% at $4\times$. Even at $4\times$, the DDH still beats correctly configured CQL on the coverage-rich datasets (3.68 vs 7.14 on `uniform random`, 8.24 vs 17.05 on `random trajectories`). Underestimating the bound breaks the guarantee and is not tested here.

Table 3: Grid DDH (IO, $k = 10$, $n = 500$) with inflated Lipschitz bounds: average cost over the 1,000 test ICs.

Dataset	true L	$L \times 2$	$L \times 4$
`optimal`	4.62	5.19	5.53
`noisy optimal`	3.21	3.35	3.53
`random traj.`	5.11	6.84	8.24
`uniform random`	2.97	3.28	3.68

A.7 Lunar Lander Dynamics and Sensitivity Bounds

As described in Section 4.2, we base our modeling of the lunar lander dynamics on Inoue et al. (2026). The lunar lander has six states and three controls. The states consist of the x and y positions and velocities along with the angle and angular velocity. We will notate the full state as $\mathbf{z}$, with

$$\mathbf{z} = \begin{bmatrix} x \\ v_x \\ y \\ v_y \\ \theta \\ \omega \end{bmatrix}$$

and control

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

corresponding to the intensity of the main thruster, right thruster, and left thruster from 0 to 1. With a given mass m, angular inertia I, horizontal spacing of the side thruster from the center of mass δ_x, vertical spacing of the side thruster below the center of mass δ_y, side thruster angle γ, maximum main thrust T_p, maximum side thrust T_s, and gravitational constant g, we get the following dynamics:

$$\dot{\mathbf{z}} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{1}{m}(-u_1 T_p \sin(\theta) - u_2 T_s \cos(\gamma - \theta) + u_3 T_s \cos(\gamma + \theta)) \\ v_y \\ \frac{1}{m}(u_1 T_p \cos(\theta) + u_2 T_s \sin(\gamma - \theta) + u_3 T_s \sin(\gamma + \theta)) - g \\ \omega \\ \frac{1}{I}\left(u_2(\delta_x T_s \sin(\gamma) - \delta_y T_s \cos(\gamma)) + u_3(-\delta_x T_s \sin(\gamma) + \delta_y T_s \cos(\gamma))\right) \end{bmatrix} = f(\mathbf{z}, \mathbf{u})$$

This can be equivalently written in a control-affine form

$$f(\mathbf{z}, \mathbf{u}) = \dot{\mathbf{z}} = f(\mathbf{z}) + g(\mathbf{z})\mathbf{u}$$

where

$$f(\mathbf{z}) = \begin{bmatrix} v_x \\ 0 \\ v_y \\ -g \\ \omega \\ 0 \end{bmatrix}, \; g(\mathbf{z}) = \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{m}T_p \sin(\theta) & -\frac{1}{m}T_s \cos(\gamma - \theta) & \frac{1}{m}T_s \cos(\gamma + \theta) \\ 0 & 0 & 0 \\ \frac{1}{m}T_p \cos(\theta) & \frac{1}{m}T_s \sin(\gamma - \theta) & \frac{1}{m}T_s \sin(\gamma + \theta) \\ 0 & 0 & 0 \\ 0 & \frac{1}{I}(\delta_x T_s \sin(\gamma) - \delta_y T_s \cos(\gamma)) & \frac{1}{I}(-\delta_x T_s \sin(\gamma) + \delta_y T_s \cos(\gamma)) \end{bmatrix}$$

We assume that each control is bounded in $[0, 1]$. For our experiments in this paper we chose $m = 1$, $I = 1$, $\delta_x = 1$, $\delta_y = 1$, $\gamma = 0$, $T_p = 9$, $T_s = 3$ and $g = 1.62$.

Figure 7 shows the system.

A.7.1 Sensitivity

We can find the sensitivity of the dynamics by looking at the Jacobian of the dynamics function $f(\mathbf{z}, \mathbf{u})$ with respect to $\mathbf{z}$ and $\mathbf{u}$.

$$J_{\mathbf{z}} f(\mathbf{z}, \mathbf{u}) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{m}(u_1 T_p \cos(\theta) + u_2 T_s \sin(\gamma - \theta) + u_3 T_s \sin(\gamma + \theta)) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{m}(u_1 T_p \sin(\theta) + u_2 T_s \cos(\gamma - \theta) - u_3 T_s \cos(\gamma + \theta)) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$J_{\mathbf{u}} f(\mathbf{z}, \mathbf{u}) = g(\mathbf{z})$$

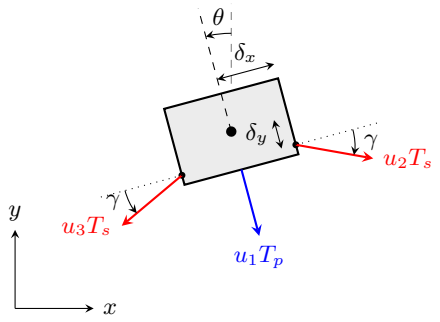


Figure 7: Lunar lander at a positive (counterclockwise) angle θ. Arrows show exhaust direction; each thruster pushes the lander opposite to its arrow. The side thrusters sit at $(\pm\delta_x, -\delta_y)$ from the center of mass, with exhaust tilted γ below the lander's horizontal axis, so $\gamma > 0$ gives the side thrust an upward component and $\gamma < 0$ a downward one. The drawing uses $\gamma = 25°$ for visibility; our experiments use $\gamma = 0$.

A simple bound for these is:

$$|J_{\mathbf{z}} f(\mathbf{z}, \mathbf{u})| \leq \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m}(T_p + 2T_s) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m}(T_p + 2T_s) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$|J_{\mathbf{u}} f(\mathbf{z}, \mathbf{u})| \leq \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{m}T_p & \frac{1}{m}T_s & \frac{1}{m}T_s \\ 0 & 0 & 0 \\ \frac{1}{m}T_p & \frac{1}{m}T_s & \frac{1}{m}T_s \\ 0 & 0 & 0 \\ 0 & \left|\frac{1}{I}(\delta_x T_s \sin(\gamma) - \delta_y T_s \cos(\gamma))\right| & \left|\frac{1}{I}(-\delta_x T_s \sin(\gamma) + \delta_y T_s \cos(\gamma))\right| \end{bmatrix}$$

With a bit more involved trigonometry, we can derive a tighter bound on $|J_{\mathbf{z}} f(\mathbf{z}, \mathbf{u})|$:

$$|J_{\mathbf{z}} f(\mathbf{z}, \mathbf{u})| \leq \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m}\left(T_p + T_s \max\left(1,\ 2|\sin(\gamma)|\right)\right) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m}\left(T_p + T_s \max\left(1,\ 2|\sin(\gamma)|\right)\right) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which we use in our experiments for $L^{x,\text{io}}$.

For the IO method, we can directly use the bounded sensitivity matrices. For the sphere-based method, we instead need the Lipschitz constant, an upper bound on which is given by the maximum singular value of the bounded matrix, $L^x = \sigma_{\max}(|J_{\mathbf{z}} f(\mathbf{z}, \mathbf{u})|)$ and $L^u = \sigma_{max}(|J_{\mathbf{u}} f(\mathbf{z}, \mathbf{u})|)$.

A.8 Lunar Lander Learned Value Functions

To visualize the value function learned by the DDH, we examine some 2-D slices of $V_\theta(\mathbf{x})$. We see qualitatively reasonable slices of the value function in Figure 8.

A.9 Lunar Lander Training Details (DDH and Baselines)

For details on the chosen configuration of the lunar lander environment, cost function, and shared parameters between our approaches, see Table 4.

Neural DDH. We train the neural DDH with Algorithm 1. For the first 5,000 gradient steps, the network is trained on the PDE loss $\mathcal{L}_{\text{PDE}}$ alone ($\lambda_2 = 0$). After that, we add the rollout supervision

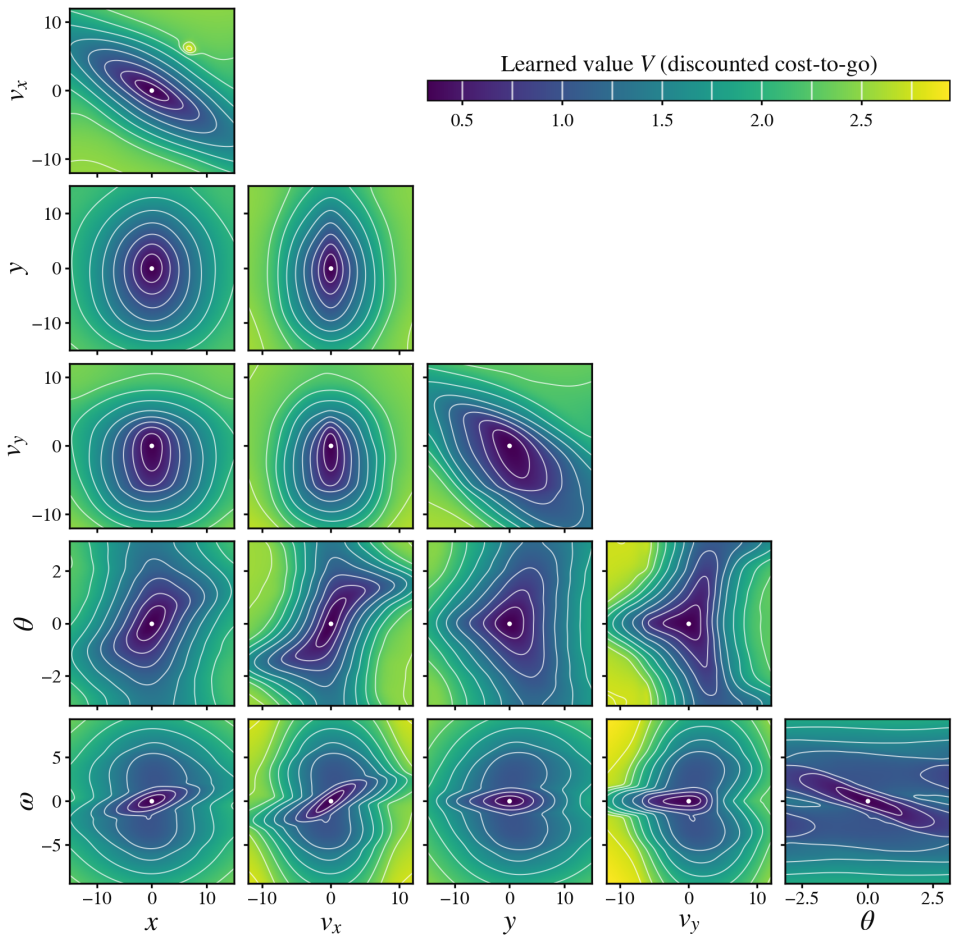


Figure 8: **The DDH learns a qualitatively reasonable value function.** These plots show 2-D slices of the learned value function from the DDH using the known $L^{x,\text{io}}$ and known single integrators. We see that they are relatively smooth, displaying few training artifacts.

loss $\mathcal{L}_{\text{rollout}}$ with weight $\lambda_2 = 3$. The supervision set $\mathcal{D}_{\text{sup}}$ holds 25,000 rollouts and is regenerated every 240 gradient steps, and the rollout horizon τ grows linearly from 0.5 to 4 s between steps 5,000 and 15,000. Rollout start states are sampled from $\mathcal{X}_{\text{sup}} = [-7.5, 7.5] \times [-6, 6] \times [-7.5, 7.5] \times [-6, 6] \times [-\pi, \pi] \times [-7.1, 7.1]$ over $(x, v_x, y, v_y, \theta, \omega)$. Note that about 4.8% of the dataset is contained within the supervision region. The lander uses the IO uncertainty set without tightening, so the controller applied at run time is the recorded control selected by the DDH. The uncertainty set for single integrator ("SI") variants of the DDH incorporates the knowledge that the dynamics of x, y, and θ are given by single integrators of other states, so there is no uncertainty in those dimensions. See Table 5 for details.

Model-based RL. For our model-based RL approach, we have two stages. We first learn a small fully connected neural network to represent an estimate of the dynamics f. We use the same dataset of 30k datapoints that the DDH was trained on. For details on the learning process see Table 6. We then apply a certainty-equivalence approach, where we solve a nonlinear MPC assuming our learned model is the true model. Details on the MPC problem (solver, parameters, etc.) can be found in Table 7.

Model-free RL. We used the original authors' CQL and FQL implementations. For details on their training, see Table 9. We also used existing implementations of PPO and SAC from the Stable-Baselines3 repository. For details on their training, see Table 8. The reset distribution for PPO and SAC was set to match the supervision region for the DDH.

Table 4: Shared problem parameters.

Parameter	Value
Mass, inertia	$m = 1, I = 1$
Gravity	$g = 1.62$
Thrust scalings	$T_p = 9$ (main), $T_s = 3$ (side)
Lever arms, side-thruster angle	$\delta_x = \delta_y = 1, 0^\circ$
Hover	$\mathbf{x}_{\text{eq}} = 0$, $\mathbf{u}_0 = (mg/T_p, 0, 0) = (0.18, 0, 0)$
Q	$\text{diag}(0.01, 0.01, 0.01, 0.01, 0.1, 0.02)$
R	$0.3\, I_3$
Discount rate γ	$0.33\,\text{s}^{-1}$ (per step $e^{-\gamma \Delta t} = 0.993422$)
Evaluation horizon N	757 steps (15.14 s)
State box $\mathcal{X}$	$x, y \in [-15, 15]$, $v_x, v_y \in [-12, 12]$, $\theta \in [-\pi, \pi]$ (periodic), $\omega \in [-3\pi, 3\pi]$
Supervision region $\mathcal{X}_{\text{sup}}$	$x, y \in [-7.5, 7.5]$, $v_x, v_y \in [-6, 6]$, $\theta \in [-\pi, \pi]$, $\omega \in [-7.1, 7.1]$
Test set	20,000 $\mathbf{x}_0 \sim \text{Unif}(\mathcal{X}_{\text{sup}})$ (+20,000 in $\mathcal{X}$)
Tuning set	5,000 $\mathbf{x}_0$ per region, disjoint from test set

Table 5: Neural DDH hyperparameters

Parameter	Value
Value network	
Architecture	MLP $7 \to 512 \to 512 \to 512 \to 512 \to 1$ (792,577 parameters)
Activation	Softplus ($\beta = 10$)
Input features	state scaled to $[-1, 1]$ over $\mathcal{X}$; $\theta \mapsto (\sin\theta, \cos\theta)$
Optimization	
Optimizer	Adam, $\beta = (0.9, 0.999)$
Learning rate	2×10^{-4}, constant
Training steps	40,000
Pretraining, gradient clipping	none, none
PDE points	25,000 per step, resampled, $\text{Unif}(\mathcal{X})$
PDE loss weight	1
Supervised loss weight	3
Supervision targets $\hat{V}$	
Rollouts per refresh	25,000, $\mathbf{x}_0 \sim \text{Unif}(\mathcal{X}_{\text{sup}})$
Refresh interval	240 steps
Supervision start	step 5,000
Rollout horizon τ	$\min(4,\ 0.5 + 3.5 \times 10^{-4}(k - 5000))$ s
Rollout Δt	0.02 s
Horizon bootstrap	$e^{-\gamma\tau} V_\theta(\mathbf{x}_\tau)$
Exit (leaves $\mathcal{X}$)	terminate; bootstrap value $1/\gamma = 3.0303$
Hardware	1× NVIDIA RTX 5090
Wall time	1 h 11 min

Table 6: System identification: learned dynamics model.

Parameter	Value
Architecture	MLP $10 \to 32 \to 32 \to 6$, tanh (1,606 parameters)
Inputs	$(p_x, v_x, p_y, v_y, \sin\theta, \cos\theta, \omega, u_1, u_2, u_3)$
Loss	MSE, normalized output space
Optimizer	Adam
Learning rate	3×10^{-3}, cosine to 3×10^{-6}
Batch size	512
Epochs (steps)	4,000 (236,000)
Seed	0
Wall time (CPU)	263 s

Table 7: MPC hyperparameters.

Parameter	MPC	MPC, 15 FPS
Prediction model	Euler step, $\Delta t = 0.02$	
Variables	States and controls	
Horizon H	4 s (200 steps)	
Stage cost	$\Delta t\, e^{-\gamma j \Delta t} L(\mathbf{x}_j, \mathbf{u}_j)$	
Terminal cost	$w_T L(\mathbf{x}_H, \mathbf{u}_0)$, $w_T = \sum_{j \geq H} \Delta t\, e^{-\gamma j \Delta t} = 0.8122$	
Constraints	$\mathbf{u} \in [0, 1]^3$; no state constraints	
Solver	IPOPT, MUMPS, exact Hessian, tol 10^{-8}, CasADi 3.8 (JIT)	
Initial guesses ($t = 0$)	hover + 5 winding branches $n \in \{-2, \ldots, 2\}$ (11 solves)	
Re-plan interval	25 steps (0.5 s)	3 steps (60 ms)
IPOPT iteration cap	50	6
Warm start	shifted previous plan	shifted primal–dual, $\mu_{\text{init}} = 10^{-4}$
Median re-plan time[†]	360 ms	57 ms

[†] one pinned core, AMD Threadripper PRO 7985WX; $t = 0$ plan excluded.

Table 8: PPO and SAC hyperparameters (Stable-Baselines3 2.9.0).

Parameter	PPO	SAC
Environment		
Observation	7-D, $\theta \mapsto (\sin\theta, \cos\theta)$	7-D, $\theta \mapsto (\sin\theta, \cos\theta)$
Reward Scaling	—	10
Reward Normalization	running return std	none
Reset distribution	$\mathrm{Unif}(\mathcal{X}_{\text{sup}})$	$\mathrm{Unif}(\mathcal{X}_{\text{sup}})$
Episode length (train)	152 steps (3.04 s)	757 steps (15.14 s)
Termination at $\partial\mathcal{X}$	none	none
Networks		
Actor	$7 \to 256 \to 256 \to 3$, tanh	$7 \to 512 \to 512 \to 3$, ReLU
Policy distribution	Gaussian, state-independent σ	tanh-Gaussian
Critic	V: $7 \to 256 \to 256 \to 1$, tanh	$2 \times Q$: $10 \to 512 \to 512 \to 1$, ReLU
Optimization		
Optimizer	Adam ($\epsilon = 10^{-5}$)	Adam ($\epsilon = 10^{-8}$)
Learning rate	3×10^{-4}, constant	3×10^{-4}, constant
Rollout steps per env	1024	—
Replay buffer	—	10^6
Minibatch size	1024	256
Epochs per update	10	—
GAE λ	0.95	—
Clip range	0.2	—
Entropy	coefficient 0.005	automatic, target -3
Value loss coefficient	0.5	—
Max gradient norm	0.5	—
Target smoothing τ	—	0.005
Learning starts	—	10,000
Runtime		
Hardware	CPU (1 thread)	CPU (1 thread)
Wall time (full budget)	4.6 h	16.7 h

Table 9: CQL and FQL hyperparameters.

Parameter	CQL	FQL
Implementation	rlkit CQL (reference)	official FQL (JAX)
Variant	CQL($\mathcal{H}$), SAC backbone	flow BC policy + one-step policy
Observation	6-D, θ wrapped	7-D, $\theta \mapsto (\sin\theta, \cos\theta)$
Reward scale	10	1
Discount (per step)	0.993422	0.993422
Actor	$6 \to 256^{\times 3} \to 3$, ReLU, tanh-Gaussian	$7 \to 512^{\times 4} \to 3$, GELU (flow and one-step)
Critic	$2 \times Q$, $256^{\times 3}$, ReLU	$2 \times Q$, $512^{\times 4}$, GELU, LayerNorm
Target Q aggregation	min	min
Conservative weight α_{CQL}	0.03 (fixed, no Lagrange)	—
CQL random actions temp.	10, 1.0	—
Max-backup	no	—
BC warm-up steps	0	—
Distillation coefficient α	—	0.001
Flow steps	—	10
Entropy	automatic, target -3	—
Learning rate	policy 10^{-4}, Q 3×10^{-4}	3×10^{-4}
Batch size	256	256
Target smoothing τ	0.005	0.005
Gradient steps	100,000	125,000
Evaluation policy	$\tanh(\mu)$	one-step policy, sampled noise
Hardware	1× RTX 5090	1× RTX 5090
Wall time per run	2.3–2.6 h	6–7 min