LBPO: Lagrangian-Based Policy Optimization

Mantas Birskus

Master of Science

School of Informatics

University of Edinburgh

2024

Abstract

Physics-Informed Neural Networks (PINNs) are revolutionizing scientific machine learning by imposing mathematical constraints on data-driven predictions. This has sparked interest in time-sensitive Physics-Informed Reinforcement Learning (PIRL). While much research has focused on incorporating known system dynamics, emerging studies are beginning to address how neural network architectures can be constrained to satisfy physical laws. In this work, we introduce a novel Lagrangian-Based Policy Optimization (LBPO) algorithm that integrates principles from Lagrangian mechanics with deep neural networks. Our results demonstrate that LBPO excels in classical dynamics and control systems. The algorithm not only adheres to physical laws, such as energy conservation, but also enhances policy optimization by uniquely sharing gradient information between the world and agent models. Our code is available at https://github.com/mantasu/lbpo.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mantas Birskus*)

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Michael Herrmann, for his continuous guidance and invaluable feedback in shaping my project. I am also thankful to my friends and relatives for their unwavering moral support, which has helped me stay motivated throughout this journey. Additionally, I extend my appreciation to my parents for their financial assistance, enabling me to conduct experiments on a high-end machine. Lastly, I am grateful to the UoE Financial Support team for their aid towards my living costs, which has significantly reduced my stress and allowed me to focus on my studies.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The application of reinforcement learning (RL) in the realm of dynamics and control systems has been a focal point of research in both academic and industrial settings [1, 2, 3]. For example, Tiumentsev and Zarubin [4] show that incorporating reinforcement learning in aircraft control policy is crucial when the properties of the dynamical object are unknown or incomplete. In recent years, physics-informed machine learning (PIML) has gained traction which allows data-driven models to integrate physics constraints that humans define or to discern physical laws directly from data [5, 6, 7]. When applied to dynamical systems, these constraints usually take the form of differential equations with respect to time. A classic example of this is the equations that describe the motion of a pendulum [8] (Equation 1.1 defines its differential motion). The integration of these physical laws ensures that the predictions made by the data-driven model align with established physical principles.

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0 \tag{1.1}$$

As such, physics-informed reinforcement learning (PIRL) has become an increasingly studied field with numerous works being published over the last few years [9]. Studies have shown that incorporating physics knowledge can help solve some major practical problems, such as sample deficiency [10], high dimensionality [11], and safety risks [12].

1.2 Research Focus and Contributions

Most works have leveraged physics knowledge by identifying pertinent features [13, 14], constraining the state or action space [15, 16], or adjusting the reward function [17, 18]. However, these methodologies are only applicable when the theory of the dynamical system is already known. This is often not the case as the physics information can be incomplete or absent, which becomes particularly apparent when transitioning from idealized simulations to real-world scenarios [19].

Therefore, We shift our focus to neural ordinary differential equations (neural ODEs) [20], more specifically, to problems where system dynamics are governed by differential equations - a foundational concept in physics [21]. This approach enables us to design neural networks that inherently respect physical laws. For example, models like Hamiltonian [22, 12] and Lagrangian [23, 24] neural networks naturally incorporate energy-preserving constraints, embedding conservation principles directly into the learning process. Similarly, architectures such as Deep Operator Networks (DeepONets) [25, 26] and Neural Implicit Flow [27, 28] models are crafted to operate over continuous functions, capturing the underlying consistency within differential fields. Additionally, symbolic regression techniques like PySR [29, 30] and sparse identification of nonlinear dynamics (SINDy) [31, 32] aim to discover interpretable symbolic expressions that represent the governing equations of a system.

Our work extends the integration of Lagrangian mechanics into model-based reinforcement learning (MBRL) [33]. Specifically, we contribute the following:

- **Lagrangian Neural Networks for Control**: We extend the work by Cranmer et al. [23] to control tasks, showing that Lagrangian-based models capture system dynamics, enhance sample efficiency, and stabilize policy learning.
- **Gradient-Enhanced PPO**: We develop a variant of proximal policy optimization (PPO) [34] that leverages gradient information from the physics-informed model, speeding up training and improving convergence.
- **Athletes Software Package**: We introduce a lightweight, efficient Python package for reinforcement learning, offering unique features like nested agents, designed for both beginners and advanced users.

The relative complexity of the project is also discussed in section A.1.

1.3 Scope and Outline

The scope of the project is limited to single-agent reinforcement learning for dynamics and control problems that do not involve contact and energy loss, such as Gymnasium's [35] Cartpole [36] and Pendulum [37] environments. This is to ensure a thorough analysis can be conducted within a fixed field without expanding into too many directions for the limited time that was given.

Here is a breakdown of the upcoming sections:

- **Literature Review** (section 2.1) - we discuss the fundamentals of reinforcement learning and move to related works in physics-informed reinforcement learning
- **Theoretical Details** (section 2.2) - we introduce to proximal policy optimization algorithm and lagrangian neural network, two concepts our work is based on
- **Lagrnagian Neural Networks for Control** (section 3.1) - we present a novel architecture for lagrangian neural networks extending them to handle control variables
- **Gradient Boosted Proximal Policy Optimization** (section 3.2) - we define the algorithms that lagrangian-based policy optimization depends on
- **Athletes Software Package** (section 3.3) - we present a supplementary Python package that was built alongside the conducted research
- **Setup** (section 4.1) - we discuss the environment and training setup for experiments
- **Individual Components** (section 4.2) - we experiment with individual architecture components of LBPO algorithm
- **Algorithm Performance** (section 4.3) - we compare our algorithm with other existing ones
- **Code Efficiency** (section 4.4) - we showcase our package efficiency and present autograd feature
- **Concludion** (chapter 5) - finally conclude our work and discuss limitations as well as future perspectives

Chapter 2

Background

2.1 Literature Review

2.1.1 Reinforcement Learning

2.1.1.1 Fundamentals

"Reinforcement Learning: An Introduction" (R.S. Sutton and A.G. Barto) is often regarded as a seminal handbook in reinforcement learning [39, 40, 41], offering a comprehensive theoretical framework. It essentially operates on the principles of a Markov Decision Process (MDP) [42] (see Figure 2.1), where an agent interactively learns which action $a_i^{(t)}$ to take at which state $s_j^{(t)}$ through a sequence of timesteps $t \in \{0, \cdots, T\}$ to maximize a cumulative return of goal-based rewards $\{r^{(1)}, \cdots, r^{(T)}\}$. In other words, the agent learns an optimal policy $\pi^*(a^{(t)}|s^{(t)})$ from trajectories $\{(s^{(t)}, a^{(t)}, r^{(t+1)})\}_{t=0}^{T_n}$ acquired by interacting with the environment through $n = \{1, \cdots, N\}$ trials, or episodes.

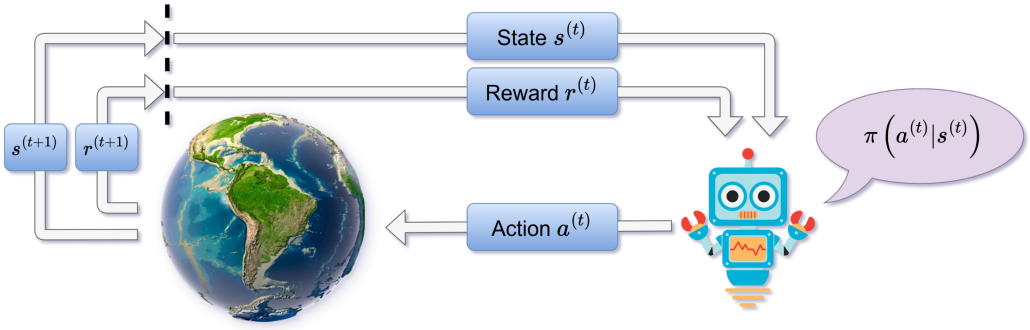
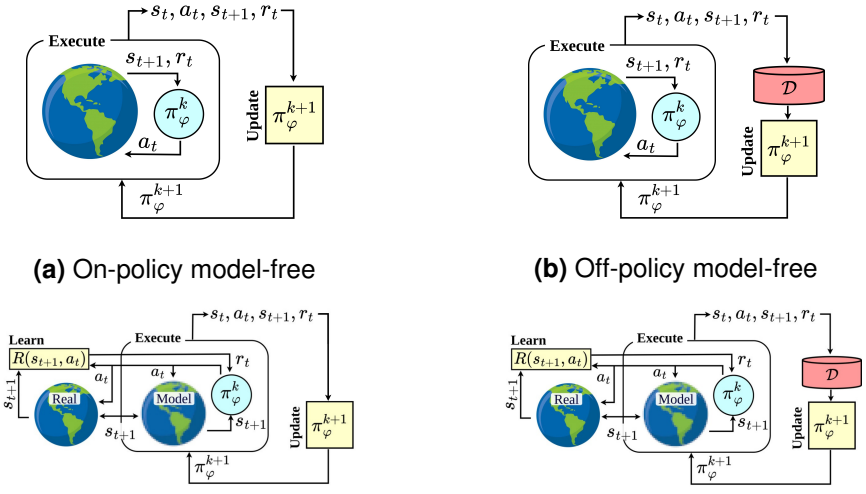


Figure 2.1: The agent–environment interaction in a Markov Decision Process.

Traditionally, reinforcement learning (RL) is split into model-free and model-based and can be further divided into on-policy and off-policy (see Figure 2.2):

- **Model-free** algorithms optimize action strategies based on direct environment interactions [43]. For instance, Saxena et al. [44] show how a simulated car can learn to safely merge into dense traffic without explicit modeling of other vehicles' movements.
- **Model-based** algorithms, on the other hand, learn a model of the environment's dynamics, which is used to plan and make decisions [33]. Nagabandi et al. [45] show how a learned dynamics neural network can significantly reduce the sample size required to reach optimal policies for locomotion tasks.
- **On-policy** RL involves learning the same policy for both exploration and exploitation [46], as exemplified by Li et al. [47] who use Proximal Policy Optimization (PPO) [34] for bipedal robot control.
- **Off-policy** RL, conversely, learns a separate policy for exploration [48], as demonstrated by Gu et al. [49] who adapt Normalized Advantage Functions (NAF) [50] to teach a robot to open a door.



(a) On-policy model-free **(b)** Off-policy model-free

(c) On-policy model-based **(d)** Off-policy model-based

Figure 2.2: Reinforcement learning types. Illustrations are taken from and more details can be found at [9].

The methodology proposed in this project employs model-based approach because physics laws can be naturally incorporated when learning a model of the environment. For simplicity and memory efficiency, we also constrain ourselves to on-policy learning.

2.1.1.2 Dynamics and Control

Optimal control theory [51] has been applied to a variety of control problems, including metal cutting processes [52], rotational translational actuators [53], missile autopilot design [54], and has been a subject of extensive research for decades [55, 56, 57]. Recently, the success of reinforcement learning [58, 59] has also extended to these control problems, such as in autonomous driving (2.3), where an agent must determine appropriate steering rates and torque to maintain course [60], and in active flow control, where an agent modulates mass flow rates to stabilize fluid flow around an object and minimize resistance [61]. As Recht [62] explains, control theory involves designing complex actions based on well-defined models, whereas reinforcement learning often generates intricate predictions using only data, without relying on explicit models. This highlights the strength of RL, particularly its applicability to control problems in noisy environments where exact solutions would be too sensitive and adaptation is necessary [63]; in scenarios where the environment is only partially observed [64]; or when deriving an exact solution is challenging and approximations are required [65].

Figure 2.3: Autonomous Driving task (for illustrative purposes)[1].

While traditional methods dominated for many years [66, 67], the advent of deep learning [68, 69] has shifted the focus towards deep reinforcement learning (DRL) [70, 71]. DRL is particularly effective for continuous state and action spaces, such as in chemical reaction optimization, where an agent must apply precise temperature or pressure controls [72]. However, these models are often black boxes, producing excellent results but are difficult to understand [73]. In the context of model-based RL, solutions to some problems, like DreamerV3's algorithm for finding diamonds in Minecraft [74], are challenging to explain due to the recurrent nature of the world models used [75, 76, 77]. These models typically maintain internal states over time, making their behavior harder to interpret. On the other hand, control problems are often simpler because the environmental model usually depends only on the previous state and action, provided they are sufficiently represented [78, 79, 80]. This simplicity in control problems motivates the development of more explainable and physics-reliable surrogate models, leading to more insightful results and analyses.

[1] https://www.flickr.com/photos/56682936@N03/35016564364 (CC BY 2.0)

2.1.2 Incorporating Physics

2.1.2.1 Physics-Informed Machine Learning

Recently, physics-informed machine learning (PIML) has emerged [81]. This approach imposes physics constraints on data-driven models, allowing handling problems with less data. It has been successfully applied to a variety of mechanical tasks, including heat transfer [82], power systems [83], and subsurface transport [84].

A physics-informed neural network (PINN) is typically designed for forward problems, where the network's output directly solves the defining partial differential equations (PDEs) of the predicted dynamics [85]. Lu et al. [86] relate network architectures to differential equations. For instance, the simplest architecture resembles a Residual Network (ResNet) [87], where the prediction of the next state follows $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + f_\theta(\mathbf{x}^{(t)})$, with $f_\theta(\cdot)$ as the network's prediction of the derivative $\frac{d\mathbf{x}}{dt}$, a black-box non-linear function parameterized by θ. Such iterative updates are essentially Euler discretization over continuous time [88]. Neural Ordinary Differential Equations (Neural ODEs) [20] improve this method by allowing unevenly spaced timesteps, extending applicability to continuous dynamics. This method computes $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \int_t^{t+1} f_\theta(\mathbf{x}^{(\tau)})d\tau$ using any ODE solver, such as Adams/BDF [89] or 8th order Runge-Kutta [90]. A more advanced architecture, Hamiltonian Neural Networks [22], leverages the Hamiltonian structure of many dynamical systems [91, 92, 93], reducing the data required for training and facilitating the learning of energy conservation laws. These networks predict a Hamiltonian $\mathcal{H}_\theta(\mathbf{q}, \mathbf{p})$, imposing constraints $\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{q}}$, where $\mathbf{q}$ and $\mathbf{p}$ are position and momentum vectors, respectively. A similar but more general approach, Lagrangian Neural Networks [23], uses Euler-Lagrange principles [94] to constrain the system. This approach is used in this paper and discussed further in 2.2.2.

Substantial research has also been conducted on inverse problems to identify PDEs from data [6, 95]. For instance, Xu and Zhang [96] combine PINN with a Genetic Algorithm [97] to recover PDEs with high-order derivatives, such as Korteweg de Vries [98] and Burger's [99] equations. Cranmer et al. [100] use Graph Neural Networks [101] to discover new cosmological laws from N-body data [102]. A leading and widely applied architecture, as claimed by Kaptanoglu et al. [103], is Sparse Identification of Nonlinear Dynamics (SINDy) [5], a kind of symbolic regression [104], which uses sparse linear regression [105] to identify which terminals ($\frac{1}{3}$, π) and non-terminals ($\div$, sin) compose a dynamical equation. We utilize inverse techniques to enhance explainability.

2.1.2.2 Physics-Informed Reinforcement Learning

Physics-informed reinforcement learning (PIRL) has become increasingly studied [9]. Chen, Liu, and Di [106] incorporate a PINN into policy learning to solve Mean Field Games (MFG) [107]. Similarly, Shilova et al. [108] use a PINN to approximate value functions for several dynamics problems in continuous-time reinforcement learning. One significant area where PINNs have been actively applied is safe reinforcement learning, where ensuring the safety and reliability of the learned policies is crucial [109]. Wang and Nakahira [110] provide a physics-informed framework to estimate risk probability for safe control systems, while Zhao, Wang, and Yue [111] present a framework for learning a safety controller that satisfies predefined boundary regions.

Other relevant works focus on model-based reinforcement learning [112, 113], where researchers develop physics-informed surrogate models to solve various simulated dynamic problems, such as Cartpole [114] and Acrobot [58]. These studies highlight that embedding physics information reduces environment model bias, even when physics laws are only partially known. Some approaches go further, using fully differentiable simulators to obtain analytical gradients that are leveraged in policy learning [115, 116, 117]. However, these methods are limited by the knowledge of the governing physics equations. To address this limitation, SINDy-RL [118, 119] learns a symbolic model of world dynamics, while deep symbolic optimization [120, 121, 122] shows that return-maximizing equations can be generated even for policies. Nonetheless, the complexity of symbolic solutions can quickly escalate [123] and undermine their explainability, which is why our method does not completely rely on them.

It is also worth noting that similar research areas exist, such as inverse reinforcement learning (IRL) [124], where the agent seeks to identify the reward function from the underlying Markov Decision Process (MDP). Other works use reinforcement learning itself to uncover PDE expressions or their coefficients [125, 126], enhancing symbolic regression, although these results are not embedded into any interactive learning process.

Overall, the convergence of physics and reinforcement learning through PIRL offers promising avenues for developing more robust, reliable, and interpretable AI systems. The incorporation of physical laws into the learning process not only enhances the performance of RL algorithms but also ensures that the learned policies adhere to real-world constraints and principles, paving the way for safer and more efficient AI applications. This alignment with PIRL principles is a key motivation for our work.

2.2 Theoretical Details

2.2.1 Proximal Policy Optimization

Our work relies on Proximal Policy Optimization (PPO) [34] which is a common choice in PIRL [13, 127, 128]. Due to simplicity, it serves as a great baseline which is adopted by our method. It essentially optimizes a surrogate objective loss $\mathcal{L}$ which is based on ratio r between new and old policy and on advantages $\hat{A}$ which asses how much better it is to take the new policy actions. We use clip objective, the most common variant [129]:

$$\mathcal{L}^{\text{CLIP}} := \mathop{\mathbb{E}}_{a,s\sim\pi_{\text{old}}} \left[\min\left(r(\theta)\hat{A}^{\pi_{\text{old}}}(a,s), \text{clip}\left(r(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}^{\pi_{\text{old}}}(a,s)\right)\right] \tag{2.1}$$

Where $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}$ is the probability ratio and $\hat{A}^{\pi_{\text{old}}}(\cdot)$ is advantage function, for instance, Generalized Advantage Estimation (GAE) [130]. PPO algorithm involves critic and actor updates; the simplified ones are shown below[2] (see also section B.2):

Algorithm 1 Critic update

Require: O, ν_ϕ, N, B ▷ Attributes - see subsection B.2.4

procedure CRITIC_UPDATE($\mathcal{D}$)
  $\mathcal{DR} \leftarrow$ concat($\mathcal{D}$, compute_returns($\mathcal{D}$)) ▷ Algorithm 8
  for $n = 1, \cdots, N$ **do**
    $\{(o_b, \text{R}_b) \leftarrow \text{sample}(\mathcal{DR})\}_{b=1}^{B}$ ▷ Sample batch
    $\mathcal{L} \leftarrow \frac{1}{B}\sum_{b=1}^{B}(\text{R}_b - \nu_\phi(o_b))^2$ ▷ MSE [131]
    $\nu_\phi \leftarrow O.\text{optimize}(\nu_\phi, \mathcal{L})$
  return ν_ϕ

Algorithm 2 Proximal actor update

Require: O, π_θ, ν_ϕ, N, B, ε ▷ Attributes - see subsection B.2.4

procedure PROXIMAL_UPDATE($\mathcal{D}$)
  $\mathcal{DA} \leftarrow$ concat($\mathcal{D}$, compute_advantages($\mathcal{D}, \nu_\phi$)) ▷ Algorithm 9
  $\pi_{\text{old}} \leftarrow \text{clone}(\pi_\theta)$ ▷ Save old π_θ
  for $n = 1, \cdots, N$ **do**
    $\{(o_b, a_b, \text{A}_b) \leftarrow \text{sample}(\mathcal{DA})\}_{b=1}^{B}$ ▷ Sample batch
    $\mathcal{L} \leftarrow \frac{1}{B}\sum_{b=1}^{B}\min(\text{A}_b\frac{\pi_\theta(a_b|o_b)}{\pi_{\text{old}}(a_b|o_b)}, \text{A}_b\text{clip}(\frac{\pi_\theta(a_b|o_b)}{\pi_{\text{old}}(a_b|o_b)}, 1\pm\varepsilon))$ ▷ Equation 2.2.1
    $\pi_\theta \leftarrow O.\text{optimize}(\pi_\theta, \mathcal{L})$
  return π_θ

[2] Normally, the two are merged into one, however, for ease of understanding, we keep them separate.

2.2.2 Lagrangian Neural Networks

Lagrangian Neural Networks (LNN) were introduced by Cranmer et al. [23]. Much like Hamiltonian dynamics, many physical systems can be expressed using Lagrangian formalism, including thermo- [132], fluid [133], and relativistic [134] dynamics. Lagrangian mechanics essentially states that the system behaves on the principle of "least action" [135], which is used to define Euler-Lagrange constraint:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} = \frac{\partial \mathcal{L}}{\partial q_i} \tag{2.2}$$

Such that $\mathcal{L} \equiv T - V$, where T and V are kinetic and potential energies, respectively. This requires the state of the system to be defined as a vector containing positions and their velocities $\mathbf{x} = \left(q_1, \cdots, q_N, \dot{q}_{N+1}, \cdots, \dot{q}_{2N}\right)^\top$, which is common practice in dynamical systems literature [136]. $\mathcal{L}$ can be learned via a neural network. From here, acceleration can be computed as follows:

$$\ddot{\mathbf{q}} = \left(\nabla_{\dot{\mathbf{q}}} \nabla_{\dot{\mathbf{q}}}^\top \mathcal{L}\right)^{-1} \left[\nabla_{\mathbf{q}} \mathcal{L} - (\nabla_{\mathbf{q}} \nabla_{\dot{\mathbf{q}}} \mathcal{L})\dot{\mathbf{q}}\right] \tag{2.3}$$

Where ∇ represents vectorized derivatives, i.e., $(\nabla_{\dot{\mathbf{q}}})_i = \frac{\partial}{\partial \dot{q}_i}$. Having $\mathbf{q}^{(t)}$, $\dot{\mathbf{q}}^{(t)}$, and $\ddot{\mathbf{q}}^{(t)}$, equips us with knowledge of how the system evolves and allows us to compute the next state $\mathbf{q}^{(t+1)}$, $\dot{\mathbf{q}}^{(t+1)}$ by solving ODE.

LNNs generalize over Hamiltonian Neural Networks (HNNs) [22] by allowing arbitrary coordinates, and over Deep Lagrangian Networks (DeLaNs) [24] by not constraining the type of dynamical system. They are a common choice for many applications, for example, video prediction [137], robotic modeling [138], and identification of mechanical system parameters [139].

As noted by Cranmer et al. [23] and Roehrl et al. [139], LNNs learn exact energy conservation laws and arbitrary lagrangians, are more performant, universal, and interpretable, therefore, more suitable for physical systems, compared to regular neural networks. It should also be emphasized that they are designed to be differentiable, i.e., their forward pass requires computing Jacobians and Hessians to infer the next state. Differentiating them w.r.t. model parameters (during the update) enforces a "physics-informed" gradient flow throughout the network, enhancing bi-directional connectivity between layers. Due to these benefits and because we only focus on classic control problems, LNNs are a natural baseline choice.

Chapter 3

Methodology

3.1 Lagrangian Neural Networks for Control

3.1.1 Lagrangian Neural Networks as World Models

Lagrangian Neural Networks (LNNs) can serve as physics-informed surrogate models in model-based reinforcement learning, offering a more accurate representation of system dynamics. Typically, a world model predicts the next state $\mathbf{x}^{(t+1)}$ from the current state $\mathbf{x}^{(t)}$ (including $\mathbf{q}$ and $\dot{\mathbf{q}}$) and action $\mathbf{a}^{(t)}$. In the original formulation [23], given $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\mathcal{L}$, one can compute $\ddot{\mathbf{q}}$. However, the challenge becomes incorporating action/control variables, such as external forces, as they can significantly alter dynamics.

Attempts to use LNNs for reinforcement learning tasks [140, 141, 142] often rely on domain-specific knowledge. For instance, Lutter, Listmann, and Peters [142] impose specific speed and torque balance equations on an electric engine, while Ramesh and Ravindran [143] use a Lagrangian-based equation specific to rigid body motion.

We propose several very simple methods to incorporate the control variable in a way that generalizes across domains:

- **Concatenation**. The simplest approach is to concatenate the state and control values. Specifically, the Lagrangian network can take as input

$$\text{concatenate}(\mathbf{x}, \mathbf{a}) = \left(q_1, \cdots, q_N, \dot{q}_{N+1}, \cdots, \dot{q}_{2N}, a_1, \cdots, a_M\right)^\top.$$

  For its efficiency and simplicity, we adopt this method, building on it with additional techniques discussed in the following subsections.

- **Outer Product**. Another method is to compute the outer product

$$\text{flatten}(\mathbf{x}\mathbf{a}^\top) = \left(q_1 a_1, \cdots, \dot{q}_{2N} a_1, \cdots, q_1 a_M, \cdots, \dot{q}_{2N} a_M\right)^\top,$$

  multiplying each state by each control value. The original values can also be concatenated with the result. This method is particularly effective for discrete action spaces, with a one-hot action representing a selective mapping. However, the drawback is that the vector size increases quadratically with the number of control and state variables.

- **Double Encoder**. A more sophisticated approach involves using separate encoders for the action and state vectors. Each encoder extracts features that can be concatenated, added, or multiplied before being fed into the Lagrangian network. This method is well-suited for high-dimensional spaces, though it introduces additional parameters.

The methods discussed can generalize across any dynamic system without requiring prior knowledge. While one might argue that Lagrangian mechanics imposes constraints, requiring an understanding of which problems support this formalism, it's important to note that the "least action" principle is a fundamental concept in nature, applicable from Newtonian to quantum mechanics [144]. As long as the state of a system can be expressed in terms of $\mathbf{q}$ and $\dot{\mathbf{q}}$, this principle holds.

If the system is known to depend on specific parameters, like the gravitational constant, existing methods might be preferable. However, our approach is better suited for situations where the parameters are not fully known, can vary, or when the system dynamics are being explored.

Toy problems like Gymnasium's [35] Cartpole and Pendulum do not involve energy loss (e.g., no friction), thus maintaining energy conservation. In contrast, real-world problems are subject to noise and unknown factors. To address this, we propose using Generalized LNNs [145], where a non-conservation term is predicted by an additional neural network, taking the same input as the Lagrangian network. While control actions explicitly change the system's total energy, they don't violate the conservation principle; the Lagrangian network accounts for these changes when predicting the Lagrangian, demonstrating the flexibility of neural networks in handling such complexities.

3.1.2 Incorporating Actor Decision Process

A simple concatenation of state and action variables works because the predicted Lagrangian is differentiated with respect to the "observation part" of the concatenated vector when computing acceleration based on Equation 2.3. In reinforcement learning, actions are typically predicted by a separate neural network (e.g., the actor network [146]), which uses the same observation input. We can compute the full derivative $\frac{d\mathcal{L}}{d\mathbf{x}}$ by combining the partial derivatives $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ and $\frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial \mathbf{x}}$. Extending from Equation 2.3, we obtain (see subsection B.1.1 for full derivation):

$$\ddot{\mathbf{q}} = \left(\nabla_{\dot{\mathbf{q}}}\boldsymbol{\mathcal{L}}'' + \nabla_{\dot{\mathbf{q}}}^{\top}\mathbf{a}\nabla_{\mathbf{a}}\boldsymbol{\mathcal{L}}''\right)^{-1}\left[\nabla_{\mathbf{q}}\mathcal{L} + \nabla_{\mathbf{q}}^{\top}\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L} - \left(\nabla_{\mathbf{q}}\boldsymbol{\mathcal{L}}'' - \nabla_{\mathbf{q}}^{\top}\mathbf{a}\nabla_{\mathbf{a}}\boldsymbol{\mathcal{L}}''\right)\dot{\mathbf{q}}\right] \tag{3.1}$$

Where $\boldsymbol{\mathcal{L}}'' = \left(\nabla_{\dot{\mathbf{q}}}\mathcal{L} + \nabla_{\dot{\mathbf{q}}}^{\top}\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}\right)^{\top}$. However, expanding this matrix-based[1] equation fully would make it very complex. It turns out that these complications can be avoided, at least from an implementation perspective - it is simpler to consider the actor as part of the Lagrangian neural network. Using auto-differentiation in deep learning frameworks [147, 148, 149], we can directly compute the full derivative w.r.t. $\mathbf{q}$ and $\dot{\mathbf{q}}$.

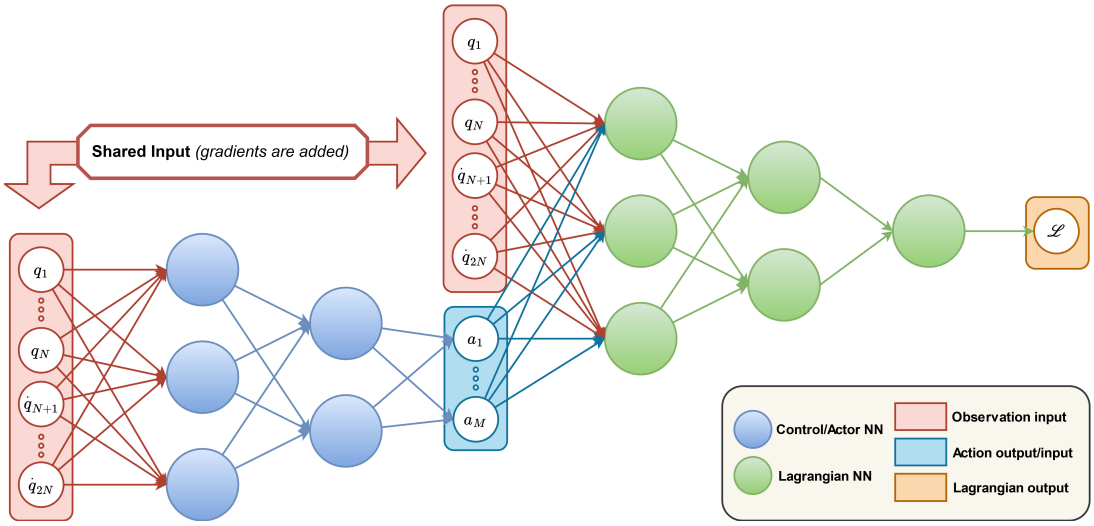


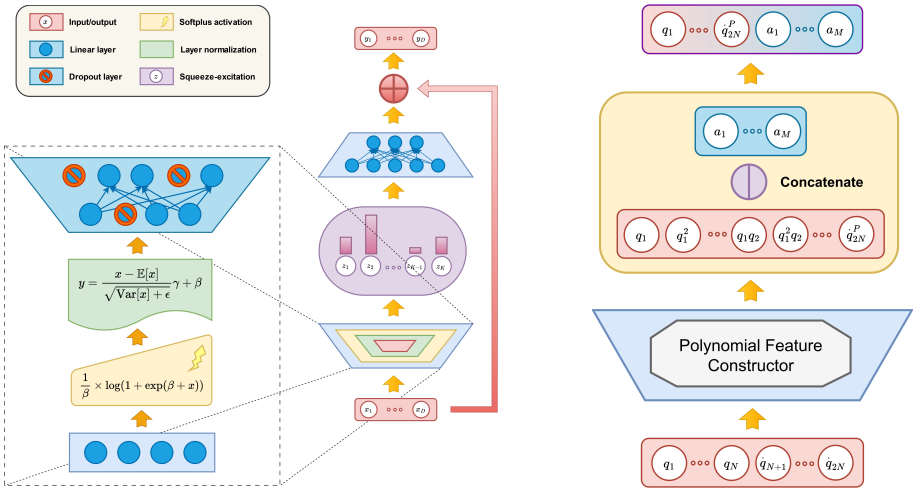
Figure 3.1: Lagrangian Neural Network for Control

One could, in fact, think of it as dense connectivity [150], where features extracted by the actor network are concatenated with the original input. The downside is that this increases training time for the Lagrangian network since the actor's learning adjusts its weights, affecting Jacobians and Hessians required for the LNN forward pass. However, this is manageable because dynamic world models are usually trained after each trajectory collection step [151, 59].

[1] Matrix-based derivation is necessary for efficiency

3.1.3 Full Architecture and Enhancements

Architecture. The main building block adopted by our Lagrangian and non-conservation model is illustrated in Figure 3.2a. It consists of the following layers: linear [152], softplus [153], layer normalization [154], and dropout [155]. We also draw inspiration from efficient networks [156, 157] and use residual connections [87] with squeeze excitation [158], tailoring them to non-convolution networks. Primary motivations are efficiency, compact model size, and generalization but please refer to the original papers and subsection B.1.3 for more details.



(a) Main building block **(b)** Feature constructor

Figure 3.2: LNNc block and feature constructor. Note that actions are produced by an actor and can be differentiated with respect to $\mathbf{q}, \dot{\mathbf{q}}$.

We also made the following enhancements, which help speed up optimization:

- **Features**. We generate polynomial features from the observations, reflecting the interactions seen in kinetic and potential energy equations [159, 160, 161].
- **Lagrangians**. We predict multiple Lagrangians (e.g., 1000) rather than just one, allowing the network to consider multiple possibilities.
- **Ensemble**. Although dropout layers could already be interpreted as a form of ensemble [162], we offer to average Lagrangian vectors from multiple LNNs. This is different from the previous point as the weights are not shared. Due to small network size, we can afford multiple models, and ensembles almost always ensure performance gains [163, 164].

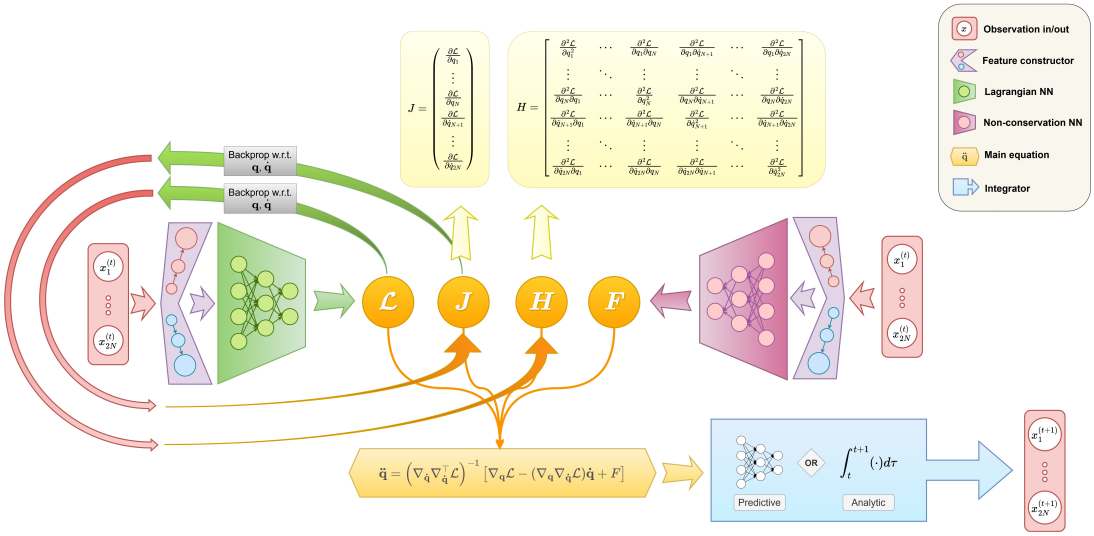


Figure 3.3: Full architecture and workflow of LNNc. Feature constructor is expanded in Figure 3.2b. For clarity, multiple lagrangians and ensemble structure is not illustrated.

Workflow. The full architecture and the workflow process from current $\mathbf{x}^{(t)}$ to next state $\mathbf{x}^{(t+1)}$ is depicted in Figure 3.3.

1. First, the observation input $\mathbf{x}$ is passed through the actor to get the actions $\mathbf{a}$ and through the polynomial network to get the features $\tilde{\mathbf{x}}$ (refer to Figure 3.2b).
2. Then, the concatenated input is passed through two separate neural networks to get the Lagrangian vector $\mathcal{L}$ and the non-conservation term F.
3. We then differentiate the Lagrangians with respect to observations to obtain the Jacobian vector J which we further differentiate to obtain the Hessian matrix H.
4. The acceleration $\ddot{\mathbf{q}}$ is computed using *Equation 11* from the work by Xiao, Zhang, and Tang [145] (also shown in the figure), however, our method computes the full derivative since we differentiate the actor model as pointed in Figure 3.1).
5. Finally, we integrate the result to get the next state, either using an additional neural network or analytical method, such as Euler integration:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \delta\tau\nabla_t\mathbf{x}^{(t)} = \begin{pmatrix}\mathbf{q}^{(t)} & \dot{\mathbf{q}}^{(t)}\end{pmatrix}^\top + \delta\tau\begin{pmatrix}\dot{\mathbf{q}}^{(t)} & \ddot{\mathbf{q}}^{(t)}\end{pmatrix}^\top \tag{3.2}$$

We also would like to add that Jacobian and Hessian computations are computationally demanding, however, we found a way to do this quickly utilizing specific PyTorch [147] functions. We refer the reader to the original code but the comparisons will also be discussed in the Analysis section.

3.2 Gradient Based Proximal Policy Optimization

3.2.1 Planning and Dreaming

Our model-based algorithm relies on Dyna [165]. The two main components in our case are the PPO agent $\mathcal{A}_\theta$ and the LNNc-based world model $\mathcal{M}_\phi$. Dyna-PPO is a common choice [119, 166, 167] and varies regarding world models. It involves a planning phase:

Algorithm 3 Planning

Require: T, B ▷ Attributes - see subsection B.2.4

procedure PLANNING($\mathcal{A}_\theta$, $\mathcal{M}_\phi$)
  $\mathcal{D} \leftarrow \text{collect}(\mathcal{A}_\theta, \mathcal{M}_\phi, T, 1, B)$ ▷ Rollout $\mathcal{M}_\phi$ (Algorithm 10)
  $\mathcal{A}_\theta \leftarrow \mathcal{A}_\theta.\text{update}(\mathcal{M}_\phi, \mathcal{D})$ ▷ Agent update (Algorithm 5)
  return $\mathcal{A}_\theta$

While proximal updates for the actor (Algorithm 2) are sufficient, we introduce a gradient-informed method inspired by the Dreamer family [168, 169, 74], where analytical gradients are used for policy updates. Unlike Dreamer, our algorithm directly sums the state values instead of optimizing cumulative rewards, simplifying the process due to no involvement of a recurrent model.

Algorithm 4 Dreaming actor update

Require: $O, \pi_\theta, \nu_\phi, N, B, H, \varepsilon$ ▷ Attributes - see subsection B.2.4

procedure DREAMING_UPDATE($\mathcal{M}_\theta$, $\mathcal{D}$)
  for $n = 1, \cdots, N$ **do**
    $\{(o_b^{(t)} \leftarrow \text{sample}(\mathcal{D}),\ v_b \leftarrow 0)\}_{b=1}^{B}$ ▷ Sample obs, init state values
    for $(h = 1, \cdots, H) \times (b = 1, \cdots, B)$ **do**
      $a_b^{(t+h)} \leftarrow \pi_\theta(o_b^{(t+h-1)}) + \varepsilon$ ▷ Act on policy + noise
      $o_b^{(t+h)} \leftarrow \mathcal{M}.\text{dynamics}(o_b^{(t+h-1)}, a_b^{(t)})$ ▷ Step dynamics model
      $v_b \leftarrow v_b + \nu_\phi(o_b^{(t+h)})$ ▷ Update b^{th} value sum
    $\mathcal{L} \leftarrow -\mathbb{E}[\{\frac{1}{H} v_b\}_{b=1}^{B}]$
    $\pi_\theta \leftarrow O.\text{optimize}(\pi_\theta, \mathcal{L})$
  return π_θ

The motivation for policy updates being analytical is that the Lagrangian dynamics model uses the actor's gradients to predict the next state. During actor updates, the model "sees" how its parameters and the gradient flow influence the next state. Further, the actor's parameters are directly involved in energy conservation.

3.2.2 Agent and World

Full agent update consists of 3 steps: one critic update and two actor updates. Note, however, in the actual code `critic` and `proximal` updates are merged for efficiency.

Algorithm 5 Agent update

Require: O, π_θ, ν_ϕ ▷ Attributes - see subsection B.2.4

procedure $\mathcal{A}_\theta$.UPDATE($\mathcal{M}_\theta$, $\mathcal{D}$)
  $\nu_\phi \leftarrow$ critic_update($\mathcal{D}$) ▷ Critic update (Algorithm 1)
  $\pi_\theta \leftarrow$ proximal_update($\mathcal{D}$) ▷ Proximal update (Algorithm 2)
  $\pi_\theta \leftarrow$ dreaming_update($\mathcal{M}_\theta, \mathcal{D}$) ▷ Dreaming update (Algorithm 4)
  return $\mathcal{A}_\theta$

The world model is updated (Algorithm 6) before each planning phase. This model includes the Lagrangian dynamics for predicting the next state $\mathbf{x}^{(t+1)}$, as well as reward and termination models, which we implement using simple multi-layer perceptrons (MLPs) [170] with Rectified Linear Unit (ReLU) activations [171]. While more complex world models, such as convolutional neural networks [172] or transformers [173], are common [174, 175, 176], we opt for simplicity because our focus is on dynamics problems where predictions depend only on the current state and action.

Algorithm 6 World update

Require: $O, \mathcal{S}_\theta, \mathcal{R}_\phi, \mathcal{C}_\psi, T, B, T_{\max}$ ▷ Attributes - see subsection B.2.4

procedure $\mathcal{M}_\phi$.UPDATE($\mathcal{A}_\rho$, $\mathcal{D}$)
  $\{((o_b^{(t)}, o_b^{(t+1)}), (r_b^{(t)}, r_b^{(t+1)}), (c_b^{(t)}, c_b^{(t+1)})) \leftarrow \text{pairs}(\mathcal{D})\}_{b=1}^{B}$ ▷ Sample
  for $n = 1, \cdots, N$ **do**
    $\mathcal{L}_\mathcal{S} \leftarrow \frac{1}{B}\sum_{b=1}^{B}(o_b^{(t+1)} - \mathcal{S}_\theta(o_b^{(t)}, \mathcal{A}_\rho(o_b^{(t)}))^2$ ▷ MSE(obs, pred)
    $\mathcal{L}_\mathcal{R} \leftarrow \frac{1}{B}\sum_{b=1}^{B}(r_b^{(t+1)} - \mathcal{R}_\phi(o_b^{(t)}, \mathcal{A}_\rho(o_b^{(t)}))^2$ ▷ MSE(rews, pred)
    $\mathcal{L}_\mathcal{C} \leftarrow \frac{1}{B}\sum_{b=1}^{B}(c_b^{(t+1)} - \mathcal{C}_\psi(o_b^{(t)}, \mathcal{A}_\rho(o_b^{(t)}))^2$ ▷ MSE(cont, pred)
    $(\mathcal{S}_\theta, \mathcal{R}_\phi, \mathcal{C}_\psi) \leftarrow O.\text{optimize}((\mathcal{S}_\theta, \mathcal{R}_\phi, \mathcal{C}_\psi), (\mathcal{L}_\mathcal{S}, \mathcal{L}_\mathcal{R}, \mathcal{L}_\mathcal{C}))$
  return $\mathcal{M}_\theta$

Admittedly, physics-based interpretations could also be sought for reward and termination models [119, 177], however, their contribution is minimal beyond explainability for simple problems. The dynamics model is the primary driver, ensuring aspects like safe exploration. In other words, physics constraints do not have to be imposed everywhere, in which regard we keep flexible - we only consider constraints for agent and world models, where bigger effects are more likely.

3.2.3 Lagrangian-Based Policy Optimization Algorithm

Finally, we present the full algorithm. We iterate N times, where at each iteration we:

1. Collect trajectories of length T with batch size B from real environment $\mathcal{E}$ into on-policy and off-policy buffers $\mathcal{D}_{\text{on}}$, $\mathcal{D}_{\text{off}}$ (see Algorithm 10).
2. Use the real data from off-policy buffer $\mathcal{D}_{\text{off}}$ to update the world model $\mathcal{M}_\phi$ (Algorithm 6) and from on-policy buffer $\mathcal{D}_{\text{on}}$ to update the agent $\mathcal{A}_\theta$ (Algorithm 5). On-policy buffer $\mathcal{D}_{\text{on}}$ is emptied afterwards.
3. Finally, perform planning (Algorithm 3) to collect surrogate environment (world model) data and update the agent using this data.

Algorithm 7 Full LBPO algorithm

Require: $\mathcal{A}_\theta$, $\mathcal{M}_\phi$, N, T, B, $T_{\max}$ ▷ Attributes - see subsection B.2.4

procedure LBPO($\mathcal{E}$)
  $\mathcal{D}_{\text{off}} \leftarrow \{\}$
  for $n = 1, \cdots, N$ **do**
    $\mathcal{D}_{\text{on}} \leftarrow \text{collect}(\mathcal{A}_\theta, \mathcal{E}, T, (n-1)\cdot T + 1, B)$ ▷ Rollout real environment
    $\mathcal{D}_{\text{off}} \leftarrow \text{trim}(\mathcal{D}_{\text{off}} \cup \mathcal{D}_{\text{on}}, T_{\max})$ ▷ Update off-policy buffer
    $\mathcal{M}_\phi \leftarrow \mathcal{M}_\phi.\text{update}(\mathcal{A}_\theta, \mathcal{D}_{\text{off}})$ ▷ Update $\mathcal{M}_\phi$ from real data
    $\mathcal{A}_\theta \leftarrow \mathcal{A}_\theta.\text{update}(\mathcal{M}_\phi, \mathcal{D}_{\text{on}})$ ▷ Update $\mathcal{A}_\theta$ from real data
    $\mathcal{A}_\theta \leftarrow \text{planning}(\mathcal{M}_\phi, \mathcal{A}_\theta)$ ▷ Update $\mathcal{A}_\theta$ from imagined data
    $\mathcal{D}_{\text{on}} \leftarrow \{\}$ ▷ Reset on-policy buffer
  return π_θ

Additionally, we found that by providing critic the previous action (Equation 3.3), i.e., not just the current state but also the action that led to it, the predicted value is more accurate. It is quite surprising considering there is no information about the previous observation, however, this could be problem-dependent. Also, it should not be confused with Q-value predictions [178, 177] where the action is at timestep t.

$$v \leftarrow \nu_\theta(x^{(t)}, a^{(t-1)}) \tag{3.3}$$

3.3 Athletes Software Package

3.3.1 Features

We would like to present an accompanying software package called *Athletes* available at https://github.com/mantasu/athletes. It was built alongside the introduced LBPO algorithm, which depends on the package. Although many reinforcement learning packages exist [179, 180, 181], they are primarily built for industrial audiences. They are convenient when it comes to applying existing SOTA to specific RL problems, however, extending them to custom algorithms or even a minor functionality change beyond parameter reconfiguration requires a deep understanding of the library's components. An example of RLLib's complexity is discussed in subsection B.3.1. Our package is lightweight and is built on the following principles:

- **Simplicity**: only a few components form the application programming interface (API) core. Ideal for academia and students.
- **Generality**: easily extendible (subsection B.3.2), configurable via Hydra [182][2], supporting PyTorch [147] (and, possibly, TensorFlow [148] in the future). Ideal for research and developers.
- **Efficiency**: few dependencies, lightweight data structures, multi-GPU/TPU support. Ideal for industry and engineers.

Our package also supports nested agents (generality principle, see subsection B.3.3), a feature not seen in other libraries. This could be exploited in some algorithms, e.g., hierarchical or compositional RL [184, 185]. Additionally, it also supports tensorized environments (efficiency principle), which are recreations of Gymnasium's [35] environments but in PyTorch, yielding more efficiency than native vectorized versions.

One of the drawbacks is that it is not complete - many SOTA algorithms and tensorized environments are missing. Further, the target version is Python 3.13[3] due to certain features[4] and is not planned to be compatible with lower versions. Additionally, multi-agent RL [186] is not supported for now (as is not by Gymnasium).

[2] Will be supported when certain issues are fixed, such as OmegaConf's [183], which Hydra depends on, support for generic classes (https://github.com/omry/omegaconf/issues/731)

[3] To be released in autumn 2024 https://docs.python.org/3.13/whatsnew/3.13.html

[4] For instance, we utilize type defaults (https://peps.python.org/pep-0696/) to serve the dual purpose of clearer type annotations and the ability for the config class to instantiate a configurable purely from generic (https://docs.python.org/3/library/typing.html#user-defined-generic-types) argument

3.3.2 Structure

Our package primarily depends on Gymnasium and PyTorch. We would like to highlight a couple of things based on the class diagram illustrated below (Figure Figure 3.4):

- The two main components are environment and agent, two opposing steppers. Common classes in other frameworks like policy, algorithm, and learner are avoided. In our case everything that is not a runner, tool, or utility, is an agent, which keeps the architecture easy-to-read and beginner-friendly.
- Athletes conveniently splits between real and surrogate environments, allowing trained surrogate world models to be run in place of real environments and allowing surrogate environments to borrow real functionalities, such as termination conditions, which is especially helpful when debugging performance.

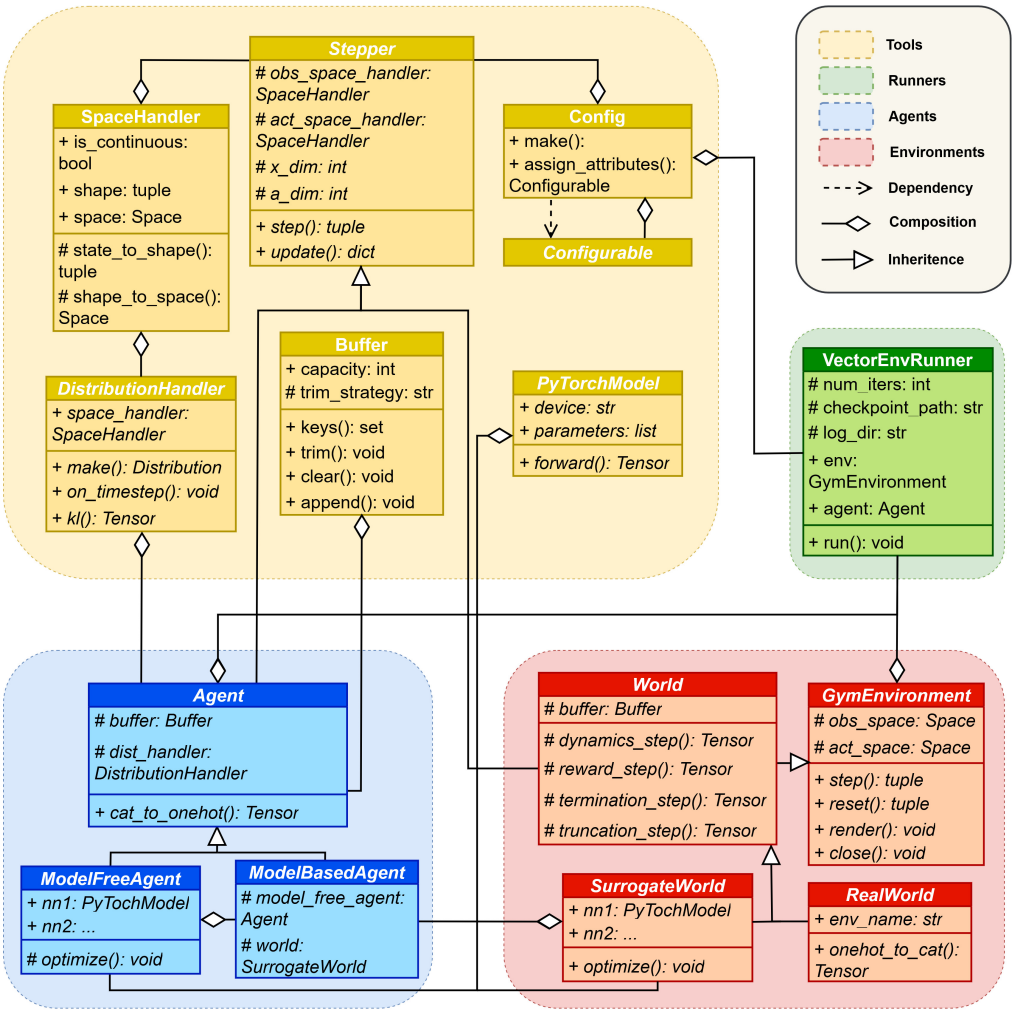


Figure 3.4: Athletes class diagram. It only shows a simplified architecture of the package (mostly *abstract* classes) and does not fully reflect the real structure.

Chapter 4

Analysis

4.1 Setup

4.1.1 Environments

We conduct a series of experiments to inspect the performance across different policy optimization algorithms and their components. Primarily we focus on Cartpole [36] and Pendulum [37] environments, which are used to analyze the algorithmic behavior and architecture components due to simplicity and fast training:

- **Cartpole** - cartpole environment's observation space consists of cart's position, poles agle and their velocities $(x, \theta, \dot{x}, \dot{\theta})$. The action space is discrete - the direction to which the force is applied (0 or 1).
- **Pendulum** - pendulum environment's observation space is simply - only one angle and its velocity $(\theta, \dot{\theta})$. However, the action, i.e., the force to apply is a continuous variable.

Additionally, we perform algorithm evaluation on more sophisticated environments in subsection 4.3.3, namely Inverted Pendulum [114], a continuous version of Cartpole, Acrobot [58], and Hopper [187]:

- **Inverted Pendulum** - the inverted pendulum has the same representation as cartpole, except the action is continuos.
- **Acrobot** - acrobot's observation space consists of two joint angles and their velocities $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$. The action space is discrete with three options - either

apply ± 1 to the actuated joint or do nothing.

- **Hopper** - hopper has the largest observation space. It consists of two torso coordinates, its angle, and three joint angles, followed by velocities of all the respective variables $(x, z, \theta_1, \theta_2, \theta_3, \theta_4, \dot{x}, \dot{z}, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4)$.

All environments were presented in a way that we process them - we always make sure the state is represented as $\mathbf{q}, \dot{\mathbf{q}}$ vector, and if the state has, for example, $\sin\theta, \cos\theta$, we convert that to single θ.

4.1.2 Training

The hyperparameters for each experiment are discussed individually and experiment configurations are available with the provided code. Some additional details on the architectures used:

- **Actor**. The actor model by default consists of only two hidden layers with 64 units each. This design is rather standard across different packages, with the exception that our activation function is hyperbolic tangent (Tanh) [188] instead of ReLU [171], which we choose because our gradients cannot be zero when the Lagrangian neural network differentiates the actor.
- **Critic**. The critic uses the same architecture as the actor, however, this time its predictions are not required to compute the dynamics step, thus we keep ReLU activation. We choose a leaky variant [189] as it helps to maintain the gradient flow when computing the analytical gradients of the value function during the dreaming update.

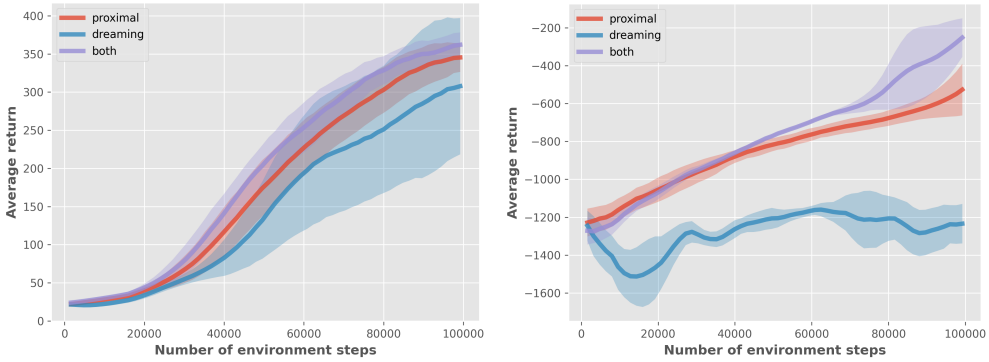
For all experiments, we use AdamW optimizer [190] and, where indicated, a cosine annealing with warm restarts scheduler [191]. The actor-critic and world model loss functions have already been introduced in section 3.2. For proximal loss, we additionally perform entropy regularization [192], which is essential for on-policy algorithms as it helps exploration. As a further regularizer, we utilize Kullback Leibler (KL) divergence [193], a common way to stabilize policy learning [194].

The presented results are averaged over multiple experiment trials (at least 3). We also apply smoothing for visualization purposes. The variance is denoted by coloured plots around the curves.

4.2 Individual Components

4.2.1 Gradient Boosted Proximal Policy Optimization

We perform groups of experiments for individual components to assess and justify the design choices. This section focuses on algorithmic behavior and, in particular, how the additional dreaming step in proximal policy optimization (see Algorithm 5) enhances the performance. Here we do not use the Lagrangian neural network as our world model - instead, we compute the ground-truth dynamics to fairly assess the agent update, irrespective of the accuracy of the world model. Further experimental setup, including hyperparameters, is outlined in section C.1.



(a) Update type comparison for Cartpole **(b)** Update type comparison for Pendulum

Figure 4.1: Comparison between PPO (proximal), gradient-based optimization (dreaming), and the combination of them, which is our algorithm (both).

Figure 4.1 illustrates the average return value over the number of environment steps. As we can see, the dreaming update on its own is not capable of optimizing the policy reliably. We suspect there are three reasons for that:

1. The critic predictions for state values are relative, rather than absolute (this is how PPO works). The critic is always retrained to reflect the relative goodness of the states that were recently encountered. This makes it difficult to determine to what global extent the state was bad when computing the dreaming loss (the negated state value), which is not an issue in the case of proximal update where old vs new policy ratio is utilized.

2. Ground-truth values are not involved - the true cumulative returns are only used in the proximal update, thus not having any pivot in the dreaming update could lead to deviations.

3. In the case of Pendulum, where a bit of exploration is needed to catch the direction of where to improve, state values are only but so helpful - advantages are a stronger way to indicate where an improvement can be made due to temporal difference information.

We could speculate that a more global critic or advantage predictor would help the dreaming update. Choosing an off-policy method could also help the learning.

On the other hand, using both proximal and dreaming updates proves to be beneficial. Based on the points before, it makes sense - the proximal step already takes care of the update direction, therefore the dreaming step can enhance it by adjusting the actor in a way its actions result in even higher state values as guided by the critic. One might say that the results are better simply because we perform more update steps when we combine both updates. We show that this is not necessarily the case - Figure 4.2 illustrates how the performance changes based on the number of stochastic gradient descent iterations.

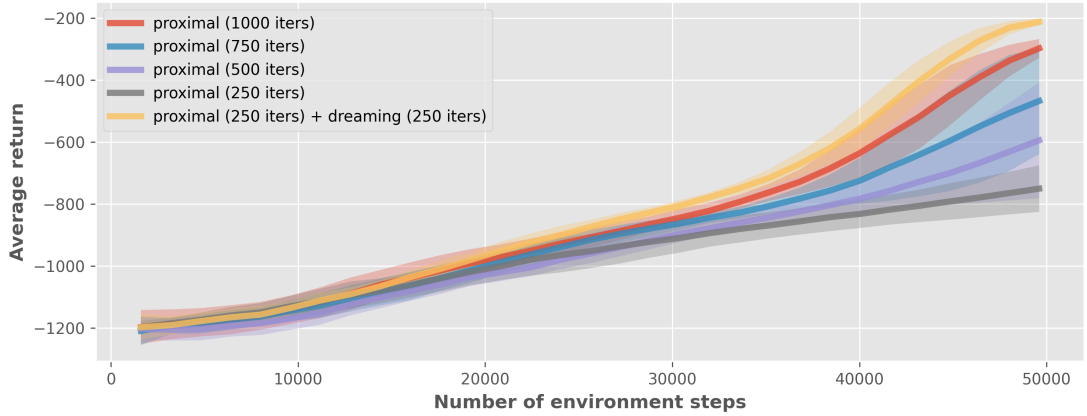


Figure 4.2: Comparison of different numbers of stochastic gradient descent updates for Pendulum environment.

As is clear from the graph, combining dreaming with proximal policy update leads to a faster convergence, compared to when using the same number of gradient steps just for the proximal update. One could probably further improve it by fine-tuning the ratio of proximal vs dreaming steps. Overall we can conclude that our introduced dreaming update boosts the PPO algorithm.

4.2.2 Planning

In this section, we justify the use of planning - an offline agent training method via the learned dynamics and reward models. Here, we used a two-layer perceptron with 120 hidden units and ReLU activation to predict the state derivative. The results for Cartpole environment can be seen in Figure 4.3:

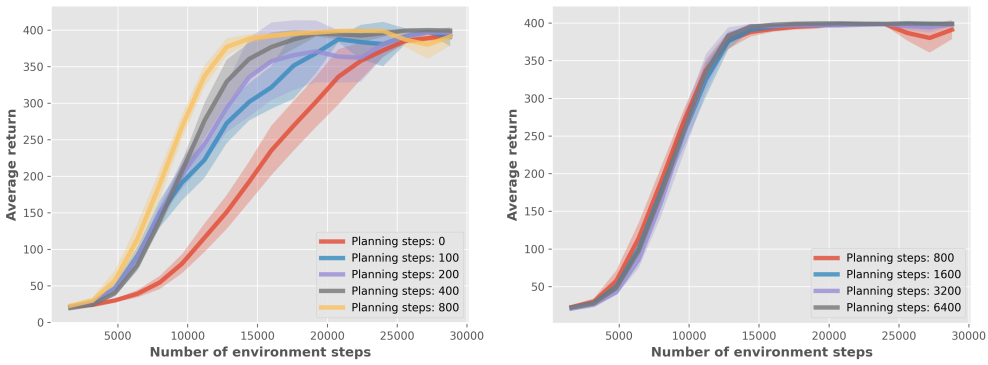


Figure 4.3: Comparison of different numbers of planning steps for Cartpole environment. The number of environment steps only corresponds to the real environment. Coloured plots represent standard deviation.

As expected, planning indicates that average return can increase more quickly as the agent additionally learns from the surrogate environment. The more planning steps are performed, the better the sample efficiency is.

However, there seems to be a cap, beyond which the sample efficiency does not increase. This is natural because the dynamics model, especially such as a multi-layer perception, cannot learn the full range of dynamics just from a few data points. The state distribution shifts as the agent performs better actions and only then the dynamics model is able to narrow down the precise dynamics of the environment. The only advantage of having more planning steps beyond this cap is to ensure stability - for instance, we can see that with 800 planning steps, we may still have some fluctuations, such as the dip at around 2,700th step.

One must also not forget the drawbacks — planning could backfire if the world model is not accurate. Additionally, it can be very computationally demanding.

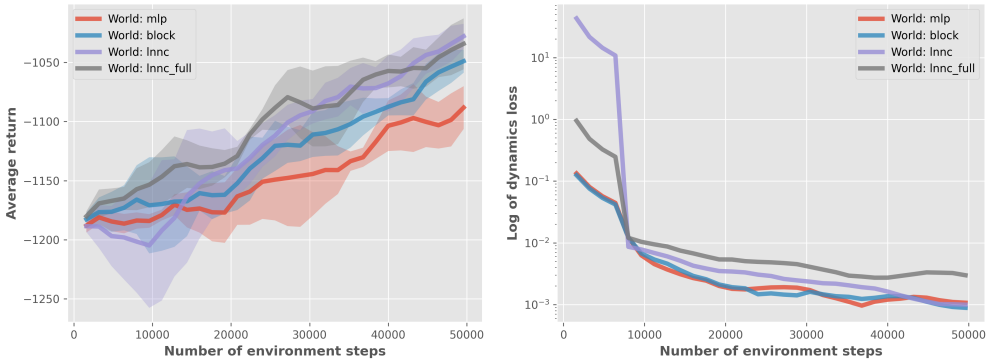
4.3 Algorithm Performance

4.3.1 World Models

Here we inspect the performance of four surrogate world models:

1. **MLP** - a two-layer perception with 64 hidden units and ReLU in between. It is configured to directly predict $\dot{\mathbf{q}}$, $\dot{\mathbf{q}}$.
2. **MobileNet** - a MobileNet-like model described in subsection 3.1.3. It is essentially two LNNc blocks stacked together and also directly predict $\dot{\mathbf{q}}$, $\dot{\mathbf{q}}$
3. **LNNc** *(partial)* - our proposed Lagrangian NN but it does not backpropagate the actor network and only computes partial derivatives of the Lagrangian.
4. **LNNc** *(full)* - the full LNN for control as shown in Figure 3.1

For computational reasons, we do not perform the dreaming step. The results are below:



(a) Average return **(b)** Dynamics log MSE

Figure 4.4: Comparison between different world models for pendulum environment.

This experiment confirms that the Lagrangian neural networks perform better than plain neural ODE predictors, although, arguably performances are similar and evaluation over a larger range of parameters would be more insightful. However, it seems that the LNNc *full* outraces the other architectures, meaning the hypothesis that it "respects" the actor parameters during inference holds. It slows down, however, as other networks catch up with it. This could be explained by its high dynamics error, which is expected because LNNc needs more effort to adjust its weights in response to better-performing actor for correct prediction of Lagrangians. This, therefore, highlights a disadvantage of our architecture which is that one would have to dedicate more resources to train our network, especially if it scales.

4.3.2 Physical Properties

In this section, we evaluate the physical properties of our Lagrangian world model for control by comparing it with a standard two-layer perceptron, which we trained using a Dyna-style algorithm (see Algorithm 3 and Algorithm 6). Specifically, we generate imaginary trajectories—similar to those produced during planning — and examine the cumulative dynamics loss and energy conservation error.

We focus on the cartpole and pendulum environments, as the dynamics equations for these systems are simple, allowing for confident verification of our analysis. These Gymnasium environments do not involve friction, so the dynamics for the cartpole problem are defined as [195]:

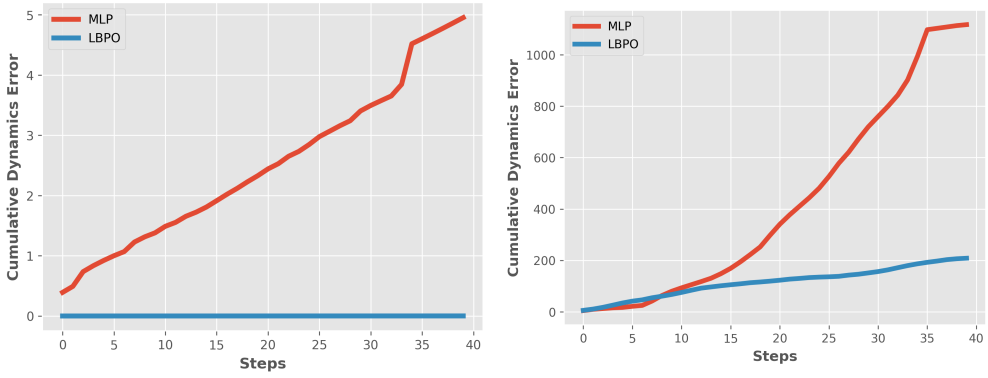
$$\ddot{\theta}_{\text{cartpole}} = \frac{g\sin\theta + \cos\theta\left[\frac{-F - m_p l\dot{\theta}^2\sin\theta + \mu_c \text{sgn}(\dot{x})}{m_c + m_p}\right]}{l\left[\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c + m_p}\right]} \tag{4.1}$$

$$\ddot{x}_{\text{cartpole}} = \frac{F + m_p l(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta) - \mu_c \text{sgn}(\dot{x})}{m_c + m_p} \tag{4.2}$$

And the dynamics for the pendulum are approximated as:

$$\ddot{\theta}_{\text{pendulum}} = \frac{3g}{2l}\sin(\theta) + \frac{3}{ml^2}\tau \tag{4.3}$$

The figure below illustrates the accumulated trajectory error over time:



(a) Cumulitive generated trajectory error for CartPole

(b) Cumulitive generated trajectory error for Pendulum

Figure 4.5: Cumulitive generated trajectory errors. The error for a single time-step is computed as the absolute error between the the predicted next state and the actual next state simulated by a ground-truth dynamics model.

As shown, LBPO maintains stability in the relatively simple cartpole environment and produces small errors in the pendulum environment. In contrast, the MLP method generates larger errors that accumulate linearly and may eventually diverge if the error becomes too large. It may seem surprising that both algorithms produce larger errors in the pendulum environment, given that its dynamics equation (Equation 4.3.2) is simpler than that of the cartpole. However, it is actually more challenging for the agent to learn the solution in the pendulum environment due to the need for more precise (continuous) actions. The agent spends more time in its initial condition, causing the models to develop an initial bias. This highlights a drawback of model-based RL methods—if we do not carefully refine our buffer to include a broader data distribution, the agent's learning may be hindered by numerous planning steps.

Next, we analyze how well the models conserve energy over time. Recall that the total energy of the system is the sum of kinetic and potential energy. For the cartpole, it is computed as follows:

$$KE_{\text{cartpole}} = \frac{1}{2}m_c\dot{x}^2 + \frac{1}{2}m_p((l\omega)^2 + (\dot{x}\cos^2\theta)) \tag{4.4}$$

$$PE_{\text{cartpole}} = m_p g l(1-\cos\theta) \tag{4.5}$$

For the pendulum, the calculation is simpler:

$$KE_{\text{pendulum}} = 0.5m(l\omega)^2 \tag{4.6}$$

$$PE_{\text{pendulum}} = mgl(1-\cos\theta) \tag{4.7}$$

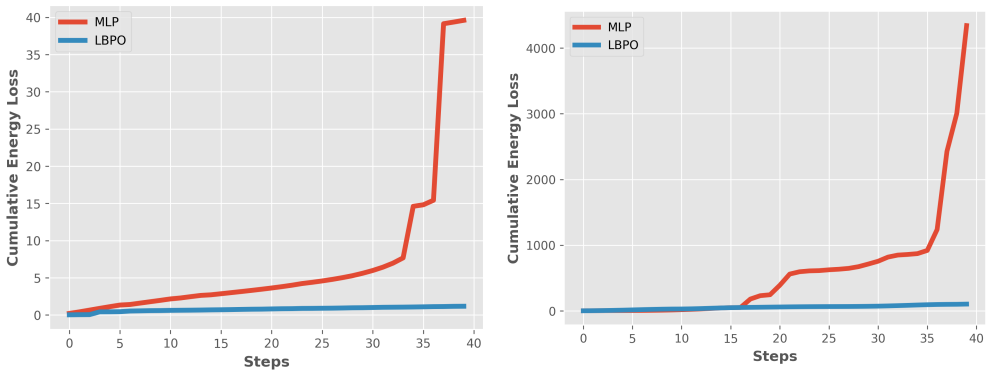
Additionally, we know that external forces are applied, which represent the work done by the actuators. For both systems, the equation is fundamentally the same:

$$W_{\text{cartpole}} = F\delta t\dot{x} \tag{4.8}$$

$$W_{\text{pendulum}} = F\delta t\omega \tag{4.9}$$

When the agent takes an action, it introduces an external force, causing the total energy of the system to change. For the system to be conservative, the change in total energy must equal the work performed by the actuators. We define the error as the absolute difference between these two quantities. The results are shown in the figure below:

It is clear that the error in energy loss for LBPO is extremely low, whereas energy loss for the MLP world model appears to increase exponentially. This suggests that energy conservation is an even more significant issue than state error. Our findings are consistent with other studies on Lagrangian neural networks [23, 112], which demonstrate that energy conservation is a fundamental property of such networks.



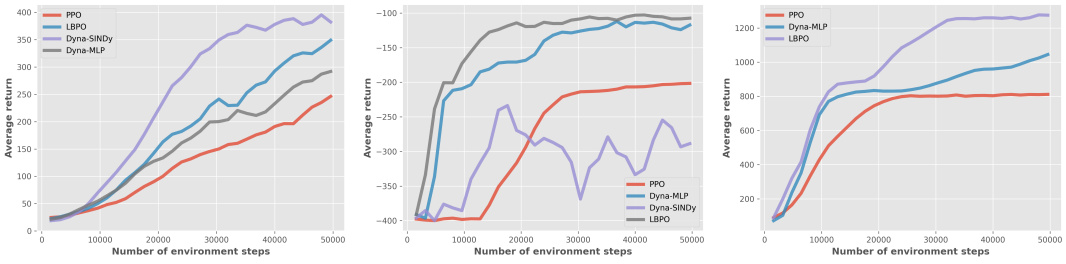
(a) Energy conservation for cartpole environment

(b) Energy conservation for pendulum environment

Figure 4.6: Cumulative generated trajectory energy loss. Both LBPO and Dyna-MLP consist of two hidden layers with 128 neurons each. Both were trained for 200,000 environment interactions. The trajectory error is averaged over 64 samples.

4.3.3 Policy Algorithms

We also present an empirical study on more complex environments to compare our algorithm with other established techniques. As mentioned earlier, we evaluate our approach in three additional environments: Inverted Pendulum, Acrobot, and Hopper. The algorithms we compared include standard Proximal Policy Optimization (PPO) [34], Dyna-MLP with two hidden layers, the symbolic Dyna-SINDy algorithm [119], and our LBPO method. While not all agents successfully reached the solution, the performance curves provide a general intuition of each algorithm's efficiency. The results are shown in Figure 4.7:



(a) Inverted Pendulum **(b)** Acrobot **(c)** Hopper

Figure 4.7: Algorithm comparisons across different environments. For these problems we increased the number of hidden units of all networks to 256. The number of planning steps is 1600 for all algorithms except PPO which does not use it.

The figure above shows that our algorithm scales effectively across different observation spaces. It outperforms other algorithms in the Acrobot and Hopper environments, validating our design choices and demonstrating the applicability of our approach to such problems. Although LBPO did not outperform SINDy-RL in the Inverted Pendulum case, this is understandable. The symbolic solution for the inverted pendulum is relatively straightforward, and once the algorithm identifies the shortest solution, it essentially ensures stability. For the Acrobot, the solution space is more complex, causing SINDy-RL to struggle. As for Hopper, we found it infeasible to run SINDy-RL and thus excluded it. Surprisingly, LBPO performed very well for Hopper, despite the challenges posed by contact dynamics. This demonstrates the generality of our approach and the ability of LNNs to approximate complex dynamics effectively.

4.4 Code Efficiency

4.4.1 Package

In this section, we test how efficient our package is compared to other existing ones, namely RLLib [179] and TorchRL [181]. The results are shown in Figure 4.8.

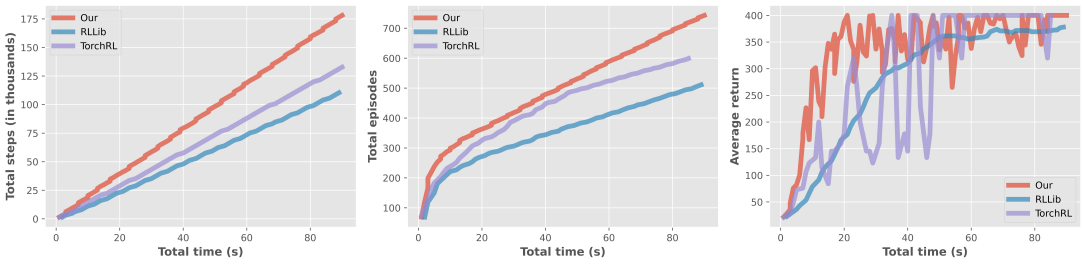


Figure 4.8: Comparing PPO efficiency for CartPole environment. All efficiency parameters, such as trajectory lengths, model sizes, batch sizes, etc., are selected the same. Note that only the first graph truly indicates the performance difference - other ones depend on learning accuracy (e.g., better cartpole agents yield fewer but longer episodes).

We can see that our package outperforms all other libraries regarding the PPO algorithm. This efficiency comes the fact that the package is lightweight, meaning the callback traces are much smaller. Additionally, data structures have also been optimized, e.g., instead of storing experience replay tensors in dictionaries, they are stored as direct object attributes.

4.4.2 Autograd

As mentioned at the end of subsection 3.1.3, we implemented a highly efficient Hessian computation method that uses experimental PyTorch `is_grads_batched` feature. To the best of our knowledge, no other work related to Lagrangian neural networks has utilized it. More specifically, we identify three ways to perform gradient computation:

1. **Naive** - uses `jacobian` and `hessian` from `torch.autograd.functional`. They are primarily designed to be used for single vectors, thus requiring a python loop if the input is batched.
2. **Functional** - uses `jacrev` and `hessian` from `torch.func`. These can be combined with `torch.vmap` to build an implicit C loop when computing the result.
3. **Autograd** - uses `grad` two times from `torch.autograd`. Unlike the previous methods, this one does not accept target function as input, it directly takes the output and the input, with respect to which the differentiation should happen.

It may sound intuitive and simple to just call `grad` twice, however, due to batched vectors, certain tricks have to be performed (see subsection C.2.1) to individualize hessian computations. For more details, please refer to our code.

As can be seen in Table 4.1, our method is twice as fast as the functional method that is provided off-the-shelf by PyTorch. Such speedups in Hessian computations are essential in training Lagrangian neural networks - it means we can speed up the optimization twice purely from code perspective.

Method	Jacobian (s)	Hessian (s)
Naive [196][1]	0.07	14.24
Functional [112][2]	2×10^{-4}	1.19
Autograd (our)	6×10^{-4}	0.57

Table 4.1: Efficiency comparison of different gradient computation methods using PyTorch and example papers that use them. Computations are performed for a random function $f(\mathbf{x}) := \sum_{d=1}^{D} 2x_d^3 - x_d^2$ for a batch $B = 256$ of random inputs $\mathbf{x} \in \mathbb{R}^{1000}$.

[1] https://github.com/uwsbel/sbel-reproducibility/tree/master/2024/MNODE-code
[2] https://github.com/adi3e08/Physics_Informed_Model_Based_RL/tree/main

Chapter 5

Conclusions

5.1 Discussion

We have introduced a novel physics-informed reinforcement learning algorithm based on Lagrangian mechanics. Previous works like DeLaN [24] and PIMB-RL [112] applied Lagrangian neural networks primarily to problems involving rigid body motion. Our work generalizes this approach by integrating the actor-network as part of the Lagrangian world model (section 3.1). In this framework, the Lagrangian neural network "understands" the actor's intentions by "reading" its parameters when computing the Jacobian and Hessian to predict the next state. We further enhance this connection by introducing a gradient-boosted Proximal Policy Optimization algorithm (section 3.2), which includes dreaming steps that allow the critic's parameters to more directly influence the actor's updates. In this sense, the world model acts as a mediator, translating the actor's intentions for the critic and facilitating communication via double differentiation. It's as if the actor can query the critic to verify whether its decisions are sensible through a physics-informed world model.

This is just one interpretation of the gradient-informed interaction between these three entities, but our results confirm its effectiveness. We've demonstrated that dreaming and planning are effective methods for improving sample efficiency (section 4.2), and that Lagrangian neural networks possess the necessary physical properties (subsection 4.3.2), making them a superior choice for classical dynamics problems. However, it is also important to acknowledge the limitations of this approach (section 5.3).

5.2 Future Work

There are several important directions for the future work:

- **World Models**: Firstly, the architecture of the Lagrangian neural network could be explored further. We only considered an ensemble Lagrangian predictor with various enhancements like dense connectivity (see subsection 3.1.3). However, recent architectures, such as deep operator networks (DeepONets) [25], have challenged the standard data-to-data predictors and have shown that function-to-function approximators are more suitable for solving and analyzing ODEs and PDEs. In this sense, we could use DeepONet to learn a solution operator G that maps input function $x(t)$ and control function $u(t)$ to a Lagrangian function $l(t|x(t),u(t))$. One could even look for connections with hyper-heuristic methods [197] which also operate in problem space. Methods like these are deemed to be highly generalizable [198, 199].

- **Agent Models**: We have only experimented with proximal policy optimization agents. However, there are many other algorithms that could challenge its performance. For example, soft actor-critic (SAC) [200] is an off-policy algorithm, which has been shown to be more sample efficient than on-policy algorithms like PPO. Furthermore, our introduced dreaming proximal policy optimization could be modified to differentiate advantages, such as in gradient-informed PPO [201].

- **Explainability**: One would probably associate physics-informed machine learning with some form of interpretability. Although our architectural design makes the predictive behavior more interpretable than black-box neural networks due to the Euler-Lagrange constraint, we could still explore this aspect further. Some models, such as deep symbolic optimization (DSO) [202] and SINDy-RL [119] train models to predict symbolic forms of dynamics and policy equations. We could use these models on top of our learned one to at least partially explain the dynamics or we could look for symbolic, differentiable forms of computing a Lagrangian, the primary equation of which could be further refined by a neural network. We actually have experimented with symbolic world models, by reimplementing SINDy-RL, and implementing two custom world models for control adopting PySR [29] and symbolic identification of non-linear dynamics (SymINDy) [203]. Unfortunately, due to time constraints and because these are slightly off-topic, we did not include them nor systematically experimented

with them. However, the draft implementations are available in our code at `src/supplementary/draft/envs/surrogate/worlds`.

- **Evaluation**: Our experiments were primarily performance-focused. We could perform qualitative evaluation by visualizing world model predictions on a reduced-dimension time-space coordinate system, and inspecting how consistent and smooth the trajectories are, and whether they make sense. Additionally, we could evaluate how sensitive the models are to the initial observation conditions and perform experiments over a range of parameters to determine training stability.
- **Scalibility** An important factor to assess is algorithm's scalibility. While we only experimented on toy problems, it would be more practical to test the algorithm in real environments, with lots of unknown factors. Furthermore, simple problems like cartpole and pendulum only have a few observation and action variables, making the problem feasibly solvable by hand or random search for short episodes. Extending the search space to hundreds or even thousands of dimensions could reveal an even bigger benefit of using LNNs because artificial neural networks are generally less susceptible to the curse of dimensionality [204]. On the other hand, symbolic methods, as shown in Figure 4.7a, may work well for low dimensional problems, however, as most of them utilize some form of genetic programming [205], they are subject to bloat and generalization issues [206] as the search space grows exponentially with the number of terminals/non-terminals added.

5.3 Limitations

Despite our successes, our work has some limitations that warrant discussion. First, the gradient-informed communication flow is highly computationally demanding, and due to time constraints, we were unable to fully experiment with and compare the time required for the algorithms to converge. Second, the framework we introduced is limited to generalized coordinate representations, where the observation space can be expressed as $\mathbf{q}, \dot{\mathbf{q}}$. This limitation means that our algorithm cannot be applied to common reinforcement learning benchmarks, such as Atari games, which may involve image-based state spaces. Finally, our approach may struggle with environments that involve highly stochastic dynamics or significant discontinuities, where the assumptions of smoothness in Lagrangian mechanics may not hold, potentially leading to degraded performance or instability.

5.4 Summary

To conclude, our project was a success. Namely, we introduced a powerful physics-informed world model that generalizes to allow actions as inputs and a policy boosting method via critic gradient information. Not only do these methods separately work to improve sample efficiency and physics compliance, but they also seamlessly work together due being more informed about each other's parameters. We would like to also emphasize that this project has also involved software development by which we built a research-friendly Athletes package allowing to easily develop config-based lightweight and efficient policy optimization algorithms.

Bibliography

[1] Hamid Benbrahim and Judy A. Franklin. “Biped Dynamic Walking Using Reinforcement Learning”. In: *Robotics and Autonomous Systems* 22.3 (1997). Robot Learning: The New Wave, pp. 283–302. DOI: 10.1016/S0921-8890(97)00043-2.

[2] Frank L. Lewis, Draguna Vrabie, and Kyriakos G. Vamvoudakis. “Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers”. In: *IEEE Control Systems Magazine* 32.6 (2012), pp. 76–105. DOI: 10.1109/MCS.2012.2214134.

[3] Zhongyu Li et al. “Reinforcement Learning for Versatile, Dynamic, and Robust Bipedal Locomotion Control”. In: *ArXiv* abs/2401.16889 (2024). DOI: 10.48550/arXiv.2401.16889.

[4] Yu. V. Tiumentsev and R. A. Zarubin. “Lateral Motion Control of a Maneuverable Aircraft Using Reinforcement Learning”. In: *Optical Memory and Neural Networks* 33.1 (Mar. 2024), pp. 1–12. DOI: 10.3103/S1060992X2401003X.

[5] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. “Discovering Governing Equations From Data by Sparse Identification of Nonlinear Dynamical Systems”. In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937. DOI: 10.1073/pnas.1517384113.

[6] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

[7] Eric Aislan Antonelo et al. “Physics-Informed Neural Nets for Control of Dynamical Systems”. In: *Neurocomputing* 579 (2024), p. 127419. DOI: 10.1016/j.neucom.2024.127419.

[8] Kenneth S. Krane. “The Pendulum: A Case Study in Physics”. In: *Physics Today* 59.7 (July 2006), pp. 52–53. DOI: 10.1063/1.2337835.

[9] Chayan Banerjee et al. “A Survey on Physics Informed Reinforcement Learning: Review and Open Problems”. In: *ArXiv* abs/2309.01909 (2023). DOI: 10.48550/arXiv.2309.01909.

[10] Yu Han et al. “A Physics-Informed Reinforcement Learning-Based Strategy for Local and Coordinated Ramp Metering”. In: *Transportation Research Part C: Emerging Technologies* 137 (2022), p. 103584. DOI: 10.1016/j.trc.2022.103584.

[11] Majdi I. Radaideh et al. “Physics-informed reinforcement learning optimization of nuclear assembly design”. In: *Nuclear Engineering and Design* 372 (2021), p. 110966. DOI: 10.1016/j.nucengdes.2020.110966.

[12] Desong Du et al. “Reinforcement Learning for Safe Robot Control using Control Lyapunov Barrier Functions”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9442–9448. DOI: 10.1109/ICRA48891.2023.10160991.

[13] Peng Zhao and Yongming Liu. “Physics Informed Deep Reinforcement Learning for Aircraft Conflict Resolution”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 8288–8301. DOI: 10.1109/TITS.2021.3077572.

[14] Di Cao et al. “Physics-Informed Graphical Representation-Enabled Deep Reinforcement Learning for Robust Distribution System Voltage Control”. In: *IEEE Transactions on Smart Grid* 15.1 (2024), pp. 233–246. DOI: 10.1109/TSG.2023.3267069.

[15] Colin Rodwell and Phanindra Tallapragada. “Physics-Informed Reinforcement Learning for Motion Control of a Fish-Like Swimming Robot”. In: *Scientific Reports* 13.1 (July 2023), p. 10754. DOI: 10.1038/s41598-023-36399-4.

[16] Atriya Biswas et al. “Safe Reinforcement Learning for Energy Management of Electrified Vehicle with Novel Physics-Informed Exploration Strategy”. In: *IEEE Transactions on Transportation Electrification* (2024), pp. 1–1. DOI: 10.1109/TTE.2024.3361462.

[17] Yuanzheng Li et al. “Federated Multiagent Deep Reinforcement Learning Approach via Physics-Informed Reward for Multimicrogrid Energy Management”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2023), pp. 1–13. DOI: 10.1109/TNNLS.2022.3232630.

[18] Rajarshi Mukhopadhyay, Ashoke Sutradhar, and Paramita Chattopadhyay. “A novel investigation on the effects of state and reward structure in designing deep reinforcement learning-based controller for nonlinear dynamical systems”. In: *International Journal of Dynamics and Control* (Mar. 2024). DOI: 10.1007/s40435-024-01407-6.

[19] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[20] Ricky T. Q. Chen et al. “Neural Ordinary Differential Equations”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. *NIPS’18*. Montréal, Canada: Curran Associates Inc., 2018, pp. 6572–6583. DOI: 10.5555/3327757.3327764.

[21] Gerald Teschl. *Ordinary Differential Equations and Dynamical Systems*. Vol. 140. American Mathematical Society, 2024.

[22] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. “Hamiltonian Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. DOI: 10.48550/arXiv.1906.01563.

[23] Miles Cranmer et al. “Lagrangian Neural Networks”. In: *ArXiv Preprint ArXiv: 2003.04630* (2020). DOI: 10.48550/arXiv.2003.04630.

[24] Michael Lutter, Christian Ritter, and Jan Peters. “Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning”. In: *International Conference on Learning Representations*. 2019. DOI: 10.48550/arXiv.1907.04490.

[25] Lu Lu et al. “Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators”. In: *Nature Machine Intelligence* 3.3 (Mar. 2021), pp. 218–229. DOI: 10.1038/s42256-021-00302-5.

[26] Sifan Wang, Hanwen Wang, and Paris Perdikaris. “Learning the Solution Operator of Parametric Partial Differential Equations with Physics-Informed DeepONets”. In: *Science Advances* 7.40 (2021), eabi8605. DOI: 10.1126/sciadv.abi8605.

[27] Shaowu Pan, Steven L. Brunton, and J. Nathan Kutz. “Neural Implicit Flow: a Mesh-Agnostic Dimensionality Reduction Paradigm of Spatio-Temporal Data”. In: *The Journal of Machine Learning Research* 24.1 (Mar. 2024). DOI: 10.5555/3648699.3648740.

[28] Imran Nasim and Joaõ Lucas de Sousa Almeida. “Using Neural Implicit Flow To Represent Latent Dynamics Of Canonical Systems”. In: *ArXiv* abs/2404.17535 (2024). DOI: 10.48550/arXiv.2404.17535.

[29] Miles Cranmer. “Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl”. In: *ArXiv* abs/2305.01582 (2023). DOI: 10.48550/arXiv.2305.01582.

[30] Marco Calapristi et al. “Interpretability analysis of Symbolic Regression models for dynamical systems”. In: *2024 International Conference on Control, Automation and Diagnosis (ICCAD)*. 2024, pp. 1–6. DOI: 10.1109/ICCAD60883.2024.10553801.

[31] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. “A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes”. In: *Machine Learning* 49.2 (Nov. 2002), pp. 193–208. DOI: 10.1023/A:1017932429737.

[32] Urban Fasel et al. “SINDy with Control: A Tutorial”. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. Austin, TX, USA: IEEE Press, 2021, pp. 16–21. DOI: 10.1109/CDC45484.2021.9683120.

[33] Thomas M. Moerland et al. “Model-based Reinforcement Learning: A Survey”. In: *Foundations and Trends® in Machine Learning* 16.1 (2023), pp. 1–118. DOI: 10.1561/2200000086.

[34] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv Preprint ArXiv:1707.06347* (2017). DOI: 10.48550/arXiv.1707.06347.

[35] Mark Towers et al. “Gymnasium: A Standard Interface for Reinforcement Learning Environments”. In: *ArXiv* abs/2407.17032 (2024). DOI: 10.48550/arXiv.2407.17032.

[36] Swagat Kumar. “Balancing a CartPole System with Reinforcement Learning–A Tutorial”. In: *ArXiv* abs/2006.04938 (2020). DOI: 10.48550/arXiv.2006.04938.

[37] Yifei Bi, Xinyi Chen, and Caihui Xiao. “A Deep Reinforcement Learning Approach towards Pendulum Swing-up Problem based on TF-Agents”. In: *ArXiv* abs/2106.09556 (2021). DOI: 10.48550/arXiv.2106.09556.

[38] R.S. Sutton and A.G. Barto. “Reinforcement Learning: An Introduction”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 1054–1054. DOI: 10.1109/TNN.1998.712192.

[39] J. E. R. Staddon. “The dynamics of behavior: Review of Sutton and Barto: Reinforcement Learning: An Introduction (2nd ed.)” In: *Journal of the Experimental Analysis of Behavior* 113.2 (2020), pp. 485–491. DOI: https://doi.org/10.1002/jeab.587.

[40] Tom Verguts. “Computational Models of Cognitive Control”. In: *The Wiley Handbook of Cognitive Control*. Hoboken, NJ, US: Wiley Blackwell, 2017, pp. 127–142. DOI: 10.1002/9781118920497.ch8.

[41] Cangqing Wang et al. “Theoretical Analysis of Meta Reinforcement Learning: Generalization Bounds and Convergence Guarantees”. In: *ArXiv Preprint ArXiv:2405.13290* (2024). DOI: 10.48550/arXiv.2405.13290.

[42] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc., 1994. DOI: 10.5555/528623.

[43] Sinan Çalışır and Meltem Kurt Pehlivanoğlu. “Model-Free Reinforcement Learning Algorithms: A Survey”. In: *2019 27th Signal Processing and Communications Applications Conference (SIU)*. 2019, pp. 1–4. DOI: 10.1109/SIU.2019.8806389.

[44] Dhruv Mauria Saxena et al. “Driving in Dense Traffic with Model-Free Reinforcement Learning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 5385–5392. DOI: 10.1109/ICRA40945.2020.9197132.

[45] Anusha Nagabandi et al. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 7559–7566. DOI: 10.1109/ICRA.2018.8463189.

[46] Marcin Andrychowicz et al. “What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study”. In: *ArXiv Preprint ArXiv:2006.05990* (2020). DOI: 10.48550/arXiv.2006.05990.

[47] Tianyu Li et al. “Using Deep Reinforcement Learning to Learn High-Level Policies on the ATRIAS Biped”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 263–269. DOI: 10.1109/ICRA.2019.8793864.

[48] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. “A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2023), pp. 1–. DOI: 10.1109/TNNLS.2023.3250269.

[49] Shixiang Gu et al. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

[50] Shixiang Gu et al. “Continuous deep Q-learning with model-based acceleration”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. *ICML’16*. New York, NY, USA: JMLR.org, 2016, pp. 2829–2838. DOI: 10.5555/3045390.3045688.

[51] D.E. Kirk. *Optimal Control Theory: An Introduction*. *Dover Books on Electrical Engineering*. Dover Publications, 2012.

[52] Paul Albrecht. “Dynamics of the Metal-Cutting Process”. In: *Journal of Engineering for Industry* 87.4 (Nov. 1965), pp. 429–441. DOI: 10.1115/1.3670857.

[53] Robert T. Bupp, Dennis S. Bernstein, and Vincent T. Coppola. “A Benchmark Problem for Nonlinear Control Design”. In: *International Journal of Robust and Nonlinear Control* 8.4-5 (1998), pp. 307–310. DOI: 10.1002/(SICI)1099-1239(19980415/30)8:4/5<307::AID-RNC354>3.0.CO;2-7.

[54] M. Rimer, D.K. Frederick, and C.Y. Huang. “Solutions of the Second Benchmark Control Problem”. In: *IEEE Control Systems Magazine* 10.5 (1990), pp. 33–39. DOI: 10.1109/37.60422.

[55] R. M. Centner and J. M. Idelsohn. “Adaptive Controller for a Metal Cutting Process”. In: *IEEE Transactions on Applications and Industry* 83.72 (1964), pp. 154–161. DOI: 10.1109/TAI.1964.5407783.

[56] M. Margaliot and G. Langholz. “Fuzzy Control of a Benchmark Problem: A Computing With Words Approach”. In: *IEEE Transactions on Fuzzy Systems* 12.2 (2004), pp. 230–235. DOI: 10.1109/TFUZZ.2004.825083.

[57] Alberto M. Simões and Vinícius M. G. B. Cavalcanti. “Missile Autopilot Design via Structured Robust Linear Parameter-Varying Synthesis”. In: *Journal of Guidance, Control, and Dynamics* 46.8 (2023), pp. 1649–1656. DOI: 10.2514/1.G007580.

[58] Richard S Sutton. “Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky, M.C. Mozer, and M. Hasselmo. Vol. 8. MIT Press, 1995.

[59] David Silver et al. “Mastering the Game of Go Without Human Knowledge”. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. DOI: 10.1038/nature24270.

[60] Kenan Ahmic et al. “Reinforcement Learning-Based Path Following Control with Dynamics Randomization for Parametric Uncertainties in Autonomous Driving”. In: *Applied Sciences* 13.6 (2023). DOI: 10.3390/app13063456.

[61] Jean Rabault et al. “Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control”. In: *Journal of Fluid Mechanics* 865 (2019), pp. 281–302. DOI: 10.1017/jfm.2019.62.

[62] Benjamin Recht. “A Tour of Reinforcement Learning: The View from Continuous Control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 2.Volume 2, 2019 (2019), pp. 253–279. DOI: 10.1146/annurev-control-053018-023825.

[63] Steven Douglas Whitehead. “Reinforcement Learning for the Adaptive Control of Perception and Action”. UMI Order No. GAX92-31329. PhD thesis. USA: University of Rochester, 1992. DOI: 10.5555/150910.

[64] J.A. Bagnell and J.G. Schneider. “Autonomous Helicopter Control Using Reinforcement Learning Policy Search Methods”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 1615–1620 vol.2. DOI: 10.1109/ROBOT.2001.932842.

[65] Benjamin Van Roy and John N. Tsitsiklis. “Learning and value function approximation in complex decision processes”. AAI0599623. PhD thesis. USA: Massachusetts Institute of Technology, 1998. DOI: 10.5555/928091.

[66] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. “Reinforcement learning: a survey”. In: *J. Artif. Int. Res.* 4.1 (May 1996), pp. 237–285. DOI: 10.5555/1622737.1622748.

[67] Yaakov Engel. “Algorithms and Representations for Reinforcement Learning”. PhD thesis. Hebrew University of Jerusalem, Israel, 2005.

[68] Terrence J. Sejnowski. *The Deep Learning Revolution*. The MIT Press, Oct. 2018. DOI: 10.7551/mitpress/11474.001.0001.

[69] Ajay Shrestha and Ausif Mahmood. “Review of Deep Learning Algorithms and Architectures”. In: *IEEE Access* 7 (2019), pp. 53040–53065. DOI: 10.1109/ACCESS.2019.2912200.

[70] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

[71] Yuxi Li. “Deep Reinforcement Learning: An Overview”. In: *ArXiv Preprint ArXiv:1701.07274* (2018). DOI: 10.48550/arXiv.1701.07274.

[72] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. “Optimizing Chemical Reactions with Deep Reinforcement Learning”. In: *ACS Central Science* 3.12 (Dec. 2017), pp. 1337–1344. DOI: 10.1021/acscentsci.7b00492.

[73] Jabbar Hussain. “Deep Learning Black Box Problem”. MA thesis. Uppsala University, Department of Informatics and Media, 2019, p. 59.

[74] Danijar Hafner et al. “Mastering Diverse Domains through World Models”. In: *ArXiv Preprint ArXiv:2301.04104* (2024). DOI: 10.48550/arXiv.2301.04104.

[75] Michael I. Jordan. “Chapter 25 - Serial Order: A Parallel Distributed Processing Approach”. In: *Neural-Network Models of Cognition*. Ed. by John W. Donahoe and Vivian Packard Dorsel. Vol. 121. *Advances in Psychology*. North-Holland, 1997, pp. 471–495. DOI: 10.1016/S0166-4115(97)80111-2.

[76] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[77] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Ed. by Dekai Wu et al. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012.

[78] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete-Time Dynamical Systems*. 1st ed. *Studies in Systems, Decision and Control*. Springer Cham, 2017, pp. XVIII, 284. DOI: 10.1007/978-3-319-50763-7.

[79] Manfred Morari and Jay H. Lee. “Model predictive control: past, present and future”. In: *Computers & Chemical Engineering* 23.4 (1999), pp. 667–682. DOI: 10.1016/S0098-1354(98)00301-9.

[80] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. “NeuroAnimator: fast neural network emulation and control of physics-based models”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. *SIGGRAPH ’98*. New York, NY, USA: Association for Computing Machinery, 1998, pp. 9–20. DOI: 10.1145/280814.280816.

[81] George Em Karniadakis et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (June 2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[82] Shengze Cai et al. “Physics-Informed Neural Networks for Heat Transfer Problems”. In: *Journal of Heat Transfer* 143.6 (Apr. 2021), p. 060801. DOI: 10.1115/1.4050542.

[83] George S. Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. “Physics-Informed Neural Networks for Power Systems”. In: *2020 IEEE Power & Energy Society General Meeting (PESGM)*. 2020, pp. 1–5. DOI: 10.1109/PESGM41954.2020.9282004.

[84] QiZhi He et al. “Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport”. In: *Advances in Water Resources* 141 (2020), p. 103610. DOI: https://doi.org/10.1016/j.advwatres.2020.103610.

[85] Salvatore Cuomo et al. “Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What’s Next”. In: *Journal of Scientific Computing* 92.3 (July 2022), p. 88. DOI: 10.1007/s10915-022-01939-z.

[86] Yiping Lu et al. “Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. *Proceedings of Machine Learning Research*. PMLR, July 2018, pp. 3276–3285. DOI: 10.48550/arXiv.1710.10121.

[87] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[88] Masaya Yamaguti and Hiroshi Matano. “Euler’s finite difference scheme and chaos”. In: *Proceedings of the Japan Academy, Series A, Mathematical Sciences* 55.3 (1979), pp. 78–80. DOI: 10.3792/pjaa.55.78.

[89] Linda Petzold. “Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations”. In: *SIAM Journal on Scientific and Statistical Computing* 4.1 (1983), pp. 136–148. DOI: 10.1137/0904010.

[90] Ernst Hairer, Gerhard Wanner, and Syvert P. Nørsett. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd ed. Vol. 8. *Springer Series in Computational Mathematics*. Springer Berlin, Heidelberg, 1993, pp. XV, 528. DOI: 10.1007/978-3-540-78862-1.

[91] Rick Salmon. “Hamiltonian Fluid Mechanics”. In: *Annual Review of Fluid Mechanics* 20.Volume 20, 1988 (1988), pp. 225–256. DOI: 10.1146/annurev.fl.20.010188.001301.

[92] Fritz Rohrlich. “Relativistic Hamiltonian dynamics I. Classical mechanics”. In: *Annals of Physics* 117.2 (1979), pp. 292–322. DOI: https://doi.org/10.1016/0003-4916(79)90357-9.

[93] Barry Simon. *Quantum Mechanics for Hamiltonians Defined as Quadratic Forms*. Vol. 72. *Princeton Series in Physics*. Princeton: Princeton University Press, 1971. DOI: 10.1515/9781400868834.

[94] Romeo Ortega et al. "Euler-Lagrange systems". In: *Passivity-based Control of Euler-Lagrange Systems: Mechanical, Electrical and Electromechanical Applications*. London: Springer London, 1998, pp. 15–37. DOI: 10.1007/978-1-4471-3603-3_2.

[95] Steven L. Brunton and J. Nathan Kutz. "Promising directions of machine learning for partial differential equations". In: *Nature Computational Science* 4.7 (July 2024), pp. 483–494. DOI: 10.1038/s43588-024-00643-2.

[96] Hao Xu and Dongxiao Zhang. "Robust Discovery of Partial Differential Equations in Complex Situations". In: *Phys. Rev. Res.* 3 (3 Sept. 2021), p. 033270. DOI: 10.1103/PhysRevResearch.3.033270.

[97] Darrell Whitley. "A Genetic Algorithm Tutorial". In: *Statistics and Computing* 4.2 (June 1994), pp. 65–85. DOI: 10.1007/BF00175354.

[98] D. J. Korteweg and G. de Vries. "XLI. On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 39.240 (1895), pp. 422–443. DOI: 10.1080/14786449508620739.

[99] Harry Bateman. "Some Recent Researches on the Motion of Fluids". In: *Monthly Weather Review* 43.4 (1915), pp. 163–170. DOI: 10.1175/1520-0493(1915)43<163:SRROTM>2.0.CO;2.

[100] Miles Cranmer et al. "Discovering Symbolic Models From Deep Learning with Inductive Biases". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. *NIPS '20*. Vancouver, BC, Canada: Curran Associates Inc., 2020. DOI: 10.5555/3495724.3497186.

[101] Franco Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[102] Francisco Villaescusa-Navarro et al. "The Quijote Simulations". In: *The Astrophysical Journal Supplement Series* 250.1 (Aug. 2020), p. 2. DOI: 10.3847/1538-4365/ab9d82.

[103] Alan A. Kaptanoglu et al. “Benchmarking Sparse System Identification with Low-Dimensional Chaos”. In: *Nonlinear Dynamics* 111.14 (July 2023), pp. 13143–13164. DOI: 10.1007/s11071-023-08525-4.

[104] G. Kronberger et al. *Symbolic Regression*. CRC Press, 2024.

[105] Xiaogang Su, Xin Yan, and Chih-Ling Tsai. “Linear regression”. In: *WIREs Computational Statistics* 4.3 (2012), pp. 275–294. DOI: 10.1002/wics.1198.

[106] Xu Chen, Shuo Liu, and Xuan Di. “A Hybrid Framework of Reinforcement Learning and Physics-Informed Deep Learning for Spatiotemporal Mean Field Games”. In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. *AAMAS ’23*. , London, United Kingdom, International Foundation for Autonomous Agents and Multiagent Systems, 2023, pp. 1079–1087. DOI: 10.5555/3545946.3598748.

[107] Jean-Michel Lasry and Pierre-Louis Lions. “Mean Field Games”. In: *Japanese Journal of Mathematics* 2.1 (Mar. 2007), pp. 229–260. DOI: 10.1007/s11537-007-0657-8.

[108] Alena Shilova et al. *Learning HJB Viscosity Solutions with PINNs for Continuous-Time Reinforcement Learning*. Tech. rep. RR-9541. Inria Lille - Nord Europe, CRIStAL - Centre de Recherche en Informatique, Signal et Automatique de Lille - UMR 9189 ; Univ. Lille, CNRS, Centrale Lille, Inria UMR 9189 - CRIStAL,INRIA Lille Nord Europe, Villeneuve d’Ascq, France ; Univ. Grenoble Alps, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France, Feb. 2024, pp. 1–30.

[109] Shangding Gu et al. “A Review of Safe Reinforcement Learning: Methods, Theory and Applications”. In: *ArXiv Preprint ArXiv:2205.10330* (2024). DOI: 10.48550/arXiv.2205.10330.

[110] Zhuoyuan Wang and Yorie Nakahira. “A Generalizable Physics-informed Learning Framework for Risk Probability Estimation”. In: *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*. Ed. by Nikolai Matni, Manfred Morari, and George J. Pappas. Vol. 211. *Proceedings of Machine Learning Research*. PMLR, June 2023, pp. 358–370. DOI: 10.48550/arXiv.2407.08868.

[111] Tianqiao Zhao, Jianhui Wang, and Meng Yue. "A Barrier-Certificated Reinforcement Learning Approach for Enhancing Power System Transient Stability". In: *IEEE Transactions on Power Systems* 38.6 (2023), pp. 5356–5366. DOI: 10.1109/TPWRS.2022.3233770.

[112] Adithya Ramesh and Balaraman Ravindran. "Physics-Informed Model-Based Reinforcement Learning". In: *ArXiv Preprint ArXiv:2212.02179* (2023). DOI: 10.48550/arXiv.2212.02179.

[113] Xin-Yang Liu and Jian-Xun Wang. "Physics-Informed Dyna-Style Model-Based Deep Reinforcement Learning for Dynamic Control". In: *Proceedings of the Royal Society A* 477 (2021). DOI: 10.1098/rspa.2021.0618.

[114] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[115] Miguel Angel Zamora Mora et al. "PODS: Policy Optimization via Differentiable Simulation". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. *Proceedings of Machine Learning Research*. PMLR, July 2021, pp. 7805–7817.

[116] Michael Lutter et al. "Differentiable Physics Models for Real-world Offline Model-based Reinforcement Learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE Press, 2021, pp. 4163–4170. DOI: 10.1109/ICRA48506.2021.9561805.

[117] Jie Xu et al. *Accelerated Policy Learning with Parallel Differentiable Simulation*. 2022. DOI: 10.48550/arXiv.2204.07137.

[118] Rushiv Arora, Eliot Moss, and Bruno Castro da Silva. "Model-Based Reinforcement Learning with SINDy". In: *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*. 2022. DOI: 10.48550/arXiv.2208.14501.

[119] Nicholas Zolman et al. "SINDy-RL: Interpretable and Efficient Model-Based Reinforcement Learning". In: *ArXiv Preprint ArXiv:2403.09110* (2024). DOI: 10.48550/arXiv.2403.09110.

[120] Mikel Landajuela et al. “Discovering symbolic policies with deep reinforcement learning”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. *Proceedings of Machine Learning Research*. PMLR, July 2021, pp. 5979–5989.

[121] Soo Kyung Kim et al. “Discovering Symbolic Policy for Building Control using Reinforcement Learning”. In: *IFAC-PapersOnLine* 56.2 (2023). 22nd IFAC World Congress, pp. 1522–1527. DOI: 10.1016/j.ifacol.2023.10.1848.

[122] Jiaming Guo et al. “Efficient Symbolic Policy Learning With Differentiable Symbolic Expression”. In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. *NIPS ’23*. New Orleans, LA, USA: Curran Associates Inc., 2024. DOI: 10.5555/3666122.3667696.

[123] Wenqing Zheng et al. “Symbolic Learning to Optimize: Towards Interpretability and Scalability”. In: *International Conference on Learning Representations*. 2022. DOI: 10.48550/arXiv.2203.06578.

[124] Michael Herman et al. “Inverse Reinforcement Learning with Simultaneous Estimation of Rewards and Dynamics”. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. *Proceedings of Machine Learning Research*. Cadiz, Spain: PMLR, May 2016, pp. 102–110. DOI: 10.48550/arXiv.1604.03912.

[125] Maxime Bassenne and Adrián Lozano-Durán. “Computational model discovery with reinforcement learning”. In: *ArXiv Preprint ArXiv:2001.00008* (2019). DOI: 10.48550/arXiv.2001.00008.

[126] Mengge Du, Yuntian Chen, and Dongxiao Zhang. “DISCOVER: Deep Identification of Symbolically Concise Open-Form Partial Differential Equations via Enhanced Reinforcement Learning”. In: *Phys. Rev. Res.* 6 (1 Feb. 2024), p. 013182. DOI: 10.1103/PhysRevResearch.6.013182.

[127] Mingsheng Yin et al. “Zero-Shot Wireless Indoor Navigation through Physics-Informed Reinforcement Learning”. In: *ArXiv Preprint ArXiv:2306.06766* (2023). DOI: 10.48550/arXiv.2306.06766.

[128] Amartya Mukherjee and Jun Liu. “Bridging Physics-Informed Neural Networks with Reinforcement Learning: Hamilton-Jacobi-Bellman Proximal Policy Optimization (HJBPPO)”. In: *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*. 2023. DOI: 10.48550/arXiv.2302.00237.

[129] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. "Revisiting Design Choices in Proximal Policy Optimization". In: *ArXiv Preprint ArXiv:2009.10897* (2020). DOI: 10.48550/arXiv.2009.10897.

[130] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. DOI: 10.48550/arXiv.1506.02438.

[131] E. L. Lehmann and George Casella. "Mean Squared Error". In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 337–339. DOI: 10.1007/978-0-387-32833-1_251.

[132] M. A. Biot. "Lagrangian Thermodynamics of Heat Transfer in Systems Including Fluid Motion". In: *Journal of the Aerospace Sciences* 29.5 (1962), pp. 568–577. DOI: 10.2514/8.9559.

[133] Andrew Bennett. *Lagrangian Fluid Dynamics*. *Cambridge Monographs on Mechanics*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511734939.

[134] G. Kalman. "Lagrangian Formalism in Relativistic Dynamics". In: *Phys. Rev.* 123 (1 July 1961), pp. 384–390. DOI: 10.1103/PhysRev.123.384.

[135] Karl Friedrich Siburg. *The Principle of Least Action in Geometry and Dynamics*. 1st ed. Vol. 1844. *Lecture Notes in Mathematics*. Springer Berlin, Heidelberg, 2004, pp. XII, 132. DOI: 10.1007/978-3-540-40985-4.

[136] Anatole Katok and Boris Hasselblatt. *Introduction to the Modern Theory of Dynamical Systems*. *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1995. DOI: 10.1017/CBO9780511809187.

[137] Christine Allen-Blanchette et al. "LagNetViP: A Lagrangian Neural Network for Video Prediction". In: *ArXiv Preprint ArXiv:2010.12932* (2020). DOI: 10.48550/arXiv.2010.12932.

[138] Shuangshuang Wu et al. "Dynamic Modeling of Robotic Manipulator via an Augmented Deep Lagrangian Network". In: *Tsinghua Science and Technology* 29.5 (2024), pp. 1604–1614. DOI: 10.26599/TST.2024.9010011.

[139] Manuel A. Roehrl et al. “Modeling System Dynamics with Physics-Informed Neural Networks Based on Lagrangian Mechanics”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 9195–9200. DOI: 10.1016/j.ifacol.2020.12.2182.

[140] Ziming Yan and Yan Xu. “Real-Time Optimal Power Flow: A Lagrangian Based Deep Reinforcement Learning Approach”. In: *IEEE Transactions on Power Systems* 35.4 (2020), pp. 3270–3273. DOI: 10.1109/TPWRS.2020.2987292.

[141] Hailong Zhang et al. “A Deep Reinforcement Learning-Based Energy Management Framework With Lagrangian Relaxation for Plug-In Hybrid Electric Vehicle”. In: *IEEE Transactions on Transportation Electrification* 7.3 (2021), pp. 1146–1160. DOI: 10.1109/TTE.2020.3043239.

[142] Michael Lutter, Kim Listmann, and Jan Peters. “Deep Lagrangian Networks for End-to-End Learning of Energy-based Control for Under-Actuated Systems”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 7718–7725. DOI: 10.1109/IROS40897.2019.8968268.

[143] Adithya Ramesh and Balaraman Ravindran. “Physics-Informed Model-Based Reinforcement Learning”. In: *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*. Ed. by Nikolai Matni, Manfred Morari, and George J. Pappas. Vol. 211. *Proceedings of Machine Learning Research*. PMLR, June 2022, pp. 26–37. DOI: 10.48550/arXiv.2212.02179.

[144] Maurice A de Gosson and Basil Hiley. *The Principles of Newtonian and Quantum Mechanics*. 2nd. World Scientific, 2017. DOI: 10.1142/10307.

[145] Shanshan Xiao, Jiawei Zhang, and Yifa Tang. “Generalized Lagrangian Neural Networks”. In: *ArXiv Preprint ArXiv:2401.03728* (2024). DOI: 10.48550/arXiv.2401.03728.

[146] Ivo Grondman et al. “A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307. DOI: 10.1109/TSMCC.2012.2218595.

[147] Jason Ansel et al. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and*

Operating Systems, Volume 2 (ASPLOS '24). ACM, Apr. 2024. DOI: 10.1145/3620665.3640366.

[148] Martín Abadi et al. *TensorFlow, Large-scale Machine Learning on Heterogeneous Systems*. Nov. 2015. DOI: 10.5281/zenodo.4724125.

[149] James Bradbury et al. *JAX: Composable Transformations of Python+NumPy Programs*. Version 0.3.13. 2018.

[150] Gao Huang et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[151] Anusha Nagabandi et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 7559–7566. DOI: 10.1109/ICRA.2018.8463189.

[152] Mansfield Merriman. *A list of writings relating to the method of least squares, with historical and critical notes.* From the Transactions of the Connecticut Academy, v.4, 1877. New Haven: Kessinger Publishing, 1877, [151]–232 p.

[153] Charles Dugas et al. "Incorporating second-order functional knowledge for better option pricing". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. *NIPS'00*. Denver, CO: MIT Press, 2000, pp. 451–457. DOI: 10.5555/3008751.3008817.

[154] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *ArXiv* abs/1607.06450 (2016). DOI: 10.48550/arXiv.1607.06450.

[155] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313.

[156] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *ArXiv Preprint ArXiv:1905.11946* (2020). DOI: 10.48550/arXiv.1905.11946.

[157] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training". In: *ArXiv Preprint ArXiv:2104.00298* (2021). DOI: 10.48550/arXiv.2104.00298.

[158] Jie Hu et al. "Squeeze-and-Excitation Networks". In: *ArXiv Preprint ArXiv:1709.01507* (2019). DOI: 10.48550/arXiv.1709.01507.

[159] A Kragten. “Basic Knowledge About Electrical, Chemical, Mechanical, Potential and Kinetic Energy to Understand Literature About the Generation of Energy by Small Wind Turbines”. In: *KD 378* (2008).

[160] R. F. Snider. “Conversion between kinetic energy and potential energy in the classical nonlocal Boltzmann equation”. In: *Journal of Statistical Physics* 80.5 (Sept. 1995), pp. 1085–1117. DOI: 10.1007/BF02179865.

[161] C. A. Coulson and R. P. Bell. “Kinetic Energy, Potential Energy and Force in Molecule Formation”. In: *Trans. Faraday Soc.* 41 (0 1945), pp. 141–149. DOI: 10.1039/TF9454100141.

[162] Kazuyuki Hara, Daisuke Saitoh, and Hayaru Shouno. “Analysis of Dropout Learning Regarded as Ensemble Learning”. In: *Artificial Neural Networks and Machine Learning – ICANN 2016*. Ed. by Alessandro E.P. Villa, Paolo Masulli, and Antonio Javier Pons Rivero. Cham: Springer International Publishing, 2016, pp. 72–79. DOI: 10.1007/978-3-319-44781-0_9.

[163] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: *Proceedings of the First International Workshop on Multiple Classifier Systems*. *MCS ’00*. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 1–15. DOI: 10.5555/648054.743935.

[164] Rincy Thomas and Roopam Gupta. “Ensemble Learning Techniques and its Efficiency in Machine Learning: A Survey”. In: *2nd International Conference on Data, Engineering and Applications (IDEA)*. IEEE. Feb. 2020, pp. 1–6. DOI: 10.1109/IDEA49133.2020.9170675.

[165] Richard S. Sutton. “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming”. In: *Machine Learning Proceedings 1990*. Ed. by Bruce Porter and Raymond Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224. DOI: 10.1016/B978-1-55860-141-3.50030-4.

[166] Guanlin Wu et al. “Dyna-PPO Reinforcement Learning with Gaussian Process for the Continuous Action Decision-making in Autonomous Driving”. In: *Applied Intelligence* 53.13 (July 2023), pp. 16893–16907. DOI: 10.1007/s10489-022-04354-x.

[167] Christof Angermueller et al. “Model-based Reinforcement Learning for Biological Sequence Sesign”. In: *International Conference on Learning Representations*. 2020.

[168] Danijar Hafner et al. “Dream to Control: Learning Behaviors by Latent Imagination”. In: *International Conference on Learning Representations*. 2020. DOI: 10.48550/arXiv.1912.01603.

[169] Danijar Hafner et al. “Mastering Atari with Discrete World Models”. In: *International Conference on Learning Representations*. 2021. DOI: 10.48550/arXiv.2010.02193.

[170] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[171] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. *ICML’10*. Haifa, Israel: Omnipress, 2010, pp. 807–814. DOI: 10.5555/3104322.3104425.

[172] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. DOI: 10.1145/3065386.

[173] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. *NIPS’17*. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. DOI: 10.5555/3295222.3295349.

[174] Lili Chen et al. “Decision Transformer: Reinforcement Learning via Sequence Modeling”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 15084–15097. DOI: 10.48550/arXiv.2106.01345.

[175] Sahika Genc et al. “Zero-Shot Reinforcement Learning with Deep Attention Convolutional Neural Networks”. In: *ArXiv Preprint ArXiv:2001.00605* (2020). DOI: 10.48550/arXiv.2001.00605.

[176] Tomoki Nishi et al. “Traffic Signal Control Based on Reinforcement Learning with Graph Convolutional Neural Nets”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 877–883. DOI: 10.1109/ITSC.2018.8569301.

[177] Penghui Lin et al. “Physics-informed Deep Reinforcement Learning for Enhancement on Tunnel Boring Machine’s Advance Speed and Stability”. In: *Automation in Construction* 158 (2024), p. 105234. DOI: 10.1016/j.autcon.2023.105234.

[178] Derong Liu, Xiaoxu Xiong, and Yi Zhang. “Action-dependent adaptive critic designs”. In: *IJCNN’01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. Vol. 2. 2001, 990–995 vol.2. DOI: 10.1109/IJCNN.2001.939495.

[179] Eric Liang et al. “RLlib: Abstractions for Distributed Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. *Proceedings of Machine Learning Research*. PMLR, July 2018, pp. 3053–3062. DOI: 10.48550/arXiv.1712.09381.

[180] Sergio Guadarrama et al. *TF-Agents: A Library for Reinforcement Learning in TensorFlow*. https://github.com/tensorflow/agents. 2018.

[181] Albert Bou et al. “TorchRL: A Data-Driven Decision-Making Library for PyTorch”. In: *ArXiv* abs/2306.00577 (2023). DOI: 10.48550/arXiv.2306.00577.

[182] Omry Yadan. *Hydra - A Framework for Elegantly Configuring Complex Applications*. Github. 2019.

[183] Omry Yadan, Jasha Sommer-Simpson, and Olivier Delalleau. *omegaconf*. Nov. 2019.

[184] Shubham Pateria et al. “Hierarchical Reinforcement Learning: A Comprehensive Survey”. In: *ACM Comput. Surv.* 54.5 (June 2021). DOI: 10.1145/3453160.

[185] Aishwarya Mandyam et al. “Compositional Q-learning for Electrolyte Repletion with Imbalanced Patient Sub-Populations”. In: *ArXiv* abs/2110.02879 (2024). DOI: 10.48550/arXiv.2110.02879.

[186] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms". In: *Handbook of Reinforcement Learning and Control*. Ed. by Kyriakos G. Vamvoudakis et al. Cham: Springer International Publishing, 2021, pp. 321–384. DOI: 10.1007/978-3-030-60990-0_12.

[187] Hugh Durrant-Whyte, Nicholas Roy, and Pieter Abbeel. "Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts". In: *Robotics: Science and Systems VII*. The MIT Press, 2012, pp. 73–80. DOI: 10.7551/mitpress/9481.003.0015.

[188] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomput.* 503.C (Sept. 2022), pp. 92–108. DOI: 10.1016/j.neucom.2022.06.111.

[189] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. 3. Atlanta, GA. 2013, p. 3.

[190] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *ArXiv* abs/1711.05101 (2019). DOI: 10.48550/arXiv.1711.05101.

[191] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *International Conference on Learning Representations*. 2017. DOI: 10.48550/arXiv.1608.03983.

[192] Zafarali Ahmed et al. "Understanding the Impact of Entropy on Policy Optimization". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. *Proceedings of Machine Learning Research*. PMLR, June 2019, pp. 151–160. DOI: 10.48550/arXiv.1811.11214.

[193] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[194] Nino Vieillard et al. "Leverage the Average: An Analysis of KL Regularization in Reinforcement Learning". In: *Proceedings of the 34th International Con-*

ference on Neural Information Processing Systems. *NIPS '20*. Vancouver, BC, Canada: Curran Associates Inc., 2020. DOI: 10.5555/3495724.3496744.

[195] Răzvan Florian. "Correct Equations for the Dynamics of the Cart-Pole System". In: *ResearchGate* (Aug. 2005).

[196] Jingquan Wang et al. "MBD-NODE: Physics-Informed Data-Driven Modeling and Simulation of Constrained Multibody Systems". In: *Multibody System Dynamics* (July 2024). DOI: 10.1007/s11044-024-10012-6.

[197] Edmund Burke et al. "Hyper-Heuristics: An Emerging Direction in Modern Search Technology". In: *Handbook of Metaheuristics*. Ed. by Fred Glover and Gary A. Kochenberger. Boston, MA: Springer US, 2003, pp. 457–474. DOI: 10.1007/0-306-48056-5_16.

[198] Nikola B. Kovachki, Samuel Lanthaler, and Andrew M. Stuart. "Operator Learning: Algorithms and Analysis". In: *ArXiv* abs/2402.15715 (2024). DOI: 10.48550/arXiv.2402.15715.

[199] Edmund K. Burke et al. "Hyper-Heuristics: A Survey of the State of the Art". In: *Journal of the Operational Research Society* 64.12 (Dec. 2013), pp. 1695–1724. DOI: 10.1057/jors.2013.71.

[200] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. *Proceedings of Machine Learning Research*. PMLR, July 2018, pp. 1861–1870. DOI: 10.48550/arXiv.1801.01290.

[201] Sanghyun Son et al. "Gradient informed proximal policy optimization". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. *NIPS '23*. New Orleans, LA, USA: Curran Associates Inc., 2024. DOI: 10.5555/3666122.3666506.

[202] Mikel Landajuela et al. "Discovering symbolic policies with deep reinforcement learning". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. *Proceedings of Machine Learning Research*. PMLR, July 2021, pp. 5979–5989.

[203] Andrei Kitaitsev and Matteo Manzi. "SymINDy: Symbolic Identification of Nonlinear Dynamics Statement of need". In: *ResearchGate* (Aug. 2022). DOI: 10.13140/RG.2.2.22197.55528.

[204] Pierfrancesco Beneventano et al. “High-Dimensional Approximation Spaces of Artificial Neural Networks and Applications to Partial Differential Equations”. In: *ArXiv* abs/2012.04326 (2020). DOI: 10.48550/arXiv.2012.04326.

[205] John R. Koza. “Genetic programming as a means for programming computers by natural selection”. In: *Statistics and Computing* 4.2 (June 1994), pp. 87–112. DOI: 10.1007/BF00175355.

[206] Grant Dick. “Bloat and Generalisation in Symbolic Regression”. In: *Simulated Evolution and Learning*. Ed. by Grant Dick et al. Cham: Springer International Publishing, 2014, pp. 491–502. DOI: 10.1007/978-3-319-13563-2_42.

[207] Emilio Novati (https://math.stackexchange.com/users/187568/emilio-novati). *Derivative of Vector With Respect To Vector*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2197287 (version: 2017-03-21). 2017.

[208] Wikipedia contributors. *Jacobian Matrix and Determinant — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-August-2024]. 2024.

[209] Won Young Yang et al. “Appendix C: Differentiation with Respect to a Vector”. In: *Applied Numerical Methods Using MATLAB®*. John Wiley & Sons, Ltd, 2005, pp. 471–472. DOI: 10.1002/0471705195.app3.

[210] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *ArXiv* abs/1502.03167 (2015). DOI: 10.48550/arXiv.1502.03167.

[211] Aditya Bhatt et al. “CrossQ: Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency and Simplicity”. In: *The Twelfth International Conference on Learning Representations*. 2024. DOI: 10.48550/arXiv.1902.05605.

[212] Shibani Santurkar et al. “How Does Batch Normalization Help Optimization?” In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. DOI: 10.48550/arXiv.1805.11604.

[213] Li Wan et al. “Regularization of Neural Networks Using DropConnect”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. *ICML’13*. Atlanta, GA, USA: JMLR.org, 2013, III–1058–III–1066. DOI: 10.5555/3042817.3043055.

[214] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018. DOI: 10.48550/arXiv.1801.04381.

[215] Michel Lacroix and Alain Pirotte. “Domain-Oriented Relational Languages”. In: *Proceedings of the Third International Conference on Very Large Data Bases*. Vol. 3. *VLDB ’77*. Tokyo, Japan: VLDB Endowment, 1977, pp. 370–378. DOI: 10.5555/1286580.1286620.

[216] Nan Jiang. “Reinforcement Learning and Monte-Carlo Methods”. In: *University of Illinois Urbana-Champaign* (2021).

[217] Richard Bellman and Stuart Dreyfus. *Dynamic Programming*. Vol. 33. Princeton University Press, 2010. DOI: 10.2307/j.ctv1nxcw0f.

Appendix A

Introduction

A.1 Contributions

In addition to the mentioned contributions, I would personally like to add more details to highlight the complexity relative to my experience:

- **Deep learning** - it was not too difficult for me to design and train neural networks as I am highly familiar with deep learning.
- **Reinforcement Learning** - this was a new area for me. Although I had a university module in RL, the teaching material mostly focused on traditional techniques. Thus, it was quite difficult to familiarize myself with various policy optimization algorithms, such as PPO and Dreamer, from scratch. Many of the concepts I experimented with (for a more general understanding and more general software), e.g., MARL, are not even discussed in this project.
- **Physics** - although the project does not involve many physics concepts, the fundamental ones, such as derivative-based dynamics and Lagrangian mechanics took quite a bit of time to understand and, especially, to apply to code.
- **Software development** - not a big challenge because I have developed Python packages before. However, even though the presented package is not complete, it took a tremendous amount of time to design and debug many components.

Overall, my expertise before starting this project was mainly in software development, computer vision, and deep learning. Therefore, I am definitely satisfied I achieved successful results in this research, whose main focus is far from my comfort zone.

Appendix B

Methodology

B.1 Lagrangian Neural Networks for Control

B.1.1 Lagrangian Equation for Control

We begin by defining the full derivatives of $\frac{d\mathcal{L}}{d\mathbf{q}}$ and $\frac{d\mathcal{L}}{d\dot{\mathbf{q}}}$ to be dependent on $\mathbf{q}$ and $\dot{\mathbf{q}}$, both of which are part of action function $a(\cdot)$ that computes an action $\mathbf{a}$. In other words, instead of being presented with

$$\mathcal{L}(\mathbf{q},\dot{\mathbf{q}}) = T(\mathbf{q},\dot{\mathbf{q}}) - V(\mathbf{q}) \tag{B.1}$$

where $\mathbf{q} \in \mathbb{R}^N, \dot{\mathbf{q}} \in \mathbb{R}^N$ and T, V are kinetic and potential energy functions, we are also given a control action $\mathbf{a} \in \mathbb{R}^M$, or, more specifically, a vector function $\mathbf{a} : \mathbb{R}^{2N} \to \mathbb{R}^M$. It influences the Lagrangian computation, which is approximated by a neural network:

$$\mathcal{L}(\mathbf{q},\dot{\mathbf{q}},\mathbf{a}(\mathbf{q},\dot{\mathbf{q}})) = \mathrm{NN}_{\mathrm{Lagrangian}}(\mathbf{q},\dot{\mathbf{q}},\mathrm{NN}_{\mathrm{Actor}}(\mathbf{q},\dot{\mathbf{q}})) \tag{B.2}$$

By multivariate calculus chain rule and also by knowing that the Jacobian of a vector is a Jacobian matrix (thereby $\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top} \in \mathbb{R}^{M\times N}$, see subsection B.1.2), we have:

$$\begin{aligned}
(\boldsymbol{\mathcal{L}}')^\top = \frac{d\mathcal{L}}{d\mathbf{q}^\top} &= \frac{\partial \mathcal{L}}{\partial \mathbf{q}^\top} + \frac{\partial \mathcal{L}}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top} = \nabla_{\mathbf{q}^\top}\mathcal{L} + \nabla_{\mathbf{a}^\top}\mathcal{L}\nabla_{\mathbf{q}^\top}\mathbf{a} \\
(\boldsymbol{\mathcal{L}}'')^\top = \frac{d\mathcal{L}}{d\dot{\mathbf{q}}^\top} &= \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}^\top} + \frac{\partial \mathcal{L}}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \dot{\mathbf{q}}^\top} = \nabla_{\dot{\mathbf{q}}^\top}\mathcal{L} + \nabla_{\mathbf{a}^\top}\mathcal{L}\nabla_{\dot{\mathbf{q}}^\top}\mathbf{a}
\end{aligned} \tag{B.3}$$

We also know the Euler-Lagrange constraint $\frac{d}{dt}\frac{d\mathcal{L}}{d\dot{\mathbf{q}}^\top} = \frac{d\mathcal{L}}{d\mathbf{q}^\top}$, which we can expand through time using the multivariate chain rule and through Lagrangian using Equation B.3:

$$\frac{\partial \mathcal{L}''}{\partial \mathbf{q}^\top}\frac{\partial \mathbf{q}}{\partial t}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top}\frac{\partial \mathbf{q}}{\partial t}+\frac{\partial \mathcal{L}''}{\partial \dot{\mathbf{q}}^\top}\frac{\partial \dot{\mathbf{q}}}{\partial t}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \dot{\mathbf{q}}^\top}\frac{\partial \dot{\mathbf{q}}}{\partial t}=\mathcal{L}' \tag{B.4}$$

$$\frac{\partial \mathcal{L}''}{\partial \mathbf{q}^\top}\dot{\mathbf{q}}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top}\dot{\mathbf{q}}+\frac{\partial \mathcal{L}''}{\partial \dot{\mathbf{q}}^\top}\ddot{\mathbf{q}}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \dot{\mathbf{q}}^\top}\ddot{\mathbf{q}}=\mathcal{L}' \tag{B.5}$$

$$\left(\frac{\partial \mathcal{L}''}{\partial \mathbf{q}^\top}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top}\right)\dot{\mathbf{q}}+\left(\frac{\partial \mathcal{L}''}{\partial \dot{\mathbf{q}}^\top}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \dot{\mathbf{q}}^\top}\right)\ddot{\mathbf{q}}=\mathcal{L}' \tag{B.6}$$

$$\left(\frac{\partial \mathcal{L}''}{\partial \dot{\mathbf{q}}^\top}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \dot{\mathbf{q}}^\top}\right)^{-1}\left[\mathcal{L}'-\left(\frac{\partial \mathcal{L}''}{\partial \mathbf{q}^\top}+\frac{\partial \mathcal{L}''}{\partial \mathbf{a}^\top}\frac{\partial \mathbf{a}}{\partial \mathbf{q}^\top}\right)\dot{\mathbf{q}}\right]=\ddot{\mathbf{q}} \tag{B.7}$$

In gradient-vector notation and, subsequently, in the notation adopted by the original paper[1] [23], we have:

$$\ddot{\mathbf{q}}=\left(\nabla_{\dot{\mathbf{q}}^\top}\mathcal{L}''+\nabla_{\mathbf{a}^\top}\mathcal{L}''\nabla_{\dot{\mathbf{q}}^\top}\mathbf{a}\right)^{-1}\left[\mathcal{L}'-\left(\nabla_{\mathbf{q}^\top}\mathcal{L}''-\nabla_{\mathbf{a}^\top}\mathcal{L}''\nabla_{\mathbf{q}^\top}\mathbf{a}\right)\dot{\mathbf{q}}\right] \tag{B.8}$$

$$\ddot{\mathbf{q}}=\left(\nabla_{\dot{\mathbf{q}}}(\mathcal{L}'')^\top+\nabla_{\dot{\mathbf{q}}}^\top\mathbf{a}\nabla_{\mathbf{a}}(\mathcal{L}'')^\top\right)^{-1}\left[\mathcal{L}'-\left(\nabla_{\mathbf{q}}(\mathcal{L}'')^\top-\nabla_{\mathbf{q}}^\top\mathbf{a}\nabla_{\mathbf{a}}(\mathcal{L}'')^\top\right)\dot{\mathbf{q}}\right] \tag{B.9}$$

Now, to save space, let's redefine $\mathcal{L}'$ and $\mathcal{L}''$ to be equal to their transposes and also in more compact authors' notation

$$\begin{aligned}\mathcal{L}'&=\left(\nabla_{\mathbf{q}}\mathcal{L}+\nabla_{\mathbf{q}}^\top\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}\right)^\top\\ \mathcal{L}''&=\left(\nabla_{\dot{\mathbf{q}}}\mathcal{L}+\nabla_{\dot{\mathbf{q}}}^\top\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}\right)^\top\end{aligned} \tag{B.10}$$

Now we can get rid of $\mathcal{L}'$ term, which would now be $(\mathcal{L}')^\top$, in Equation B.9 and rewrite it in its final form

$$\ddot{\mathbf{q}}=\left(\nabla_{\dot{\mathbf{q}}}\mathcal{L}''+\nabla_{\dot{\mathbf{q}}}^\top\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}''\right)^{-1}\left[\nabla_{\mathbf{q}}\mathcal{L}+\nabla_{\mathbf{q}}^\top\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}-\left(\nabla_{\mathbf{q}}\mathcal{L}''-\nabla_{\mathbf{q}}^\top\mathbf{a}\nabla_{\mathbf{a}}\mathcal{L}''\right)\dot{\mathbf{q}}\right] \tag{B.11}$$

where:

- $\mathbf{q},\ \dot{\mathbf{q}},\ \ddot{\mathbf{q}}\in\mathbb{R}^{N\times 1}$, $\mathbf{a}\in\mathbb{R}^{M\times 1}$
- $\mathcal{L}\in\mathbb{R}$
- $\nabla_{\dot{\mathbf{q}}}\mathcal{L}\in\mathbb{R}^{N\times 1}$, $\nabla_{\mathbf{a}}\mathcal{L}\in\mathbb{R}^{M\times 1}$
- $\nabla_{\mathbf{q}}\mathbf{a},\ \nabla_{\dot{\mathbf{q}}}\mathbf{a}\in\mathbb{R}^{M\times N}$
- $\mathcal{L}''\in\mathbb{R}^{1\times N}$
- $\nabla_{\mathbf{q}}\mathcal{L}'',\ \nabla_{\dot{\mathbf{q}}}\mathcal{L}''\in\mathbb{R}^{N\times N}$, $\nabla_{\mathbf{a}}\mathcal{L}''\in\mathbb{R}^{M\times N}$

[1] It is not very clear what notation they choose for $\nabla_{\mathbf{x}}\mathbf{f}$, thus we use a commonly accepted Jacobian matrix notation which results in $\mathbb{R}^{M\times N}$ for $\mathbf{f}:\mathbb{R}^N\to\mathbb{R}^M$ and $\mathbf{x}\in\mathbb{R}^N$. For a scalar function f, i.e., when $M=1$, it is clear that the authors choose a column-vector as a result, i.e., $\mathbb{R}^{N\times 1}$.

B.1.2 Vector Differentiation

Assume, we have a function $f : \mathbb{R}^N \to \mathbb{R}$ producing a scalar $s \in \mathbb{R}$ and two vectors $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$. Differentiating a scalar function with respect to a vector and vice versa is easy - we know that the result is a Jacobian vector.

Instead of defining the differential operator as $\frac{d\cdot}{d\cdot}$, let's just say that it is some operator $\cdot$ op $\cdot$ with the constraint that the term on the left can only interact with the term on the right when the transition is smooth. In other words, much like we cannot do a dot product between $\mathbf{a}^\top \in \mathbb{R}^{1\times 2}$ and $B \in \mathbb{R}^{3\times 2}$, we also cannot do $\mathbf{a}^\top$ op B due to shape mismatch. This constraint ensures consistency as we scale from scalars to tensors.

Now we can define vector Jacobians in the following way, understanding that the scalar can be expressed as a vector of size 1, i.e., $\mathbf{s} \in \mathbb{R}^1$:

$$\frac{d\mathbf{s}}{d\mathbf{v}^\top} = \nabla_{\mathbf{v}^\top} s = \begin{pmatrix} \frac{\partial s}{\partial v_1} & \cdots & \frac{\partial s}{\partial v_N} \end{pmatrix} \qquad \frac{d\mathbf{v}}{d\mathbf{s}} = \nabla_s \mathbf{v} = \begin{pmatrix} \frac{\partial v_1}{\partial s} \\ \vdots \\ \frac{\partial v_N}{\partial s} \end{pmatrix} \tag{B.12}$$

Note how $\frac{d\mathbf{s}}{d\mathbf{v}}$ would be illegal according to our constraint, however, people simply define $\frac{ds}{d\mathbf{v}} \in \mathbb{R}^N$ which, instead of representing matrix differentiation, corresponds to element-wise differentiation.

From here we can interpolate that differentiating a vector with respect to another vector results in a matrix, commonly known as Jacobian matrix [207, 208, 209]. More precisely, given a vector function $\mathbf{f} : \mathbb{R}^N \to \mathbb{R}^K$, we have that

$$\frac{d\mathbf{f}}{d\mathbf{x}^\top} = \nabla_{\mathbf{x}^\top} \mathbf{f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_K}{\partial x_1} & \cdots & \frac{\partial f_K}{\partial x_N} \end{bmatrix} \tag{B.13}$$

Jacobian matrix, when the function maps to a scalar, i.e., when $M = 1$, reduces to the row vector $\nabla_{\mathbf{x}^\top} f$. Now assume $\mathbf{f}' : \mathbb{R}^K \to \mathbb{R}^N$ is a first-order derivative of $\mathbf{x}$ that can be further differentiated with respect to $\mathbf{y}$. We have:

$$\frac{d\mathbf{f}'}{d\mathbf{y}^\top} = \nabla_{\mathbf{y}^\top} \mathbf{f}' = \nabla_{\mathbf{y}^\top} \nabla_{\mathbf{x}^\top}^\top f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial y_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial y_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial y_1} & \cdots & \frac{\partial^2 f}{\partial x_N \partial y_N} \end{bmatrix} \tag{B.14}$$

Note that authors in the original paper [23] define the transpose of it as $\nabla_{\mathbf{y}} \nabla_{\mathbf{x}}^\top f = \left(\nabla_{\mathbf{y}^\top} \nabla_{\mathbf{x}^\top}^\top f \right)^\top$. Although this notation is more compact, by swapping rows with columns, the chain rule for matrices becomes especially hard to understand.

B.1.3 LNNc Block Architecture Details

We did not expand a lot on why we use specific layers and architecture choices when we presented the LNNc structure. This is because we simply reuse the components rather than introducing something novel, thus we limited the discussion beyond mentioning the papers for a more curious reader (they're ideas are well-known and commonly used in deep learning architectures). Some details still need a few more comments:

- **Normalization** - we use layer normalization [154] instead of batch normalization [210] because the distribution of encountered states shifts as the actor learns which actions to take. Relying on minibatch statistics would be especially hurtful for on-policy learning. Layer normalization, on the other hand, normalizes across features for each individual data point. There are techniques specific to reinforcement learning like CrossQ [211], however, for simplicity, we use what is already available in PyTorch. Normalization guides the optimization faster by smoothing the loss hypersurface [212].
- **Softplus** - as noted by Cranmer et al. [23], we cannot use ReLU because the gradients are shut down. It is not an issue for regular neural networks, however, we need the gradients to compute Jacobians and Hessians used to infer the next state. We would further like to emphasize that this criteria also extends to actor network because, in our case, we differentiate through actor as well. We found it more performant to use hyperbolic tangent activation for actor networks rather than softplus. Activations are needed to introduce nonlinearity between layers.
- **Dropout** - dropout [155] improves generalization by making hidden units less dependent on each other and can even be perceived as a type of ensemble technique [162]. To further enhance generalization, we use the L2 penalty [213] via AdamW [190]. Generalization is necessary for a dynamics model since it is used for planning and dreaming, and we need to ensure the model generalizes to the best of its ability to unseen states.
- **Mobile-block** - as mentioned before, we implement squeeze-and-excitation [158] with residual connections [87]. This is similar to inverted residuals [214], however, our design choice is simpler since we are not dealing with convolutional layers. The core motivation is to enhance features while preserving the gradient flow.

That being said, please refer to the original code, more specifically we construct `BlockNet` and `EnsembleBlockNet` classes with the components mentioned above.

B.2 Dreaming Proximal Policy Optimization

B.2.1 Get, Sample, and Pairs

Although it should be straightforward what `get`, `sample`, and `pairs` mean from the context, we utilize domain relational calculus queries [215] to formalize the definitions. To keep notations simpler, we do not specify which kinds of elements the operations are performed over - it should be clear from the returned value. For example, instead of defining `get_o`$(\cdot)$ for elements of type o and `get_rc`$(\cdot)$ for tuples of elements of types r and c, just a single `get` is used and it will be clear from the retrieved value over which kinds of objects the function was performed.

We define $y_1, \cdots, y_F$ to be a sequence of object kinds that we want to retrieve (free elements), and $x_1, ..., x_N$ to be a sequence of object kinds that we do not care about. $F+N$ is each tuple's size in a queried set (usually a data buffer $\mathcal{D}$). Furthermore, each object is equipped with attributes t and b, representing timestep and batch indices.

Given a data buffer set $\mathcal{D}$, method `get` returns the element(-s) with matching attributes t and b, method `sample` returns a random element(-s), and method `pairs` returns a random pair(-s) of consecutive elements:

$$\text{get}(\mathcal{D}, t, b) := \left\{ \left(y_{1,b}^{(t)}, \cdots, y_{F,b}^{(t)}\right) \;\middle|\; \exists_{x_1,\cdots,x_N} \left(x_1, \cdots, x_N, y_{1,b'}^{(t')}, \cdots, y_{F,b'}^{(t')}\right) \in \mathcal{D} \wedge t' = t \wedge b' = b \right\} \tag{B.15}$$

$$\text{sample}(\mathcal{D}) := \left\{ \left(y_{1,b}^{(t)}, \cdots, y_{F,b}^{(t)}\right) \;\middle|\; t \sim \text{Uniform}(1,T) \wedge b \sim \text{Uniform}(1,B) \wedge \left(\exists_{x_1,\cdots,x_N} \left(x_1, \cdots, x_N, y_{1,b'}^{(t')}, \cdots, y_{F,b'}^{(t')}\right) \in \mathcal{D} \wedge t' = t \wedge b' = b\right) \right\} \tag{B.16}$$

$$\text{pairs}(\mathcal{D}) := \left\{ \left(\left(y_{1,b}^{(t)}, y_{1,b}^{(t+1)}\right), \cdots, \left(y_{F,b}^{(t)}, y_{F,b}^{(t+1)}\right)\right) \;\middle|\; t \sim \text{Uniform}(1,T) \wedge b \sim \text{Uniform}(1,B) \wedge \left(\exists_{x_1,\cdots,x_N} \left(x_1, \cdots, x_N, y_{1,b'}^{(t')}, \cdots, y_{F,b'}^{(t')}\right) \in \mathcal{D} \wedge t' = t \wedge b' = b\right) \wedge \left(\exists_{x_1,\cdots,x_N} \left(x_1, \cdots, x_N, y_{1,b}^{(t'+1)}, \cdots, y_{F,b}^{(t'+1)}\right) \in \mathcal{D} \wedge t' = t \wedge b' = b\right) \right\} \tag{B.17}$$

Although the operations are defined to return a set, we will only write the returned value, because the returned set only contains a single element or tuple.

B.2.2 Returns and Advantages

Algorithm 8 shows how to compute simple Monte-Carlo values [216]. Specifically, instead of invoking Bellman equations [217], we rely on random trajectory discounted returns to form direct value estimates. It is simple because it does not depend on the size of the state space, a property of Monte-Carlo methods [31]. However, they have high variance and require a lot of data due to stochasticity [38].

Algorithm 8 Monte-Carlo estimation of returns

Require: B, γ ▷ Batch size, discount factor

procedure COMPUTE_RETURNS($\mathcal{D}$)
  $T \leftarrow \frac{1}{B}|\mathcal{D}|$ ▷ Buffer cardinality
  $\mathcal{R} \leftarrow \{R_b^{(T+1)} \leftarrow 0\}_{b=1}^{B}$ ▷ Init returns
  for $(t = T, \cdots, 1) \times (b = 1, \cdots, B)$ **do**
    $(r_b^{(t)}, c_b^{(t)}) \leftarrow \text{get}(\mathcal{D}, t, b)$ ▷ Reward and continue flag
    $R_b^{(t)} \leftarrow r_b^{(t)} + \gamma \cdot c_b^{(t)} \cdot R_b^{(t+1)}$ ▷ Compute return for b at t
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_b^{(t)}\}$ ▷ Add return to returns set
  return $\mathcal{R}$

Algorithm 9 shows how to compute simple advantages from the Monte-Carlo returns. It utilizes (old) critic to make the updates more stable (more gradual distribution shift). Although more sophisticated methods, like GAE [130] exist, we stick with simplicity.

Algorithm 9 Standardized advantage estimation

Require: B, ε ▷ Batch size, epsilon

procedure COMPUTE_ADVANTAGES($\mathcal{D}$, ν_ϕ)
  $T \leftarrow \frac{1}{B}|\mathcal{D}|$ ▷ Buffer cardinality
  $\mathcal{R} \leftarrow \text{compute\_returns}(\mathcal{D})$ ▷ Compute returns
  $\mathcal{A} \leftarrow \{\}$ ▷ Init advantages
  for $(t = 1, \cdots, T) \times (b = 1, \cdots, B)$ **do**
    $o_b^{(t)} \leftarrow \text{get}(\mathcal{D}, t, b)$ ▷ Retrieve observation
    $\tilde{R}_b^{(t)} \leftarrow (R_b^{(t)} - \mathbb{E}[\mathcal{R}]) \div (\sigma(\mathcal{R}) + \varepsilon)$ ▷ Standardize return
    $\mathrm{A}_b^{(t)} \leftarrow \tilde{R}_b^{(t)} - \nu_\phi(o_b^{(t)})$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathrm{A}_b^{(t)}\}$ ▷ Compute advantage
  return $\mathcal{A}$

B.2.3 Collect, Concat, and Extras

Algorithm 10 simply describes how batched environment rollouts are performed:

Algorithm 10 Data collection

procedure COLLECT($\mathcal{A}$, $\mathcal{E}$, T, t_0, B)
  $\mathcal{D}_{\mathcal{E}} = \{(o_b^{(t_0)}, r_b^{(t_0)}, c_b^{(t_0)}) \leftarrow \mathcal{E}.\text{reset}()\}_{b=1}^{B}$ ▷ Initial obs, rews, terms
  $\mathcal{D}_{\mathcal{A}} = \{(a_b^{(t_0)} \leftarrow \mathbf{0} \in \mathbb{R}^B)\}_{b=1}^{B}$ ▷ Initial actions (zeros)
  $\mathcal{D} \leftarrow \text{concat}(\mathcal{D}_{\mathcal{E}}, \mathcal{D}_{\mathcal{A}})$ ▷ Concatenate to form D
  for $(t = t_0, \cdots, t_0 + T) \times (b = 1, \cdots, B)$ **do**
    $a_b^{(t)} \leftarrow \mathcal{A}.\text{act}(o_b^{(t-1)})$ ▷ Act on agent policy
    if $c_b^{(t-1)}$ **then**
      $(o_b^{(t)}, r_b^{(t)}, c_b^{(t)}) \leftarrow \mathcal{E}.\text{step}(a_b^{(t)})$ ▷ If continue, step environment
    else
      $(o_b^{(t)}, r_b^{(t)}, c_b^{(t)}) \leftarrow \mathcal{E}.\text{reset}()$ ▷ Otherwise, sample new state
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_b^{(t)}, a_b^{(t)}, r_b^{(t)}, c_b^{(t)})\}$ ▷ Update data buffer
  return $\mathcal{D}$

We would also like to define a handy `concat` function which simply concatenates corresponding tuples (at t and b) of some two sets $\mathcal{D}$ and $\mathcal{D}'$:

$$\text{concat}(\mathcal{D}, \mathcal{D}') := \Bigg\{ \left(d_{1,b}^{(t)}, \cdots, d_{N,b}^{(t)}, d_{1,b}^{(t)}, \cdots, d_{N',b}^{(t)}\right) \Bigg| \exists_{t',b'} t' \in \{1, \cdots, T\} \wedge b' \in \{1, \cdots B\} \wedge \left(d_{1,b}^{(t)}, \cdots, d_{N,b}^{(t)}\right) \in \mathcal{D} \wedge \left(d_{1,b}^{(t)}, \cdots, d_{N',b}^{(t)}\right) \in \mathcal{D}' \wedge t' = t \wedge b' = b \Bigg\} \tag{B.18}$$

There are a few functions mentioned that do not have clear definitions - they only indicate what the algorithm should be doing. A few more details about these:

- `step` - takes action, steps environment, and returns a next-state tuple
- `reset` - resets environment, and returns a random new-state tuple
- `act` - takes observation and produces an action based on it
- `dynamics` - takes observation and action to produce next observation
- `trim` - takes buffer set and capacity value and returns trimmed buffer
- `optimize`- takes model(-s) and loss(-es), backpropagates, and returns the updated model(-s)

B.2.4 Notations

- $\mathcal{A}$ - agent *or* advantages
- π - policy/actor
- ν - value function/critic
- $\mathcal{E}$ - real environment
- $\mathcal{M}$ - surrogate model
- $\mathcal{S}$ - state model
- $\mathcal{R}$ - reward model *or* returns
- $\mathcal{C}$ - continue model
- $\mathcal{D}$ - data buffer
- $\mathcal{O}$ - optimizer
- $\mathcal{L}$ - loss
- N - total iteration
- T - total timesteps
- B - batch size
- H - horizon
- n - iteration counter
- t - timestep
- b - sample index
- h - horizon step
- o - observation
- a - action
- r - reward
- c - continue
- R - discounted return
- v - value
- A - advantage
- γ - discount factor
- ε - small noise value
- θ - denotation of learnable params
- ϕ - denotation of learnable params
- ψ - denotation of learnable params
- on - denotation of “on-policy”
- off - denotation of “off-policy”

B.3 Athletes Software Package

B.3.1 RLLib's Complexity

For instance, if one would like to create an environment runner that alters between real and surrogate models using Ray `v2.31.0`, then one would have to significantly change the components of RLLib because they highly depend on each other. For instance, RLlib creates multiple environments, which are controlled by an environment runner, which by itself is part of a bigger group. Further, RLLib trains agents via learner groups that contain multi-agent learners. Extending the functionality of some existing class is not a complex task if one is familiar with RLLib's framework, however, implementing a new component, e.g., multi-agent and multi-world hybrid learner requires deep knowledge of how each component utilizes each other. We have managed to implement various features facilitating model-based learning and they can be checked at `src/supplementary/rllib`. However, we had to discard the idea of working with it further because adding anything extra always requires modifying at least a few components, whose changes may propagate to further components making it very time-consuming to develop research-focused code which is mainly experimental and not stable. Still, some useful utilities can still be utilized now or in the future, such as the surrogate switch wrapper:

```
from supplementary.rllib.env.wrappers import SurrogateSwitchWrapper

# Init real env & world model
env = SurrogateSwitchWrapper(
    env=gymnasium.make("CartPole-v1"),
    world_model_config={"cls_name": "MLPWorldModel", "framework": "torch"}
)

# Load weights into the world model & reset
wrapped_env.world_model.load_state_dict(torch.load("path/to/weights.pth"))
env.reset()

# Step through the real environment
action = env.action_space.sample()
state, reward, term, trunc, info = env.step(action)

env.switch() # To surrogate
o, r, t1, t2, i = env.step(action)
env.switch() # To real
```

Listing B.1: A switch wrapper for Gymnasium environments

B.3.2 Example Algorithm Using Athletes

```python
import dataclasses
import torch
from athletes.agents import Agent

class SimplePPO(Agent):
    @dataclasses.dataclass
    class Config[T=PPO](Agent.Config[T]):
        gamma: float = 0.99
        num_iters: int = 30
        lr: float = 1e-4

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        # Create very simple actor and critic models
        self.actor = torch.nn.LazyLinear(self.a_dim)
        self.critic = torch.nn.LazyLinear(1)
        self.optimizer = torch.optim.AdamW([
            {"params": self.actor.parameters(), "lr": self.lr},
            {"params": self.critic.parameters(), "lr": self.lr},
        ])

    def forward(self, observation, action=None):
        # Make dist & use it for extra outputs
        dist = self.dist.make(self.actor(observation))
        action = dist.rsample() if action is None else action
        logprob = dist.log_prob(action.detach())

        return action, self.critic(observation), logprob

    def step(self, observation, reward, terminated, truncated, info):
        # Store in buffer
        self.buffer.append(
            obs=observation, rew=reward, term=terminated, trun=truncated,
            **dict(zip(("act", "val", "logp"), out := self(observation)))
        )

        return out[0]

    def update(self) -> dict[str, float]: ...
        # Implemention of PPO update ...
```

Listing B.2: Simplified PPO using python 3.13 (notice type param default)

B.3.3 Nested Agent Code Snippet

```python
# Default Dyna config but MFA is overridden to be another Dyna agent
agent_config = Dyna.Config(model_free_agent=Dyna.Config())
agent = agent_config()
```

Listing B.3: Agent Instance Example

Appendix C

Analysis

C.1 Default Hyperparameters

These are the default parameters used in the experiments. In some experiments, it is obvious if there are modifications, e.g., `total_steps`, in others, changes are explicitly stated. Please also refer to our code for any minor parameter changes.

Parameter	Default value	Parameter	Default value
max_episode_steps	400	num_envs	32
total_steps	50000	lr_actor	5e-5
report_every	1600	lr_critic	1e-3
num_sgd_iters (world)	2000	lr_dynamics	1e-4
num_sgd_iters (agent)	30	lr_reward	1e-4
batch_size (world)	256	weight_decay	1e-2
batch_size (agent)	*all*	actor_hidden	[64, 64]
num_planning_steps	800	critic_hidden	[64, 64]
planning_batch_size	32	dynamics_hidden	[64, 64]
allow_update_from_real	True	reward_hidden	[64, 64]
activation_actor	Tanh	num_lagrangians	1000
activation_critic	ReLU	is_generalized	False
activation_dynamics	Softplus	gamma	0.99
activation_reward	ReLU	clip_eps	0.2

Table C.1: Default hyperparameters.

C.2 Autograd

C.2.1 Autograd

```python
import torch
from torch.autograd.functional import jacobian, hessian

def jacobian_hessian_naive(func, input):
    jacobians, hessians = [], []

    for sample in input:
        jacobians.append(jacobian(func, sample))
        hessians.append(hessian(func, sample))

    return torch.stack(jacobians), torch.stack(hessians)
```

Listing C.1: Naive computation of Jacobian and Hessian

```python
import torch
from torch.func import jacrev, hessian

def jacobian_hessian_functional(func, input):
    return torch.vmap(jacrev(func))(input), torch.vmap(hessian(func))(input)
```

Listing C.2: Functional computation of Jacobian and Hessian

```python
import torch
from torch.autograd import grad

def jacobian_hessian_autograd(func, input):
   input = input.detach().clone().requires_grad_(True)
   output = func_batched(input)

   jacobian = grad(output, input, torch.ones_like(output), create_graph=True)[0]
   grad_outs = torch.eye(jacobian.size(1))[:, None].repeat(1, jacobian.size(0), 1)
   hessian_T = grad(jacobian, input, grad_outs, is_grads_batched=True)[0]

   return jacobian, hessian_T.transpose(0, 1)
```

Listing C.3: Autograd computation of Jacobian and Hessian

```python
import torch

def func(x: torch.Tensor): # Expected x shape: (D,)
    return (2 * x.pow(3) - x.pow(2)).sum()

def func_batched(x: torch.Tensor): # Expected x shape: (B, D)
    return (2 * x.pow(3) - x.pow(2)).sum(dim=1, keepdim=True)
```

Listing C.4: Example target function