To the Graduate Council:
I am submitting herewith a thesis written by Andrei Liviu Cozma entitled "Koopman-Inspired Proximal Policy Optimization (KIPPO)." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Hairong Qi, Major Professor

We have read this thesis
and recommend its acceptance:

Hairong Qi

Catherine Schuman

Dan Wilson

Amir Sadovnik

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a thesis written by Andrei Liviu Cozma entitled "Koopman-Inspired Proximal Policy Optimization (KIPPO)." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Hairong Qi, Major Professor

We have read this thesis
and recommend its acceptance:

Hairong Qi

Catherine Schuman

Dan Wilson

Amir Sadovnik

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Koopman-Inspired Proximal Policy Optimization (KIPPO)

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Andrei Liviu Cozma

August 2024

Acknowledgements

I wish to express my deep gratitude to my advisor, Dr. Hairong Qi, for her support and guidance throughout this research. Her insights and expertise have been invaluable. I am also grateful to my committee members, Dr. Amir Sadovnik, Dr. Catherine Schuman, and Dr. Dan Wilson, for their constructive feedback and encouragement.

The unwavering support and belief of my parents and friends have been crucial to my resilience and persistence. I also want to acknowledge my colleagues in the Advanced Imaging and Collaborative Information Processing (AICIP) Lab for fostering a collaborative and supportive environment. This journey has taught me much about perseverance and personal growth, and I look forward to the next steps in my career.

Abstract

Reinforcement Learning (RL) has made significant strides in various domains, yet developing effective control policies for environments with complex, nonlinear dynamics remains a challenge, particularly for policy gradient methods. These methods often struggle due to high-variance in gradient estimates, non-convex optimization landscapes, and sample inefficiency, resulting in unstable learning, suboptimal policies, and trade-offs between performance and reproducibility. The quest for more robust, stable, and effective methods has led to numerous innovations and remains a critical area of research. Proximal Policy Optimization (PPO) has gained popularity in recent years due to its balance in performance, training stability, and computational efficiency. In contrast with their nonlinear counterparts, linear systems are simpler, more predictable, and easier to analyze. Koopman Theory has emerged as a powerful framework for studying nonlinear systems through a globally-linear operator that acts on a higher-dimensional space of measurement functions. Combining these two ideas, Koopman-Inspired Proximal Policy Optimization (KIPPO) extends PPO to learn a simplifying representation of the underlying system's dynamics while retaining essential features for effective policy learning. This is achieved through a Koopman-approximation auxiliary network and carefully designed constraints that enable balancing the complexity of latent dynamics. Results demonstrate improvements over the PPO baseline with 8–60% increased performance while reducing variability by up to 91% when evaluated on diverse continuous control tasks. The study also examines the effects and interactions of key hyperparameters and the impacts of individual loss components through an ablation study, providing a comprehensive analysis of the approach.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement Learning (RL) is a powerful framework for sequential decision-making tasks, enabling agents to learn complex behaviors through interaction with their environment. Within this framework, policy optimization aims to find the optimal mapping from states to actions that maximizes the agent's performance over time. Policy gradient methods, commonly used in continuous control tasks, directly optimize the policy to maximize expected returns (Sutton et al., 1999). Despite remarkable success in various domains, developing effective control policies for environments with complex, nonlinear dynamics remains challenging.

The intricate dependencies and long-term relationships in these systems make it difficult to assess the impact of policy updates on long-term performance. This issue, coupled with non-convex optimization landscapes, leads to high variance in gradient estimates, resulting in inconsistent updates and unstable learning. Consequently, the optimization process can diverge or oscillate, preventing smooth convergence to optimal policies. These challenges are further exacerbated by heightened sensitivity to hyperparameters, often necessitating extensive tuning. Moreover, the need for a large number of samples to address high variance and explore the optimization landscape makes the learning process computationally expensive and time-consuming. These interconnected challenges create a fundamental trade-off between performance and reproducibility, motivating the search for innovative solutions.

To address these challenges, researchers have proposed numerous advancements and state-of-the-art algorithms. The actor-critic architecture (Konda and Tsitsiklis, 1999) introduces a critic to guide the actor's policy updates with a more stable signal, thereby lowering variance in gradient estimates and improving sample efficiency. These methods often utilize the advantage function (Schulman et al., 2018) to focus policy updates on the most advantageous actions, further reducing variance. Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) take an additional step by constraining or penalizing excessive policy updates to promote improved training stability and convergence properties (Schulman et al., 2017a,b). In recent years, PPO has gained increased popularity due to its balance of performance, training stability, and computational efficiency.

Current research explores new architectures, innovative training strategies, and insights from other fields, alongside incorporating prior knowledge and domain-specific constraints. For instance, the Stochastic Latent Actor-Critic (SLAC) algorithm employs representation learning alongside a state-space model to effectively handle high-dimensional image inputs (Lee et al., 2020). However, reliance on computationally-intensive variational inference may limit scalability, especially in larger or real-time applications. Model-Based Policy Optimization (MBPO) employs an ensemble of dynamics models to generate synthetic trajectories for augmenting real experience during policy optimization, leading to improved sample efficiency (Janner et al., 2021). Nevertheless, its effectiveness depends on the accuracy of these models, and the computational demands of managing multiple models could constrain practical use. Robust Policy Optimization (RPO) extends PPO by introducing a perturbed action distribution to enhance exploration and mitigate premature convergence to suboptimal policies (Rahman and Xue, 2022). However, excessive exploration may slow down convergence to a stable policy if not carefully balanced.

The field of dynamical systems involves studying mathematical models that describe how processes evolve over time, focusing on their patterns, stability, and long-term behavior (Broer and Takens, 2010; Heij et al., 2021). While linear systems are simpler, more predictable, and easier to analyze and control, many real-world systems are nonlinear, exhibiting complex behaviors where small changes in initial conditions can lead to drastically

different outcomes (Mezic and Runolfsson, 2004). This poses significant challenges for estimation, prediction, and control.

Koopman Operator Theory has emerged as a powerful tool for studying nonlinear systems. It aims to find a linearized description of a nonlinear system's dynamics by lifting the state-space to a higher-dimensional space of measurement functions, known as the Koopman observable space (Koopman, 1931; Brunton et al., 2021). This process maps original state variables to observable functions, extracting useful information about the system's state. The Koopman operator, an infinite-dimensional linear operator, evolves these observables linearly in time, providing a linear description of the system even if the underlying dynamics are nonlinear. Data-driven methods like Dynamic Mode Decomposition (DMD) and recent advances in deep learning have enabled obtaining Koopman observables directly from data, bridging the gap between theoretical analysis and practical applications (Schmid, 2010; Yeung et al., 2017; Lusch et al., 2018).

Recent works have explored integrating Koopman theory with optimal control and RL. Several data-driven approaches use deep neural networks to learn Koopman operators and embedding functions, effectively learning a model of the system which can then be used to apply traditional control strategies like Linear Quadratic Regulator (LQR), Model Predictive Control (MPC), and other model-based techniques (Shi and Meng, 2022; Han et al., 2020; Yin et al., 2022). Deep Koopman Reinforcement Learning (DKRL) incorporates local Koopman operators and Gaussian Mixture Model (GMM) clustering to improve data efficiency and performance in robotics control tasks (Song et al., 2021). Koopman Forward (Conservative) Q-Learning (KFC) enhances offline RL by identifying and utilizing system symmetries for data augmentation (Weissenbacher et al., 2022). These approaches highlight the potential of Koopman-based methods to drive innovative solutions for control strategies, paving the way for future advancements and integrations.

Building upon these foundations, this thesis introduces Koopman-Inspired Proximal Policy Optimization (KIPPO), a novel framework that extends PPO with Koopman-inspired representation learning. While Koopman operator theory seeks a perfectly linear representation of a nonlinear system, this approach relaxes the strict linearity requirement.

Instead, it leverages an approximation of the Koopman operator that seeks to simplify the underlying system's nonlinear dynamics while retaining essential features for effective policy learning. This is achieved through a Koopman-approximation auxiliary network and carefully designed constraints that enable balancing the complexity of latent dynamics. By utilizing these techniques, KIPPO harnesses the flexibility of deep learning and insights from Koopman operator theory to streamline and harmonize the system's behavior for improved policy performance.

The main contributions of this study are summarized as follows:

1. Introduce the KIPPO framework, which integrates a Koopman-inspired auxiliary network, objectives, and constraints into PPO to learn a latent-space representation that enables managing the inherent complexity of navigating nonlinear environments.

2. Demonstrate improvements over the baseline on diverse continuous control tasks from the MuJoCo and Box2D benchmarking environments, showcasing improvements in performance ranging from 6% to 60% and reduction in variability from 26% to 91% (previewed in Figure 1.1).

3. Analyze the effects of key hyperparameters and conduct an ablation study to assess the individual contributions of loss function components, offering insights into the framework's robustness and adaptability.

4. Discuss the potential of Koopman-inspired representation learning to alleviate the challenges faced by policy gradient methods, and explore potential limitations and future research directions.

This thesis is organized as follows:

- Chapter 2 introduces dynamical systems, Koopman theory, and reinforcement learning. It explores recent developments, challenges, and the emerging intersection of Koopman theory with reinforcement learning.

- Chapter 3 presents the proposed framework's architecture, objectives, and integration with PPO. It emphasizes the multi-objective optimization for Koopman-inspired representations and the decoupled learning approach.

- Chapter 4 outlines the experimental setup and presents a comprehensive comparison with the baseline. It includes an analysis of key hyperparameters, an ablation study, and an exploration of hyperparameter interactions.

- Chapter 5 summarizes the study's contributions and discusses the implications of the results. It outlines future research directions and reflects on the broader impacts of this approach in reinforcement learning.

- Appendix A provides implementation details and pseudocode for the proposed framework. It offers technical insights for researchers interested in replicating or extending this work.

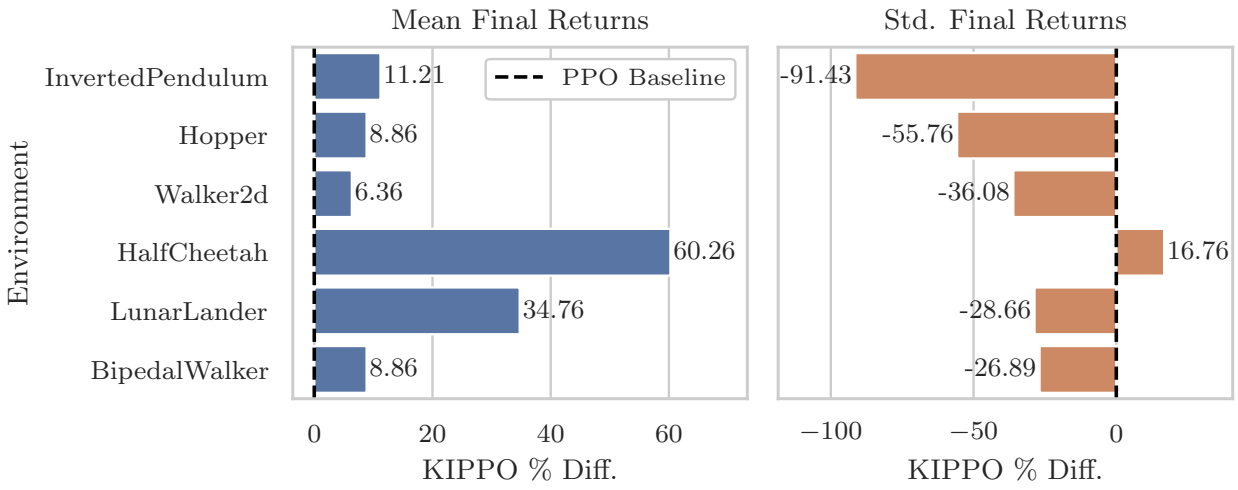

Figure 1.1: Preview of KIPPO's improvements relative to the PPO baseline in terms of average performance and variability across four trials per environment.

Chapter 2

Background and Related Works

Reinforcement Learning (RL) offers a powerful framework for developing agents capable of learning complex behaviors. However, the inherent nonlinearities prevalent in real-world environments pose significant challenges to the design of effective and efficient algorithms for this learning paradigm. This chapter establishes the foundational context for understanding these challenges and motivates the exploration of Koopman-inspired policy optimization, the novel approach presented in this thesis.

The chapter unfolds by examining the following key areas:

1. Section 2.1 introduces dynamical systems and control theory, the mathematical frameworks underpinning the modeling and regulation of systems that evolve over time. This section underscores the complexities associated with analyzing and controlling systems, particularly those characterized by nonlinear dynamics.

2. Section 2.2 presents Koopman operator theory, a mathematical tool that offers a principled approach to analyzing nonlinear systems through the lens of linear systems. This linear perspective provided by Koopman theory opens up new avenues for analysis and control, potentially mitigating the challenges inherent in working with nonlinear dynamics.

3. Section 2.3 delves into the core principles of RL, a learning paradigm in which agents learn optimal behaviors through interactions with their environment. This section emphasizes the connections between this paradigm, dynamical systems, and control theory, laying the groundwork for exploring how these fields can be synthesized to address the challenges of learning in nonlinear environments.

4. Section 2.4 explores recent advancements in policy optimization. This section highlights promising alternative approaches and examines the emerging intersection of Koopman theory and RL, suggesting a direction for developing more robust and efficient learning algorithms.

2.1 Dynamical Systems and Control Theory

Dynamical systems and control theory provide the foundational framework for understanding and manipulating systems that evolve over time. This section introduces these fundamental concepts, emphasizing the distinctions between linear and nonlinear systems and their implications for analysis and control. The section begins with a general overview of dynamical systems, followed by a deeper exploration of linear and nonlinear dynamics. It then delves into system identification and control techniques, highlighting the challenges posed by nonlinearity. Finally, the section discusses the broad applications of these concepts and outlines related research directions.

2.1.1 General Overview

Dynamical systems are mathematical frameworks used to describe how systems evolve over time (Broer and Takens, 2010; Brunton and Kutz, 2022; Devaney, 2021; Freidlin et al., 2012). They capture the relationship between a system's current state, represented by a set of variables, and its future states. This evolution is governed by rules, typically expressed as differential equations for continuous-time systems or difference equations for discrete-time

systems (Heij et al., 2021). These equations, representing the system's dynamics, dictate how the state changes in response to internal interactions and external influences.

The power of dynamical systems lies in their ability to model a wide array of phenomena across diverse fields. In physics, they describe the motion of celestial bodies, the behavior of fluids, and the interactions of particles. In biology, they model population dynamics, neural networks, and the spread of diseases. In engineering, they are indispensable for designing aircraft, controlling robots, and optimizing chemical processes.

2.1.2 Linear and Nonlinear Dynamics

Dynamical systems can be broadly categorized into two fundamental types: linear and nonlinear (Broer and Takens, 2010; Brunton and Kutz, 2022; Devaney, 2021). This distinction, rooted in the mathematical properties of the governing equations, has profound implications for the behavior of the systems and the techniques used to analyze and control them.

Linear Systems

Linear systems are governed by linear relationships between their state variables and their rates of change. This linearity leads to several desirable properties that make these systems relatively easy to analyze and control.

One fundamental property is the principle of superposition, which states that the response of a linear system to a sum of inputs is equal to the sum of its responses to each input individually. This property allows for the decomposition of complex inputs into simpler components, simplifying analysis.

Another advantage of linear systems is the frequent availability of analytical solutions. These closed-form solutions, often derived using techniques like matrix exponentials for Linear Time-Invariant (LTI) systems, provide a complete description of the system's behavior for any given initial condition and input.

Furthermore, linear systems can be conveniently represented and analyzed using matrix algebra. This representation allows for the leveraging of powerful tools from linear algebra to study the system's stability, controllability, and other important properties. For example, a discrete-time LTI system can be represented as:

$$x_{t+1} = \mathbf{C}x_t, \tag{2.1}$$

where x_t is the state vector at time t, and $\mathbf{C}$ is the system matrix that describes the dynamics.

A common example of a linear system is a simple harmonic oscillator, like a mass attached to a spring. The system's state, described by the mass's position and velocity, evolves according to linear differential equations.

Nonlinear Systems

Nonlinear systems, in contrast to their linear counterparts, are characterized by nonlinear relationships between their state variables. This nonlinearity, while making these systems more challenging to analyze, also endows them with a richness of behavior not observed in linear systems.

One key distinction is that nonlinear systems do not obey the principle of superposition. The response to a combination of inputs cannot be predicted by simply summing up the responses to each input individually. This behavior arises from the nonlinear interactions between state variables.

Furthermore, nonlinear systems can exhibit chaotic behavior. In such systems, even tiny differences in initial conditions can lead to vastly different outcomes over time, making long-term prediction extremely difficult.

Unlike linear systems, which typically have a single equilibrium point, nonlinear systems can have multiple equilibria. These points represent states where the system tends to settle, and the system's behavior can change dramatically depending on which equilibrium point it converges to.

A general representation of a discrete-time nonlinear system is:

$$x_{t+1} = \mathbf{F}(x_t), \tag{2.2}$$

where $\mathbf{F}$ is a nonlinear function mapping the current state to the next state.

Examples of nonlinear systems are abundant in the real world, with the weather being a prime example. The equations governing fluid flow in the atmosphere are inherently nonlinear, leading to the chaotic and unpredictable nature of weather patterns.

2.1.3 System Identification and Control

System identification and control are two fundamental tasks in the study of dynamical systems (Brunton and Kutz, 2022; Miranda, 2015; Morari and H. Lee, 1999). System identification aims to build mathematical models of systems from observed data, while control theory seeks to design strategies to manipulate them to achieve desired behaviors.

System identification is the process of deducing the rules governing a system's behavior from its observed inputs and outputs. For linear systems, this often involves estimating the parameters of a linear model that best fits the data. Techniques like least squares and subspace identification methods are commonly used for this purpose.

Nonlinear system identification, however, is considerably more challenging. The absence of the superposition principle and the potential for complex behaviors make it difficult to find general-purpose methods that work well across different nonlinear systems. Approaches like nonlinear regression, neural networks, and Gaussian process models have been used with some success, but finding accurate and computationally tractable models for nonlinear systems remains an active area of research.

Control theory deals with the design of strategies to influence the behavior of dynamical systems through the application of external inputs (Brunton and Kutz, 2022; Miranda, 2015). The goal is to drive the system towards a desired state or trajectory, often while optimizing for certain performance criteria.

Linear control theory, thanks to the predictability and well-understood properties of linear systems, offers a rich set of tools for designing effective control strategies (Brunton and Kutz, 2022). Techniques like Proportional-Integral-Derivative (PID) control, state-feedback control, and optimal control techniques, such as Linear Quadratic Regulator (LQR) (Scokaert and Rawlings, 1998), have been successfully applied in countless engineering applications. For instance, the state-space representation of a linear system with control inputs is given by:

$$x_{t+1} = \mathbf{C}x_t + \mathbf{D}u_t, \tag{2.3}$$

where $u_t \in \mathbb{R}^m$ is the input vector and $\mathbf{D} \in \mathbb{R}^{n \times m}$ is the input matrix.

Nonlinear control, however, poses significant challenges. The lack of general analytical solutions and the potential for complex behaviors necessitate the development of more sophisticated techniques. A general form for a controlled nonlinear system is:

$$x_{t+1} = \mathbf{F}(x_t, u_t), \tag{2.4}$$

where $\mathbf{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ represents the system dynamics under the influence of control inputs.

One powerful technique for nonlinear control is Model Predictive Control (MPC) (Holkar and Waghmare, 2010; Lee, 2011; Morari and H. Lee, 1999; Qin and Badgwell, 1997). This method uses a model of the system to predict its future behavior over a finite time horizon and then optimizes a sequence of control actions to minimize a cost function while satisfying system constraints. This optimization is performed at each time step, taking into account the current state of the system and any updated predictions. While computationally demanding, MPC's ability to handle constraints and its predictive nature make it well-suited for nonlinear systems.

2.1.4 Summary

Dynamical systems and control theory provide a powerful framework for understanding and manipulating systems that evolve over time. This section has explored the fundamental concepts, highlighting the distinctions between linear and nonlinear systems and their implications for analysis and control.

Linear systems, characterized by their adherence to the principle of superposition, offer analytical tractability and a rich set of tools for analysis and control. Their behavior can be fully described using matrix algebra, enabling the application of well-established techniques such as LQR and state-feedback control.

In contrast, nonlinear systems exhibit a broader range of behaviors, including multiple equilibria and chaos. While this complexity makes them more challenging to analyze and control, it also allows them to model a wider array of real-world phenomena. Techniques for nonlinear system identification and control, such as nonlinear regression and MPC, have been developed to address these challenges.

The field of system identification bridges the gap between observed data and mathematical models, with approaches ranging from linear parameter estimation to complex nonlinear techniques involving neural networks and Gaussian processes. Control theory builds upon these models to design strategies for manipulating system behavior, with methods spanning from classical PID control to advanced nonlinear control techniques.

As research in this field progresses, new approaches continue to emerge, often leveraging advances in machine learning and data-driven methods. These developments aim to address the ongoing challenges in modeling and controlling complex nonlinear systems, with potential applications across a wide range of disciplines, from engineering and physics to biology and economics.

The concepts and techniques discussed in this section form the foundation for understanding more advanced topics in dynamical systems and control, including the Koopman operator theory and its applications in reinforcement learning, which will be explored in subsequent sections of this thesis.

2.2 Koopman Operator Theory

Analyzing and controlling nonlinear systems often necessitates confronting their intricate and potentially chaotic nature. Koopman Operator Theory offers an alternative perspective by enabling the analysis of these systems through a linear framework. Rather than directly analyzing the system's state, Koopman theory focuses on the evolution of measurements of the state, known as observables. This shift in perspective allows for the potential application of tools from linear systems theory to understand and control nonlinear systems, offering a path towards simplifying the challenges faced by traditional methods.

2.2.1 Core Concepts and Formulations

Central to Koopman theory is the Koopman operator, denoted by $\mathcal{K}$. This infinite-dimensional linear operator acts on observable functions, rather than on the state itself. An observable $g : \mathcal{S} \to \mathbb{R}$ maps the system's state to a scalar value, capturing relevant information about its behavior. The power of the Koopman operator lies in its ability to evolve these measurements linearly over time, even when the underlying system dynamics are nonlinear.

Formally, the Koopman operator is defined as:

$$\mathcal{K}g(x_k) := g(\mathbf{F}(x_k)) = g(x_{k+1}), \tag{2.5}$$

where $\mathbf{F} : \mathcal{S} \to \mathcal{S}$ represents the nonlinear state transition function. Equation (2.5) demonstrates that the operator $\mathcal{K}$ maps an observable g evaluated at the current state x_k to the same observable evaluated at the next state x_{k+1}. This effectively allows for analyzing the system's evolution by examining the linear dynamics of the chosen measurements.

Koopman eigenfunctions play a crucial role in linearizing and decomposing dynamics (Lan and Mezić, 2013; Budišić et al., 2012). An observable g is a Koopman eigenfunction if it satisfies:

$$\mathcal{K}g(x) = \lambda g(x), \tag{2.6}$$

where λ is the corresponding eigenvalue. Analogous to eigenvectors in linear algebra, Koopman eigenfunctions provide a set of coordinates in which the nonlinear system dynamics appear linear. Each eigenfunction, representing a specific measurement or combination of measurements, evolves independently over time according to its eigenvalue, simplifying the analysis and prediction of the system's behavior (Lan and Mezić, 2013; Budišić et al., 2012).

In practical applications, the infinite-dimensional Koopman operator is often approximated by a finite-dimensional matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$. This corresponds to choosing a finite set of measurement functions to represent the system's dynamics. A vector-valued function $g : \mathcal{S} \to \mathbb{R}^m$ then represents these observables, effectively lifting the state-space to a higher-dimensional space where the dynamics are approximately linear:

$$\begin{aligned} g(x_{t+1}) &= \mathbf{K} g(x_t) \\ y_{t+1} &= \mathbf{K} y_t, \end{aligned} \tag{2.7}$$

where $y_k = g(x_k) \in \mathbb{R}^m$ is the vector of observables at time k. Equation (2.7) closely resembles the state-space representation of an LTI system (Eq. (2.1)), but it's important to note that the "state" in this case is not the original state of the system, but rather a vector of measurements of that state.

Extending this representation to systems with control inputs involves incorporating a control term that governs how the choice of control actions influences the evolution of the measurements:

$$\begin{aligned} g(x_{t+1}) &= \mathbf{K} g(x_t) + \mathbf{B} f(u_t) \\ y_{t+1} &= \mathbf{K} y_t + \mathbf{B} v_t, \end{aligned} \tag{2.8}$$

where $\mathbf{B} \in \mathbb{R}^{m \times k}$ is the control matrix, and $f : \mathcal{A} \to \mathbb{R}^k$ maps control inputs to the space of control observables. This formulation, similar to Eq. (2.3), highlights the potential of Koopman theory for controlling nonlinear systems by leveraging linear control techniques in the lifted space of observables (Proctor et al., 2016a,b).

2.2.2 Data-Driven and Modern Methods

While Koopman operator theory provides valuable theoretical insights, practical applications often require learning appropriate observables directly from data. Data-driven methods aim to uncover these representations without prior knowledge of the system's underlying equations, addressing the challenge of finding suitable observables that accurately capture system dynamics (Brunton et al., 2021).

Dynamic Mode Decomposition (DMD) is a prominent technique that analyzes complex system dynamics by decomposing them into spatiotemporal modes (Schmid, 2010; Kutz et al., 2016; Williams et al., 2015). This method approximates the Koopman operator by analyzing sequential snapshots of system measurements, offering a linear representation of underlying nonlinear dynamics. Extensions like Dynamic Mode Decomposition with Control (DMDc) incorporate control inputs, broadening its applicability to control-oriented scenarios (Proctor et al., 2016a). Limitations of DMD include difficulty in capturing transient behaviors and handling irregularly sampled data.

To address these challenges, researchers have turned to deep learning approaches (Lusch et al., 2018; Yeung et al., 2017; Dey and Davis, 2023). Two primary deep learning architectures have emerged for Koopman analysis. The first is deep auto-encoders, which extract key latent variables to parameterize the dynamics, offering a low-dimensional latent space that potentially leads to more interpretable solutions. These can be viewed as nonlinear generalizations of DMD. The second approach involves high-dimensional lifting, where the input data is lifted to an even higher dimension where the evolution is approximately linear. Both architectures enforce linear dynamics on the latent variables, aligning with Koopman theory principles.

Deep learning methods offer advantages over traditional DMD, including the ability to handle irregularly sampled data and capture more complex nonlinear dynamics. However, they also introduce challenges such as increased computational requirements and the need for larger datasets. To facilitate the adoption of these methods, software libraries such as

PyDMD, PyKoopman, and DLKoopman have been developed (Demo et al., 2018; Pan et al., 2023; Dey and Davis, 2023).

2.2.3 Summary

Koopman operator theory offers a powerful framework for analyzing and controlling nonlinear dynamical systems through a linear lens. By focusing on the evolution of observable functions rather than the system state itself, Koopman theory enables the application of linear systems tools to nonlinear problems, potentially simplifying their analysis and control.

The core of this approach is the Koopman operator, an infinite-dimensional linear operator that evolves observable functions over time. Koopman eigenfunctions play a crucial role in this framework, providing a set of coordinates in which nonlinear dynamics appear linear. In practice, the Koopman operator is often approximated by a finite-dimensional matrix, allowing for tractable computations and analysis.

Data-driven methods have emerged to address the challenge of finding suitable observables directly from data, without prior knowledge of the system's underlying equations. These methods, including DMD and deep learning approaches, enable system identification for nonlinear systems. This capability is particularly valuable to enable the use of model-based methods, which require access to a model of the system dynamics.

The application of Koopman theory extends beyond model-based planning and control. By providing a linear representation of nonlinear dynamics, it opens up possibilities for improved state estimation, prediction, and system analysis. This linear framework may also offer insights into system behavior, stability analysis, and the design of more effective control strategies.

While Koopman theory provides valuable insights and tools for analyzing nonlinear systems, practical applications still face challenges such as selecting appropriate observables and balancing computational requirements with accuracy. Ongoing research in this field continues to expand the applicability of Koopman-based methods to increasingly complex systems and control scenarios.

The concepts and techniques discussed in this section lay the groundwork for understanding how Koopman theory can be integrated with reinforcement learning approaches, as will be explored in subsequent sections. This integration has the potential to enhance both model-based and model-free reinforcement learning algorithms by leveraging the linearizing representation of system dynamics provided by the Koopman framework.

2.3 Reinforcement Learning

Reinforcement Learning (RL) provides a framework for agents to learn optimal behaviors through trial-and-error interactions with their environment, aiming to discover effective strategies over time. This learning paradigm is particularly well-suited for complex, dynamic environments where pre-programmed solutions are impractical or infeasible. This section introduces the core principles of RL, including the agent-environment interaction loop, the concepts of rewards and policies, and the goal of maximizing cumulative reward. It highlights the inherent connections between this approach, dynamical systems, and control theory, suggesting a promising direction for developing more sophisticated and effective algorithms.

2.3.1 Core Concepts and Formulations

Central to RL is the interaction between an agent and its environment, formalized as a Markov Decision Process (MDP). This interaction loop mirrors the feedback loop in control systems: the agent functions as the controller, the environment represents the system being controlled, and the reward signal provides feedback. Richard Bellman's foundational work on dynamic programming and MDPs paved the way for developing RL algorithms (Bellman, 1954; Bellman et al., 1957; Bellman, 1957).

Within an MDP, an agent operates within a state-space $\mathcal{S}$, encompassing all possible environment states. At each timestep t, the agent observes the current state $x_t \in \mathcal{S}$ and selects an action $u_t \in \mathcal{A}$, where $\mathcal{A}$ represents the set of all possible actions. This action influences the environment, causing a transition to a new state x_{t+1} governed by

the state transition dynamics, $P(x_{t+1}|x_t, u_t)$. This probability distribution captures the environment's inherent stochasticity and the uncertainty in action outcomes. Subsequently, the environment provides the agent with a reward r_t, a scalar feedback signal indicating the desirability of the agent's action in the current state.

Encoding the learning goal, the reward function R depends on the current state, action, and subsequent state: $r_t = R(x_t, u_t, x_{t+1})$. In practice, however, the reward function is often simplified to depend solely on the current state, $r_t = R(x_t)$, or the current state-action pair, $r_t = R(x_t, u_t)$.

Through its interactions, the agent generates a trajectory τ, a sequence of states, actions, and rewards:

$$\tau = (x_0, u_0, r_0, x_1, u_1, r_1, \ldots, x_T, u_T, r_T). \tag{2.9}$$

Trajectories form the fundamental unit of experience in this domain, providing the data necessary for learning and policy improvement. A policy π governs the agent's behavior, mapping states to a probability distribution over actions: $u_t \sim \pi(\cdot|x_t)$. Essentially, the policy represents the agent's decision-making strategy. The probability of a trajectory τ occurring under a policy π can be calculated as follows:

$$P(\tau|\pi) = \rho_0(x_0) \prod_{t=0}^{T-1} P(x_{t+1}|x_t, u_t)\pi(u_t|x_t). \tag{2.10}$$

This calculation considers the initial state distribution $\rho_0(x_0)$, the state transition probabilities $P(x_{t+1}|x_t, u_t)$, and the policy's action probabilities $\pi(u_t|x_t)$.

The overarching goal of an agent is to learn an optimal policy π^* that maximizes the expected cumulative reward, or expected return, over trajectories (Sutton and Barto, 2018). One standard formulation is the discounted return:

$$G = R(\tau) = \sum_{t=0}^{T} \gamma^t r_t. \tag{2.11}$$

This formulation sums the rewards obtained at each time step, weighted by the discount factor γ raised to the power of the time step. This discount factor balances the immediate and future consequences of the agent's actions.

The expected return $J(\pi)$ under a policy π represents the average return over all possible trajectories generated by following that policy:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)]. \tag{2.12}$$

Maximizing the expected return translates to finding a policy that achieves the highest average performance across all potential realizations of the agent-environment interaction. This optimization problem lies at the core of RL:

$$\pi^* = \mathop{\text{argmax}}_{\pi} J(\pi). \tag{2.13}$$

Solving this optimization problem in complex environments poses significant challenges. High-dimensional state and action spaces, nonlinear dynamics, and long-term dependencies complicate the process. The following subsections delve into algorithmic approaches to these challenges and the evolution of policy optimization methods in this area.

2.3.2 General Classifications of Algorithms

Value-Based vs Policy-Based Methods

RL algorithms can be distinguished by their focus on estimating action and state values or directly learning behavior policies.

Value-based methods learn functions estimating expected returns:

1. State-value function: $V^{\pi}(x)$, estimating the expected return from state x following policy π.

2. Action-value function: $Q^{\pi}(x, u)$, estimating the expected return from taking action u in state x and following policy π.

These methods are often sample-efficient, leveraging past experiences to update multiple value estimates. They excel in discrete action spaces but may struggle with continuous ones, often requiring discretization.

Policy-based methods directly learn a policy function $\pi(u|x)$ mapping states to action probabilities or deterministic actions. They handle both discrete and continuous action spaces and excel in high-dimensional action spaces. However, they may require more samples and can suffer from high variance in gradient estimates.

Both approaches aim to find optimal policies, but they differ in their optimization process. Value-based methods indirectly improve the policy by refining value estimates, while policy-based methods directly optimize the policy. This distinction affects their performance in different scenarios, with value-based methods often being more stable but potentially suboptimal in some cases, and policy-based methods being more flexible but potentially more difficult to train.

On-Policy vs Off-Policy Learning

On-policy algorithms learn from experiences generated by the current policy being optimized. They adapt well to non-stationary environments and are preferred in applications where minimizing exploration risks is crucial. However, they may be sample-inefficient and limit exploration.

Off-policy algorithms can learn from experiences generated by any policy, including past versions or demonstrations. They offer improved sample efficiency and allow more extensive state space exploration. However, they may suffer from instability, especially with significant mismatches between the data-generating policy and the policy being improved.

Both approaches aim to improve the policy, but they differ in how they utilize data. On-policy methods ensure consistency between the learned policy and the data-generating policy, leading to more stable learning. Off-policy methods offer greater flexibility in data use but may require more complex techniques to ensure stability.

Model-Based vs Model-Free Methods

Model-based algorithms learn an explicit model of the environment's dynamics, capturing state transition and reward functions. They offer better sample efficiency and planning capabilities by leveraging the learned model to reason about action consequences without direct environment interaction. However, their effectiveness depends on model accuracy, which can be challenging to achieve in complex environments.

Model-free methods directly learn value functions or policies from environment interactions without an explicit model. They are often simpler to implement and more robust to model misspecification, making them flexible in handling complex or poorly understood environments. However, they typically require more direct interactions to learn effective policies and may struggle in scenarios where long-term planning is crucial.

Model-based methods can potentially learn more efficiently by leveraging their understanding of the environment, while model-free methods are more adaptable to unknown or changing environments. The choice between these methods often involves a trade-off between sample efficiency and robustness to model errors.

The selection among these classifications depends on various factors including environment complexity, data availability, computational resources, and specific task requirements. Many algorithms combine elements from different classifications to leverage their respective strengths while mitigating limitations.

2.3.3 Evolution of Policy Gradient Methods

Policy gradient methods represent a cornerstone in RL, offering a direct approach to optimizing parameterized policies. These methods have undergone substantial evolution since their inception, driven by persistent challenges of sample efficiency, stability, and performance in complex environments. This evolutionary journey reflects broader progress in the field, showcasing how theoretical insights and practical innovations have shaped modern algorithms.

This subsection traces the development of policy gradient methods, examining key milestones and algorithmic advancements. The discussion begins with foundational vanilla policy gradients, progresses through the integration of value functions in actor-critic methods, and culminates in the development of trust region and proximal policy optimization techniques. Each stage addresses specific limitations of its predecessors, expanding the capabilities of learning agents and pushing the boundaries of achievable performance.

This overview sets the stage for understanding how Koopman-inspired approaches can potentially address persistent challenges in policy optimization, particularly in environments with complex, nonlinear dynamics.

Vanilla Policy Gradients

Vanilla policy gradient methods, such as REINFORCE (Williams, 1992), directly optimize a parameterized policy $\pi_\theta(u|x)$ by performing gradient ascent on the expected return $J(\pi_\theta)$. This relies on the policy gradient theorem (Sutton et al., 1999), which establishes a link between the gradient of the expected return and the policy parameters. REINFORCE updates these parameters using:

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)|_{\theta_k} \,, \tag{2.14}$$

where the policy gradient is computed using the policy gradient theorem:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(u_t|x_t) R(\tau) \right]. \tag{2.15}$$

This represents the expected value over all trajectories of the sum of log-probabilities of actions, each weighted by the discounted return obtained after taking that action.

While straightforward, vanilla policy gradient methods suffer from high variance in gradient estimates, leading to unstable learning and often poor convergence, especially in complex environments with long-term dependencies. This high variance stems from basing gradient estimates on potentially long trajectories with noisy cumulative rewards. This noise, amplified over updates, can cause policy oscillations or divergence. Furthermore, the

non-convex optimization landscape can trap these methods in local optima, highlighting the need for better exploration strategies or transformations of the optimization landscape. Their reliance on Monte Carlo estimates for the expected return also results in low sample efficiency, requiring extensive data collection and careful hyperparameter tuning.

Actor-Critic Methods

Actor-critic methods, such as the Advantage Actor-Critic (A2C) algorithm (Mnih et al., 2016), address the high variance and sample inefficiency of vanilla policy gradients by combining value-based and policy-based learning. They simultaneously optimize a policy (actor) and a value function (critic), which approximates the expected return for a given state or state-action pair, providing a lower-variance target for policy updates. This bootstrapping from value estimates contributes to more stable learning (as the critic learns from shorter horizons of experience, reducing variance in gradient estimates) and improves sample efficiency.

To further reduce variance and improve learning, actor-critic methods often employ the advantage function $A^\pi(x, u)$, which quantifies the relative benefit of taking a specific action u in a given state x compared to following the current policy π:

$$A^\pi(x, u) = Q^\pi(x, u) - V^\pi(x). \tag{2.16}$$

However, actor-critic methods face challenges in continuous action spaces. The high dimensionality of continuous action spaces makes exploration and optimization more challenging, as effectively exploring the action space becomes crucial for finding optimal policies. Deterministic Policy Gradient (DPG) algorithms, such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019), address this by working with deterministic policies in continuous action spaces and employing techniques like target networks and experience replay to stabilize learning and improve sample efficiency. Building on these ideas, Twin Delayed DDPG (TD3) (Fujimoto et al., 2018) introduces twin critic networks and delayed policy updates to mitigate overestimation bias. Another advancement is Soft Actor-Critic

(SAC) (Haarnoja et al., 2018), an off-policy actor-critic algorithm that incorporates entropy maximization to encourage the policy to remain stochastic, promoting exploration and preventing premature convergence.

Despite these advancements, learning accurate value estimates and stable policies in environments with highly nonlinear dynamics remains challenging. The non-stationarity of target values in these environments, coupled with the challenges of function approximation, can lead to instability and slow convergence. This sensitivity to nonlinearity motivates exploring Koopman theory, as its ability to linearize dynamics could simplify the learning problem and lead to more stable and efficient actor-critic methods.

Trust Region Methods and Proximal Policy Optimization

Recognizing the instability that can arise from unconstrained policy updates, researchers developed trust region methods like Trust Region Policy Optimization (TRPO) (Schulman et al., 2017a). These methods stabilize policy learning by constraining the size of policy updates, ensuring the new policy remains within a trusted region around the old policy where performance improvements are more predictable. This constraint, typically defined in terms of the Kullback-Leibler (KL) divergence between the policies, limits information loss and prevents drastic changes that could lead to instability. However, this stability comes at the cost of increased computational complexity, as TRPO relies on a second-order optimization approach to solve the constrained optimization problem.

Proximal Policy Optimization (PPO) (Schulman et al., 2017b) offers a more computationally efficient alternative to TRPO while maintaining similar performance and stability. Instead of directly constraining the policy update, PPO employs a clipped surrogate objective that penalizes large deviations from the old policy:

$$\boldsymbol{\mathcal{L}}(x, u, \theta_k, \theta) = \min\left(\frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)} A^{\pi_{\theta_k}}(x, u), \quad g(\epsilon, A^{\pi_{\theta_k}}(x, u))\right). \tag{2.17}$$

Here, θ_k represents the policy parameters from the previous iteration, and θ denotes the current policy parameters. ϵ is a hyperparameter controlling the clipping range. $g(\epsilon, A)$ is a clipping function defined as:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{2.18}$$

This clipping mechanism in PPO acts as a regularizer, preventing drastic policy changes between updates and promoting smoother learning dynamics. Unlike TRPO, PPO relies on first-order optimization, making it simpler to implement and computationally more efficient, especially for large-scale problems and high-dimensional action spaces.

PPO has gained popularity as one of the most successful policy optimization algorithms due to its simplicity, stability, and strong performance across various tasks. Its ability to handle continuous action spaces effectively, coupled with its relatively low computational cost and robustness to hyperparameter settings, has made it a popular choice for research and practical applications.

Despite significant progress, the inherent complexity of learning to navigate complex environment with nonlinear dynamics continues to pose challenges for policy gradient methods. While these fundamental challenges cannot be entirely eliminated, the quest for more robust, performant, and stable algorithms continues to motivate research into innovative approaches that can further enhance learning capabilities.

2.3.4 Summary

This section examined the evolution of RL, focusing on policy optimization methods and their limitations in complex environments. The progression from vanilla policy gradients to advanced techniques like actor-critic and trust region methods reveals a consistent effort to address several fundamental challenges:

1. **High Variance and Non-Convex Optimization:** While methods like PPO have improved stability, they still struggle with highly nonlinear dynamics and long-term

dependencies. This suggests a need for more robust optimization techniques that can navigate complex loss landscapes effectively.

2. **Bias-Variance and Exploration-Exploitation Trade-offs:** Balancing these trade-offs remains challenging, especially in complex environments. Effective methods must navigate the bias-variance trade-off in value function approximation while also managing the exploration-exploitation dilemma to ensure both efficient learning and optimal performance.

3. **Sample Efficiency:** Off-policy methods and model-based approaches have improved data utilization, but sample efficiency remains a bottleneck for many real-world applications. This highlights the importance of developing methods that can learn effectively from limited data.

4. **Robustness and Generalization:** Current algorithms often lack robustness to environmental changes and struggle to generalize across diverse tasks. This limitation underscores the need for methods that can learn more transferable and adaptable policies.

These persistent challenges indicate that while RL has made significant strides, there remains a considerable gap between current capabilities and the requirements of complex, real-world applications.

2.4 Recent Developments and Integrations

The field of RL continues to evolve, driven by the pursuit of algorithms capable of handling complex environments and tasks. Recent advancements have focused on addressing key challenges in policy optimization, such as sample efficiency, robustness, and the ability to handle high-dimensional state spaces. These developments have led to innovative approaches that combine elements from different paradigms or integrate techniques from other fields of study.

This section explores recent developments in policy optimization, highlighting alternative approaches that address limitations of traditional methods. Additionally, it examines the emerging intersection of Koopman theory and RL, where the linearization capabilities of Koopman theory are leveraged to potentially enhance how agents learn in complex, nonlinear environments.

2.4.1 Integrations with Insights from Other Fields

Recent developments in RL have incorporated insights and techniques from various related fields, leading to novel approaches that address specific challenges in policy optimization.

Stochastic Latent Actor-Critic (SLAC) integrates policy optimization with representation learning by employing a sequential latent variable model and variational inference (Lee et al., 2020). This approach learns a compressed state-space representation, reducing dimensionality and improving performance and sample efficiency, particularly in high-dimensional continuous control tasks. The compressed representation facilitates policy learning by focusing on the most relevant features of the state. However, the variational inference step in SLAC can be computationally intensive, potentially limiting its scalability to very large state spaces.

Model-Based Policy Optimization (MBPO) combines model-based planning with policy optimization, leveraging the strengths of both approaches (Janner et al., 2021). It learns an ensemble of dynamics models to generate synthetic experience, augmenting the agent's collected real experience. This data augmentation technique improves sample efficiency, as the agent can learn from a wider range of potential outcomes, including those not encountered in the real environment. The effectiveness of MBPO depends on the accuracy of the learned dynamics models; inaccurate models can lead to biased policy updates and suboptimal performance, highlighting the importance of accurate model learning in this approach.

Robust Policy Optimization (RPO) extends PPO by introducing a perturbed action distribution during training (Rahman and Xue, 2022). This perturbation encourages the agent to maintain higher entropy in its policy, promoting better exploration of the

environment. Unlike traditional methods where policy stochasticity often reduces as training progresses, RPO aims to maintain a certain level of entropy throughout the training period. This approach helps prevent premature convergence to local optima and potentially improves robustness across different environments. By leveraging perturbed distributions, it provides a way to better represent the action space and reduce the need for extensive hyperparameter tuning. However, the increased exploration can sometimes lead to slower initial convergence, as the agent spends more time exploring diverse actions before settling on an effective policy.

2.4.2 Integrations with Koopman Theory

The integration of Koopman theory with control and RL offers a new approach to address challenges posed by nonlinear dynamics in complex environments. By leveraging Koopman theory's ability to represent nonlinear systems in a linear framework, researchers are exploring ways to enhance learning algorithms and control techniques.

This integration has developed along two main paths. The first approach focuses on using Koopman theory for system modeling and control. Research has shown progress in learning approximate Koopman models and applying classical control techniques such as LQR and MPC in the resulting linear space of observables (Han et al., 2020; Shi and Meng, 2022). Building on this work, further studies have explored integrating Koopman theory with the LQR formulation to develop differentiable policy representations that embed optimal control principles (Yin et al., 2022). These approaches demonstrate the potential of Koopman theory to connect nonlinear systems with established linear control methods.

The second path involves incorporating Koopman theory into model-free RL algorithms. One approach, Deep Koopman Reinforcement Learning (DKRL), uses local Koopman operators to improve data efficiency (Song et al., 2021). By learning separate operators for different regions of the state space, this method captures local dynamics more effectively, leading to improved policy learning. In a different approach, Koopman Forward (Conservative) Q-Learning (KFC), an offline algorithm, uses Koopman theory to infer symmetries in system dynamics (Weissenbacher et al., 2022). This method augments offline

datasets, improving policy performance and data efficiency in scenarios with limited data collection.

These advancements represent initial steps in exploring Koopman theory's potential impact on learning and control. Current research has focused on system identification for model-based control, planning, and data augmentation. The field presents opportunities for further exploration. Future studies could extend Koopman-inspired approaches to other aspects of learning, such as state representation, policy parameterization, or the design of new optimization objectives.

As research progresses, new methods that use Koopman theory's linearization properties are likely to emerge. These developments may lead to improvements in handling nonlinear dynamics, potentially enhancing the scalability and generalization capabilities of learning algorithms. The ongoing integration of Koopman theory with control and learning techniques could contribute to more effective performance in complex tasks and environments, advancing the development of adaptive artificial agents.

Chapter 3

Methodology

This chapter introduces Koopman-Inspired Proximal Policy Optimization (KIPPO), a novel Reinforcement Learning (RL) framework that integrates Koopman-inspired representation learning with the Proximal Policy Optimization (PPO) algorithm. The framework aims to learn a feature space that captures the essence of environment dynamics while simplifying their representation, facilitating the development of stable and high-performing control policies.

The chapter is organized as follows:

1. Section 3.1 details the representation learning component of KIPPO, including the architecture design, future state prediction process, and the objectives and constraints guiding the learning process.

2. Section 3.2 explains the integration of representation learning with the PPO algorithm and outlines the training process, including rollout and optimization phases.

3. Section 3.3 summarizes the key contributions and novelty of the methodology, discussing potential implications for policy optimization in complex environments.

3.1 Koopman-Inspired Representation Learning

The representation learning component is fundamental to KIPPO, designed to capture essential characteristics of environment dynamics while simplifying their representation in a learned feature space. This section provides a detailed exploration of the architecture design, future state prediction process, and the objectives and constraints guiding the learning process.

3.1.1 Architecture Design

KIPPO introduces a novel approach to policy optimization by learning a latent representation through carefully designed components. This representation learning approach is engineered for flexibility and adaptability, allowing seamless integration with policy gradient algorithms while preserving their core architecture and hyperparameters. The design is influenced by the data-driven methods for Koopman analysis discussed in Section 2.2.2, particularly the deep learning approaches that aim to learn appropriate observables directly from data (Lusch et al., 2018; Yeung et al., 2017).

As illustrated in Fig. 3.1, the architecture comprises the following key components:

1. **State Autoencoder:** Consists of an encoder $\boldsymbol{\varphi}_x$ and a decoder $\boldsymbol{\varphi}_x{}^{-1}$. The encoder maps the original state x_t to an encoded state y_t, while the decoder reconstructs the original state.

2. **Action Encoder:** The action encoder $\boldsymbol{\varphi}_u$ maps the action u_t to the latent representation v_t, enabling the agent to operate within this transformed space.

3. **Linear System Matrices:** The linear state-transition matrix $\mathbf{K}$ and control matrix $\mathbf{B}$ govern the dynamics within the latent space.

These components are closely related to the concepts introduced in Section 2.2. The state autoencoder and action encoder approximate the Koopman observable functions g and action observable functions, respectively, as discussed in Section 2.2.1 (Budišić et al., 2012;

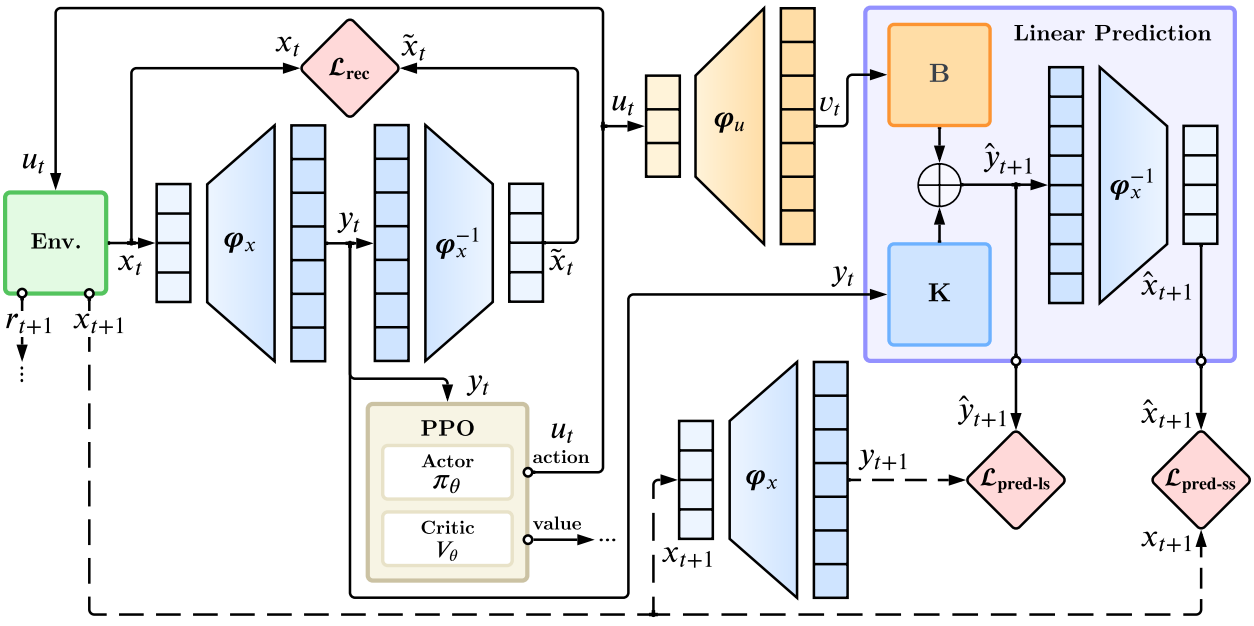

Figure 3.1: KIPPO framework architecture, highlighting key components and their interactions. The state autoencoder (encoder φ_x and decoder ${\varphi_x}^{-1}$) learns a compact encoded representation of environment states. The action encoder φ_u maps actions to this feature space. Within the encoded representation, dynamics are governed by the linear state-transition matrix $\mathbf{K}$ and control matrix $\mathbf{B}$. The policy optimization algorithm operates on the encoded states $y_t = \varphi_x(x_t)$.

Proctor et al., 2016a). This allows the agent to operate within a learned representation that accounts for both state transitions and the impact of actions, aligning with the control-oriented Koopman formulation (Proctor et al., 2016b). The linear system matrices $\mathbf{K}$ and $\mathbf{B}$ directly implement the finite-dimensional approximation of the Koopman operator, as described in equations Eqs. (2.7) and (2.8), governing the dynamics within the latent space.

The state autoencoder and action encoder are implemented as Multi-Layer Perceptrons (MLPs) with hyperbolic tangent (tanh) activation functions. This choice of architecture is inspired by the deep learning approaches for Koopman analysis discussed in Section 2.2.2 (Lusch et al., 2018; Yeung et al., 2017). Trainable parameters are initialized using Xavier uniform initialization (Glorot and Bengio, 2010). The number of layers and neurons per layer are consistent across the state encoder, state decoder, and action encoder, and are treated as hyperparameters, along with the dimensionality of the latent representation. Chapter 4 examines the impact of these hyperparameters in detail.

The use of nonlinear activation functions in the state autoencoder and action encoder might seem counterintuitive at first. While the Koopman operator governing the evolution of observables is inherently linear, the method of obtaining these observables does not need to be linear. These nonlinearities enable the MLPs to act as universal function approximators, making them well-suited for approximating the Koopman observables in the latent space. This approach aligns with the high-dimensional lifting methods discussed in Section 2.2.2 (Williams et al., 2015; Brunton et al., 2021). Nevertheless, the evolution of the system in the latent space is modeled through linear operations, formulated as an Linear Time-Invariant (LTI) system, consistent with the core principles of Koopman theory (Budišić et al., 2012; Lan and Mezić, 2013).

The state transition matrix $\mathbf{K}$ and control matrix $\mathbf{B}$ are implemented as learnable parameters, optimized alongside other components during training. This data-driven approach to learning the Koopman operator is consistent with modern methods described in Section 2.2.2 (Brunton et al., 2021). The size of these matrices corresponds to the dimensionality of the latent representation. The $\mathbf{K}$ matrix uses orthogonal initialization

(Saxe et al., 2014), while the control matrix is initialized with zeros, starting with a neutral control effect and allowing the agent to gradually learn the impact of its actions.

3.1.2 Future State Prediction

The prediction process leverages the learned dynamics to forecast states over a defined horizon H. This process begins with an initial state x_0 and an action sequence $u_{0:H}$. The initial state is first encoded into a latent representation using the state encoder $\boldsymbol{\varphi}_x$:

$$\hat{y}_0 = \boldsymbol{\varphi}_x(x_0) \tag{3.1}$$

The process then iteratively applies the state-transition matrix $\mathbf{K}$ and control matrix $\mathbf{B}$ to predict future states. At each timestep h, the next encoded state is predicted using the update rule:

$$\hat{y}_{h+1} = \mathbf{K}\hat{y}_h + \mathbf{B}\boldsymbol{\varphi}_u(u_h), \tag{3.2}$$

which generates a sequence of predicted latent states $\hat{y}_{1:H}$. In this equation, $\boldsymbol{\varphi}_x$ and $\boldsymbol{\varphi}_u$ represent g and f from the Koopman-based control formulation previously introduced in Eq. (2.8).

The predicted encoded states can be transformed back to the original state space using the state decoder ${\boldsymbol{\varphi}_x}^{-1}$, resulting in the predicted states $\hat{x}_{1:H}$. However, instead of directly using these predicted states for planning or generating additional training data, the future state prediction process serves to constrain the learning of the latent representation.

The prediction horizon H is a hyperparameter that determines the number of future time steps for which the environment's state is predicted. While longer horizons can capture more long-term dependencies, they may lead to increased computational demands and potential accumulation of prediction errors. The optimal value for the prediction horizon often depends on specific characteristics of the environment, such as the time scale of relevant dynamics and the sparsity of rewards.

For a more detailed breakdown of the future state prediction process, including the complete pseudocode, refer to Appendix A.2.

3.1.3 Objectives and Constraints

KIPPO draws inspiration from Koopman operator theory, which provides a framework for representing nonlinear dynamics as linear operations in a higher-dimensional space of observables. The goal is to learn a latent representation with properties beneficial for reinforcement learning in complex, nonlinear environments.

The desired properties of this latent space are:

1. **Informativeness:** Retain essential information from the original state space.
2. **Simplification:** Represent the environment dynamics in a form that can be more effectively approximated by linear operations.
3. **Predictability:** Exhibit dynamics that allow for accurate multi-step predictions.
4. **Consistency:** Maintain alignment with the true environment dynamics.

Informativeness and consistency ensure that the agent can make decisions based on accurate representations of the environment. Simplification and predictability aim to facilitate more manageable learning and planning, potentially leading to improved performance and stability in the learning process.

To achieve these properties, KIPPO employs three main loss components:

1. **Reconstruction Loss:** Primarily addresses informativeness.
2. **Latent-Space Prediction Loss:** Targets simplification and predictability.
3. **State-Space Prediction Loss:** Focuses on consistency and predictability.

These loss components work together to shape the latent space, balancing the sometimes competing goals of simplification and information preservation. The following subsections explore each loss component in detail, discussing their mathematical formulations and how they contribute to the desired latent space properties.

Reconstruction Loss

The reconstruction loss addresses the critical property of informativeness in the latent representation. This loss is also relevant to the Koopman-inspired approach, as it aligns with the assumption in Koopman theory that observable functions are typically invertible.

The reconstruction loss $\mathcal{L}_{\text{rec}}$ is defined as:

$$\mathcal{L}_{\text{rec}} = \left(\varphi_x{}^{-1}\left(\varphi_x(x_t)\right) - x_t\right)^2 \tag{3.3}$$

where φ_x and $\varphi_x{}^{-1}$ represent the state encoder and decoder, respectively. In the context of Koopman theory, the encoder φ_x can be viewed as an approximation of the observable functions, while the decoder $\varphi_x{}^{-1}$ approximates their inverse. This loss ensures that:

- The latent space retains sufficient information to reconstruct the original states accurately, mirroring the invertibility property of Koopman observables.
- Essential features of the environment are preserved in the latent representation.
- The representation does not become overly simplified or lossy, maintaining crucial details from the original state space.

By minimizing reconstruction error, this loss prevents the latent space from discarding important information or collapsing into an overly simplistic representation. It acts as a regularizer, ensuring that while the latent space may simplify the dynamics, it does not do so at the cost of losing essential information. This directly supports the goal of informativeness, striking a balance between simplification and detail preservation.

The reconstruction loss thus plays a crucial role in the KIPPO framework by driving the model to learn a bijective mapping between the original state space and the latent space, which is consistent with the principles of Koopman theory. This helps ensure that the learned representation maintains a meaningful and invertible connection to the original state space.

Latent-Space Prediction Loss

The latent-space prediction loss is central to KIPPO's goal of creating a simplified and predictable representation of the environment dynamics. This loss encourages the latent space to exhibit dynamics that can be more effectively approximated by linear operations, aligning with the Koopman-inspired approach.

The latent-space prediction loss $\boldsymbol{\mathcal{L}}_{\text{pred-ls}}$ is formulated as:

$$\boldsymbol{\mathcal{L}}_{\text{pred-ls}} = \frac{1}{H} \sum_{h=1}^{H} m_h \left(\hat{y}_h - \boldsymbol{\varphi}_x(x_h) \right)^2 \tag{3.4}$$

Here, H is the prediction horizon, determining the number of future time steps for state prediction. At each time step h, $\hat{y}_h$ is the predicted latent state, obtained through iterative application of the learned linear approximation using $\mathbf{K}$ and $\mathbf{B}$ as described in the future state prediction process (see Section 3.1.2). $\boldsymbol{\varphi}_x(x_h)$ is the encoded target state, derived by applying the encoder to the true environment state.

The binary mask m_h handles variable-length trajectories, ensuring the loss is only computed for relevant time steps within each episode. It is defined as:

$$m_h = \begin{cases} 1 & \text{if } h \text{ is within the current episode} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

This loss contributes to the desired latent space properties by:

- Encouraging dynamics that can be more effectively approximated by linear operations, as $\hat{y}_h$ is generated using linear matrices $\mathbf{K}$ and $\mathbf{B}$.
- Promoting predictability by minimizing multi-step prediction errors in the latent space.
- Indirectly supporting simplification by incentivizing the encoder to find a representation where linear predictions are accurate.

By minimizing this loss, the framework aims to create a latent space where complex nonlinear dynamics are represented in a form more amenable to linear approximations, potentially simplifying the learning task for the reinforcement learning algorithm.

State-Space Prediction Loss

The state-space prediction loss plays a crucial role in maintaining consistency between the learned latent representation and the true environment dynamics. This loss ensures that simplifications made in the latent space do not come at the cost of fidelity to the original system.

The state-space prediction loss $\boldsymbol{\mathcal{L}}_{\text{pred-ss}}$ is defined as:

$$\boldsymbol{\mathcal{L}}_{\text{pred-ss}} = \frac{1}{H} \sum_{h=1}^{H} m_h \left({\boldsymbol{\varphi}_x}^{-1}(\hat{y}_h) - x_h \right)^2, \tag{3.6}$$

where ${\boldsymbol{\varphi}_x}^{-1}(\hat{y}_h)$ is the reconstructed state, obtained by decoding the predicted latent state $\hat{y}_h$ at time step h, and x_h is the true environment state at time step h.

Importantly, $\hat{y}_h$ is generated through the same process as in the latent-space prediction loss, using the learned matrices $\mathbf{K}$ and $\mathbf{B}$, ensuring consistency between the two losses. The binary mask m_h, as in the latent-space prediction loss, handles variable-length trajectories.

This loss contributes to the desired latent space properties by:

- Ensuring consistency between the latent dynamics and the true environment dynamics.
- Promoting predictability by minimizing multi-step prediction errors when mapped back to the original state space.
- Indirectly supporting informativeness by ensuring the latent space captures information relevant for accurate long-term predictions.

By minimizing this loss, the framework aims to maintain a strong connection between the simplified latent representation and the true environment dynamics, ensuring that the learned representation remains grounded in reality.

Total Representation Loss

The total representation loss combines the individual loss components to create a balanced objective that addresses all desired properties of the latent space. This unified loss function allows KIPPO to learn a representation that is informative, simplified, predictable, and consistent with the true environment dynamics.

The total representation loss is formulated as:

$$\mathcal{L}_{\text{KI}} = \omega_1 \mathcal{L}_{\text{rec}} + \omega_2 \mathcal{L}_{\text{pred-ls}} + \omega_3 \mathcal{L}_{\text{pred-ss}} \tag{3.7}$$

This combined loss aims to create a latent space that balances all four primary goals:

- **Informativeness:** Primarily through $\mathcal{L}_{\text{rec}}$, supported by $\mathcal{L}_{\text{pred-ss}}$
- **Simplification:** Primarily through $\mathcal{L}_{\text{pred-ls}}$
- **Predictability:** Through both $\mathcal{L}_{\text{pred-ls}}$ and $\mathcal{L}_{\text{pred-ss}}$
- **Consistency:** Primarily through $\mathcal{L}_{\text{pred-ss}}$

The weights ω_1, ω_2, and ω_3 allow for fine-tuning the balance between these goals, adapting to the specific requirements of different environments or tasks. When choosing these weights, several factors need to be considered, including the relative scales of the latent space and state space vectors, which typically differ. Domain knowledge about the specific task or environment can also inform the choice of weights, as can empirical evaluation or hyperparameter tuning results.

The combination of loss components in KIPPO works together to potentially achieve a more effective representation of dynamics in several interconnected ways. It encourages a latent representation where dynamics can be more effectively approximated by linear operations, promoting the finding of a latent space that facilitates such approximations. The framework allows for complex transformations between original and latent spaces through nonlinear feature extraction. It also captures longer-term dependencies in system dynamics by enforcing multi-step consistency. Furthermore, KIPPO retains essential information from

the original state space, ensuring information preservation. Finally, it maintains consistency between learned and true environment dynamics, grounding the representation in reality.

This approach to dynamics representation is inspired by Koopman theory's concept of lifting nonlinear dynamics to a space where they can be represented by linear operators. While KIPPO does not seek an exact linear representation, it aims to capture some of the benefits of this approach in a practical, learnable framework.

These mechanisms work together to create a representation where the dynamics can be more effectively approximated by linear operations, while still capturing the essential behavior of the original system. The resulting representation aims to provide a potentially more manageable form of the system dynamics, which may help address some challenges associated with learning to navigate complex, nonlinear systems. The effectiveness of this approach and its impact on performance will be thoroughly examined in the experimental section.

KIPPO is designed with the goal of improving both performance and consistency across multiple trials in complex, nonlinear environments. Chapter 4 will investigate whether the learned representation achieves these aims and potentially leads to more efficient and robust policy learning.

3.2 Integration with Policy Gradients

This work integrates Koopman-inspired representation learning with the PPO algorithm, forming KIPPO. This integration leverages PPO's strengths, including stability, sample efficiency, and compatibility with continuous action spaces. The implementation builds upon the CleanRL library (Huang et al., 2022), adopting its core hyperparameters while introducing additional parameters specific to the auxiliary components, prediction horizon, and loss function weights. This approach allows for an analysis of the effects of representation learning on performance and stability.

3.2.1 Decoupled Representation and Policy Learning

A key feature of KIPPO is the decoupling of representation learning from policy optimization. This separation helps ensure that performance improvements can be attributed to the properties of the learned representation, rather than to the addition of extra network layers.

During the optimization phase, the state encoder operates in inference mode, providing a consistent latent representation of states. This allows the algorithm to operate on simplified dynamics without being affected by encoder updates during representation loss optimization. The representation learning components (state encoder, state decoder, action encoder, state-transition matrix, and control matrix) are optimized independently of the policy and value function networks. Gradients from the PPO loss do not influence representation learning; instead, the representation is optimized based on the objectives and constraints outlined in Section 3.1.3.

This approach differs from simply adding layers to the networks. In the latter case, additional layers would be optimized based on policy loss, primarily aiming to improve policy performance. In contrast, KIPPO's representation learning components are optimized independently to capture essential environment dynamics in a reduced-complexity form.

This independent optimization is important for isolating the effects of the learned representation and ensuring generalization beyond the current policy. By preventing direct influence from the policy optimization process, observed improvements can be attributed to the properties of the learned representation itself. Moreover, this approach encourages the representation to learn more general environmental features, potentially benefiting a broader range of policies.

3.2.2 Overview of Training Process

The training process alternates between two main phases: rollout and optimization, continuing until a predetermined number of environment steps is completed.

During the rollout phase, the agent interacts with the environment, collecting standard data (states, actions, rewards) and additional sequences necessary for computing representation losses. Data is stored in separate buffers for efficient processing. Each rollout phase collects 2,048 environment steps, potentially spanning multiple trajectories. If a terminal state is reached before collecting all steps, the environment is reset with new initial conditions. This approach helps ensure diverse experiences across trajectories. For a detailed description of the rollout phase, including pseudocode, refer to Appendix A.1.

In the optimization phase, collected data is used to compute losses and update framework components. The 2,048 steps are divided into 32 mini-batches. For each mini-batch, reconstruction loss, prediction loss in latent representation, and prediction loss in state-space are computed to update the parameters of representation learning components. Separately, the PPO loss (including policy and value function losses) is computed using encoded states $\boldsymbol{\varphi}_x(x)$ to update actor and critic network parameters.

The total loss for the framework combines the weighted representation loss $\boldsymbol{\mathcal{L}}_{\text{KI}}$ and the standard PPO loss $\boldsymbol{\mathcal{L}}_{\text{PPO}}$:

$$\boldsymbol{\mathcal{L}}_{\text{KIPPO}} = \boldsymbol{\mathcal{L}}_{\text{KI}} + \boldsymbol{\mathcal{L}}_{\text{PPO}} \tag{3.8}$$

Both losses update their respective components' parameters using the Adam optimizer, with gradients from each loss only updating corresponding components. This separation ensures that representation learning is guided by its own objectives and constraints, independent of policy performance.

The optimization process repeats for 10 epochs, allowing iterative refinement of both latent representation and policy. After optimization, a new rollout phase begins, collecting fresh environment steps. This cycle continues until reaching the total of 1 million environment steps.

For a comprehensive breakdown of the optimization phase, including pseudocode, refer to Algorithm 3.

3.3 Summary

This chapter has introduced KIPPO, a framework that integrates Koopman-inspired representation learning with the PPO algorithm. The core of this approach is a multi-objective optimization that aims to learn a latent representation with the following key properties:

1. **Informativeness:** Preserving essential information from the original state space.
2. **Simplification:** Representing environment dynamics in a form that can be more effectively approximated.
3. **Predictability:** Exhibiting dynamics that allow for accurate multi-step predictions.
4. **Consistency:** Maintaining alignment with the true environment dynamics.

These properties are pursued through a combination of reconstruction loss, latent-space prediction loss, and state-space prediction loss. The framework incorporates:

- A state autoencoder and action encoder to learn the latent representation.
- State-transition and control matrices to model system evolution in the latent space.
- Integration with PPO, allowing policy optimization on encoded states.
- Independent optimization of the latent representation from the policy.

By balancing complexity reduction and information preservation, KIPPO aims to facilitate policy learning in complex, nonlinear environments. The framework's effectiveness will be examined in Chapter 4, comparing its performance and consistency to a baseline PPO implementation across various continuous control tasks.

Chapter 4

Experiments and Results

This chapter presents a comprehensive evaluation of the Koopman-Inspired Proximal Policy Optimization (KIPPO) algorithm, comparing its performance to the baseline Proximal Policy Optimization (PPO) algorithm across a diverse set of continuous control tasks. The experiments are designed to assess the effectiveness of integrating Koopman-inspired representation learning into the policy optimization process, focusing on both performance improvements and variability reduction.

The evaluation uses six environments from the Gymnasium library, powered by the MuJoCo and Box2D physics simulation engines. These environments, ranging from simple to complex control tasks, serve as a testbed for examining the central hypothesis of this thesis: that leveraging Koopman theory to simplify the dynamics of complex environments can lead to improved policy optimization in terms of effectiveness and robustness.

This chapter is structured as follows:

1. Section 4.1 details the experimental setup, including the specific environments used, performance metrics, and information about the baseline algorithm and model configurations.

2. Section 4.2 presents the main results of the experimental evaluation, comparing the performance and variability to the baseline across multiple runs. This section also

includes an ablation study of the loss components and a comparative per-environment analysis into the effects of the latent dimension and prediction horizon.

3. Section 4.3 provides an in-depth analysis of hyperparameters, examining their individual effects, relative importance, and interactions. This analysis offers insights into the algorithm's behavior and guides future applications and improvements.

4. Section 4.4 synthesizes the key findings from the experiments and hyperparameter analysis, discussing their implications and potential future research directions.

4.1 Experimental Setup

This section details the experimental setup used to evaluate KIPPO, ensuring a thorough and fair assessment of its performance across various conditions. It covers the selection of environments, model training configurations, and the evaluation metrics used to quantify performance.

4.1.1 Environments

To evaluate KIPPO's performance across a range of challenges, six continuous control environments from the Gymnasium library (Towers et al., 2023) are selected. These environments, powered by the MuJoCo (Todorov et al., 2012) and Box2D (Catto, 2007) physics simulation engines, present increasing levels of complexity:

1. **Inverted Pendulum:** Balances a pole on a cart, offering a simple testbed with relatively straightforward dynamics.

2. **Lunar Lander:** Requires precise thruster control for landing, balancing fuel consumption, landing speed, and landing pad proximity.

3. **Bipedal Walker:** Features uneven terrain and lidar input, testing adaptation to varying environmental conditions.

4. **Hopper:** Involves single-leg hopping dynamics, necessitating coordinated control for stable movement and forward progression.

5. **Walker:** Presents a bipedal locomotion task requiring balance and coordination of multiple joints for forward movement.

6. **Half Cheetah:** Comprises a high-speed locomotion task with a planar body and two legs, focusing on dynamic gaits and efficient joint coordination.

Figure 4.1 provides visualizations of these environments, and Table 4.1 summarizes their complexity and dimensionality of state and action spaces.

The selected environments offer several key features that make them suitable for assessing the Koopman-inspired approach. Their nonlinear dynamics challenge the algorithm to learn effective representations in the Koopman observable space, testing KIPPO's ability to simplify complex, nonlinear systems. The varying initial conditions evaluate the algorithm's ability to learn policies that generalize across different starting states, addressing the known issue of sensitivity to initialization which can cause poor consistency in results across multiple trials and high variance in gradient estimates. These environments also assess the algorithm's capacity to capture extended temporal dynamics, particularly in locomotion tasks, through their long-term dependencies.

The continuous state and action spaces of these environments test performance in high-dimensional control tasks, challenging KIPPO to effectively learn and operate in complex, continuous domains, mirroring the characteristics of many real-world control problems. The varying levels of stochasticity, ranging from low in InvertedPendulum to high in BipedalWalker, assess the algorithm's robustness to different degrees of uncertainty. These features collectively provide a comprehensive testbed for evaluating the effectiveness and versatility of the Koopman-inspired approach in reinforcement learning across diverse control scenarios.

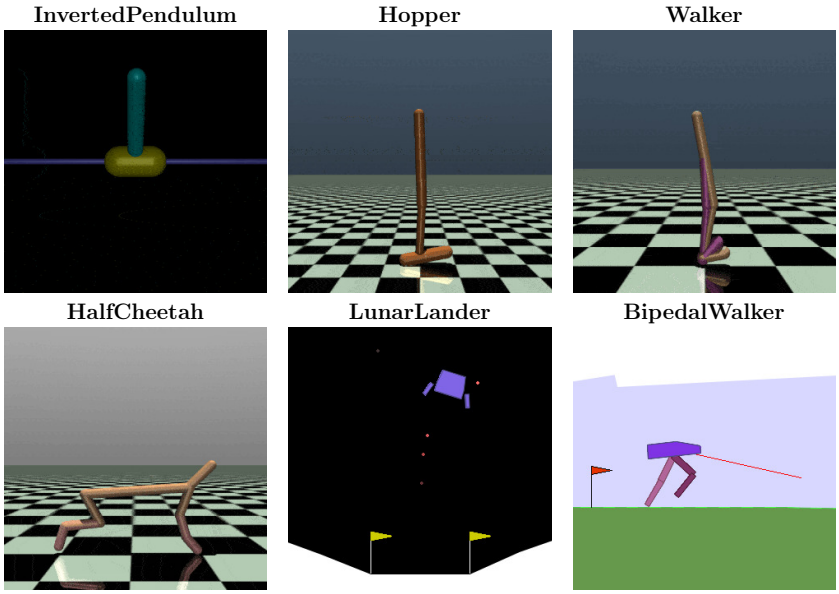

Figure 4.1: Visualizations of the six continuous control environments used for evaluation.

Table 4.1: Overview of the selected environments, highlighting their complexity and dimensionality of their state and action spaces.

Environment	Complexity	State Dim.	Action Dim.
InvertedPendulum-v4	Low	4	1
LunarLanderCont.-v2	Medium	8	2
BipedalWalker-v3	Medium	24	4
Hopper-v4	High	11	3
Walker2d-v4	High	17	6
HalfCheetah-v4	Very High	17	6

4.1.2 Model Training and Configurations

The baseline for comparison is the PPO implementation from the CleanRL library (Huang et al., 2022). For each environment, four models with different initializations are trained for both the baseline and KIPPO.

The baseline hyperparameters, presented in Table 4.2, are based on CleanRL's benchmarks. These are known to produce strong results in these environments, ensuring a well-tuned, competitive reference point. KIPPO uses the same hyperparameters for the shared aspects, with additional parameters specific to the Koopman-inspired representation learning. This strategy ensures a fair comparison while allowing the new method to leverage its distinct characteristics.

Initialization seeds are kept consistent across environments and architectures, ensuring that performance differences stem from the approaches themselves rather than initialization variations. For instance, if the KIPPO models uses seeds 1, 2, 3, and 4, the corresponding baseline models also uses seeds 1, 2, 3, and 4 for their respective runs in the same environment. This setup isolates the effects of initial conditions in the environments and network parameter initialization, allowing for a fair comparison under identical starting conditions.

Each model is trained for 1 million environment steps, consistent with the benchmarks for the same environments, providing a standardized measure for direct comparison.*

4.1.3 Evaluation Metrics

An Reinforcement Learning (RL) agent's performance is typically measured by the episodic return, which represents the cumulative reward obtained during an episode, as first introduced in Eq. (2.11) from Section 2.3.1. The episodic return provides a direct measure of how well the agent is performing in the environment, with higher returns indicating better performance. However, episodic returns can be noisy and fluctuate significantly, especially

*Training 6 models in parallel takes roughly 4 hours to complete on an Nvidia H100 GPU and 16 allocated CPU cores. The environment uses Python 3.10, PyTorch 2.1.2 with CUDA 12.1, and Gymnasium 0.29.1.

during the early stages of training when the agent is still exploring the environment and learning the task.

To address this issue, the Exponentially Weighted Moving Average (EWMA) of episodic returns is employed to assess learning progress and track agent performance over time. It is calculated recursively as follows:

$$\text{EWMA}_t = \alpha \cdot \text{EWMA}_{t-1} + (1 - \alpha) \cdot G_t \tag{4.1}$$

where α is the smoothing factor, which determines the weight given to past observations, and G_t is the episodic return observed at time step t. A higher α gives more weight to past observations, resulting in a smoother curve but slower response to recent changes.[†]

The EWMA offers several advantages over raw values or simple sliding-window averages. It is responsive to recent changes while still considering historical data, allowing it to quickly reflect performance shifts without completely disregarding past trends. This balance between recent and historical data enhances its predictive capability, providing insights into underlying learning trends. The EWMA also excels at noise reduction by filtering out short-term fluctuations, which can obscure the underlying learning progress. Furthermore, it provides robustness against outliers, providing stable measurements even when occasional extreme values occur. These characteristics collectively result in a clearer and more interpretable representation of the agent's learning trend.

The Cumulative Temporal Error (CTE) measures the prediction accuracy of the learned Koopman-inspired model, providing insights into the quality of the learned representation and its ability to capture the underlying dynamics of the environment. Unlike the EWMA, which focuses on the agent's performance, the CTE specifically evaluates the accuracy of the learned dynamics model. It is calculated as follows:

$$\text{CTE} = \frac{1}{H} \sum_{h=1}^{H} \frac{1}{h} \sum_{k=1}^{h} |\hat{x}_k - x_k| \tag{4.2}$$

[†]An α value of 0.05 is used for the EWMA calculation, balancing smoothing of short-term fluctuations with responsiveness to recent performance changes.

where h is the prediction horizon, representing the number of steps into the future for which the model makes predictions, k is the time step within the horizon, ranging from 1 to h, $\hat{x}_k$ is the predicted state at time step k within the horizon, based on the model's learned dynamics, and x_k is the actual (observed) state at time step k within the horizon.

The CTE measures the average prediction error of the learned dynamics model across different horizons. For each horizon, the model predicts the state for that many steps ahead, and the difference between the predicted and actual states is calculated at each step. These differences are then averaged, giving more weight to shorter-term predictions, and finally averaged over all horizons to obtain the CTE.

For each set of four models with different initializations, the mean and standard deviation of the EWMA of final episodic returns and CTE values are reported.

- **Mean:** Represents the average performance of the models, despite variations in initialization, providing insights into the overall effectiveness.
- **Standard deviation:** Quantifies the variability in performance across different random seeds, indicating consistency and robustness.

4.2 Comparison with Baseline

This section compares KIPPO with the PPO baseline across the six diverse continuous control environments presented in Section 4.1. The primary objective is to assess the performance and variability improvements offered by integrating Koopman-inspired representation learning into the policy optimization process. These results provide insights into whether the hypothesized benefits of the Koopman-inspired approach, as discussed in the Chapter 3, translate into tangible performance gains.

Table 4.3 presents the mean and standard deviation of the EWMA episodic returns reported at the end of training. It also shows the percent difference relative to the baseline for each environment across the four models with different initializations. KIPPO achieves higher mean performance in all environments, with improvements ranging from 6.36% to

60.26%. Moreover, it exhibits lower standard deviation in five out of the six environments, indicating enhanced consistency across different random seeds, with reductions ranging from 26.89% to 91.43%.

Figure 4.2 compares the training curves for KIPPO and the PPO baseline across the six environments, revealing several notable trends:

- **InvertedPendulum**: The proposed method surpasses the baseline around 0.2 million steps, maintaining superior performance and lower standard deviation thereafter.
- **LunarLander**: While the baseline converges to a return of 200 midway, KIPPO continues improving throughout, with similar variability.
- **BipedalWalker**: The Koopman-inspired approach maintains lower variability throughout and surpasses the baseline towards the end, while PPO shows increased standard deviation between 0.2-0.5 million steps.
- **Hopper**: KIPPO shows consistent improvement in mean returns, with variability similar to the baseline.
- **Walker2d**: The proposed method maintains lower variability and surpasses the baseline towards the end, suggesting potential for further improvement with extended training.
- **HalfCheetah**: KIPPO demonstrates significantly improved mean returns but slightly higher variability compared to the baseline.

The enhanced performance and reduced variability across different initializations underscore the potential of this approach for improving the robustness and reliability of policy optimization. These characteristics could prove advantageous in real-world applications where consistent performance is desirable. The comparison highlights the ability to optimize policies effectively in terms of both aspects across various environments with nonlinear dynamics.

Table 4.2: Shared hyperparameters for the baseline PPO models and KIPPO.

Hyperparameter	**Value**
Global Steps	1,000,000
Steps in Rollouts Phase T	2,048
Num. Mini-batches	32
Num. Update Epochs N_{epochs}	10
Discount Factor γ	0.99
GAE λ	0.95
Normalized Advantages	Yes
Clipping Coef. ϵ	0.2
Clip Value Loss	Yes
Weight of Value Loss	0.5
Weight of Entropy Loss	0.0
Optimizer	Adam
Learning Rate α	3×10^{-4}
LR Annealing	Yes
Max. Gradient Norm	0.5

Table 4.3: Per-environment overview of the main results comparing KIPPO and the PPO baseline in terms of mean and std. of final episodic returns (EWMA) across four trials, and the percent difference in these metrics relative to the baseline.

Environment	**Baseline**		**KIPPO**		**% Diff.**	
	Mean	Std.	Mean	Std.	Mean	Std.
InvertedPendulum-v4	897.57	41.61	**998.18**	**3.57**	11.21	-91.43
Hopper-v4	2315.43	226.87	**2520.53**	**100.36**	8.86	-55.76
Walker2d-v4	3126.73	450.30	**3325.74**	**287.85**	6.36	-36.08
HalfCheetah-v4	1927.59	**1030.66**	**3089.20**	1203.42	60.26	16.76
LunarLanderCont.-v2	208.38	11.59	**280.81**	**8.27**	34.76	-28.66
BipedalWalker-v3	235.09	19.03	**255.91**	**13.91**	8.86	-26.89

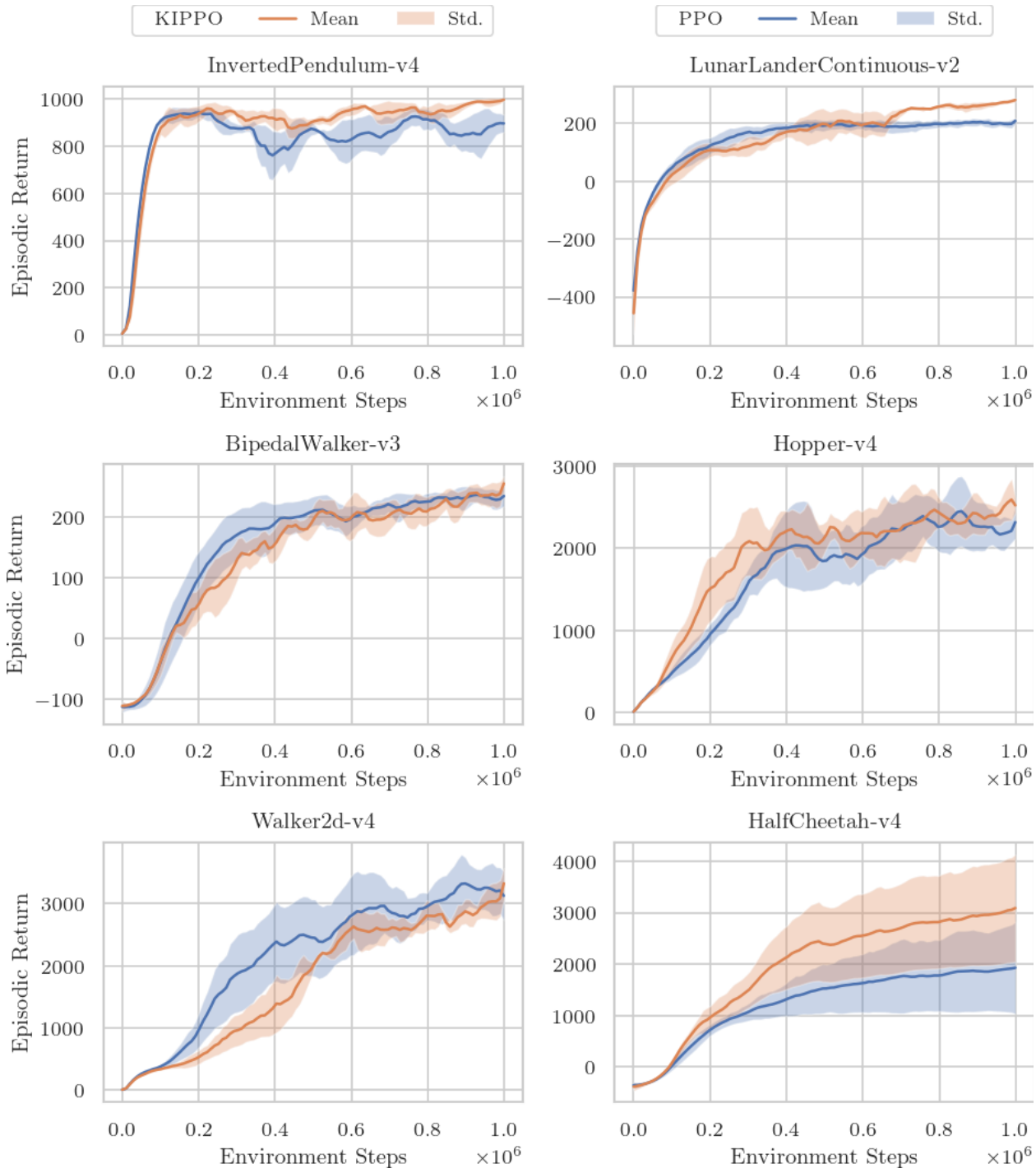

Figure 4.2: Training curves comparing KIPPO (orange) to the PPO baseline (blue) across four trials per environment. The solid lines represent the mean performance, while the shaded regions indicate the standard deviation.

Table 4.4 summarizes the hyperparameters used for the KIPPO models trained for the main results showcased in this section. The latent dimension and prediction horizon are varied per-environment, and the best-performing values are reported. Section 4.2.1 presents more details, comparisons, and analysis as compared to the baseline results. The remaining hyperparameters are fixed to the values shown, as determined empirically throughout the development process.

Section 4.3 provides a more in-depth analysis of all hyperparameters, including their individual and interactive effects, and will provide further insights into the observed performance differences.

4.2.1 Varying the Latent Dimension and Prediction Horizon

This section examines the effects of latent dimension and prediction horizon across six environments, comparing KIPPO against the baseline PPO. Latent dimensions from 16 to 48 and prediction horizons from 1 to 10 steps are analyzed, keeping other hyperparameters fixed as shown in Table 4.4.

Tables 4.5 and 4.6 present the mean and standard deviation of EWMA and CTE values for different configurations, with baseline results denoted by dashes (–) and best performances bolded.

Latent dimension generally shows a positive correlation with performance. As the dimension increases, mean episodic returns tend to improve, peaking at 48 in most environments. This suggests that more expressive representations can better capture relevant information about environment dynamics. Concurrently, the standard deviation of returns often decreases with increasing latent dimension, indicating more consistent performance across initializations.

The benefits of increased dimensionality vary across environments. InvertedPendulum shows improved mean returns up to a dimension of 32, while HalfCheetah achieves better standard deviation at this dimension. More complex environments like BipedalWalker benefit from larger latent spaces, surpassing baseline performance in mean returns for dimensions

32 and 48. LunarLanderContinuous demonstrates consistent improvement across all tested dimensions, outperforming the baseline in mean episodic returns.

Larger latent dimensions often lead to improved CTE values, suggesting better predictive accuracy of the learned models. This improvement in prediction quality likely contributes to the enhanced policy performance observed with larger latent spaces. However, the extent of this improvement varies by task, highlighting the importance of environment-specific tuning.

The effects of prediction horizon are more varied, emphasizing its task-dependent nature. Generally, longer horizons increase mean CTE values, reflecting the increasing difficulty of prediction over extended time steps. However, the impact on performance metrics differs across environments.

InvertedPendulum shows improved mean episodic returns for all horizons compared to the baseline. HalfCheetah's mean returns improve with horizon length up to 10, but horizons 1 and 3 offer better standard deviation, suggesting a trade-off between performance and consistency. LunarLanderContinuous benefits from all tested horizons in mean returns, with horizons 3 and 5 also improving standard deviation.

Some environments show clear limitations in useful prediction length. Hopper's performance degrades for horizons above 5, while BipedalWalker shows improvements for most horizons except 10. These observations underscore the importance of matching the prediction horizon to the relevant time scales of each task's dynamics.

In summary, these results reveal complex relationships between latent dimension, prediction horizon, and performance metrics across different environments. The general trend of improved performance with increased latent dimension suggests benefits in using more expressive representations. However, the variability in optimal prediction horizons underscores the need for task-specific tuning. These findings demonstrate KIPPO's ability to improve upon the baseline across various environmental complexities and prediction lengths, highlighting the potential of Koopman-inspired approaches in reinforcement learning.

Section 4.3 provides a detailed exploration of hyperparameter interactions and their combined effects on performance.

4.2.2 Ablation Study of Loss Components

An ablation study evaluates the impact of individual loss components: reconstruction loss ($\mathcal{L}_{\text{rec}}$), latent-space prediction loss ($\mathcal{L}_{\text{pred-ls}}$), and state-space prediction loss ($\mathcal{L}_{\text{pred-ss}}$). Table 4.7 presents these results, with checkmarks (✓) indicating included losses and dashes (–) indicating excluded ones. The first row per environment represents baseline PPO performance, with bolded values indicating best performance per metric.

Key observations include:

- Having only reconstruction loss shows results comparable to baseline.
- Integrating all losses yields best mean EWMA and lowest std. in 5 out of 6 cases.
- The mean CTE decreases as more loss components are incorporated.
- Latent-space prediction loss often significantly reduces CTE mean and std.

The combination of all losses generally improves performance and consistency across initializations. For instance, in InvertedPendulum, using all losses outperforms the baseline (998.18 vs 897.57) with lower standard deviation (3.57). However, in HalfCheetah, using only two losses leads to lowest standard deviation.

Decreasing CTE with additional losses indicates each loss's contribution to representation and dynamics accuracy. In Walker2d, mean CTE drops from 1.325 (reconstruction loss only) to 0.054 (all losses), demonstrating improved predictive model accuracy.

The latent-space prediction loss's impact on CTE suggests that encouraging certain latent space dynamics properties improves prediction accuracy.

Section 4.3 provides a comprehensive analysis of loss weight variations and their interactions with other hyperparameters.

4.3 Hyperparameter Analysis

This section analyzes the hyperparameters used in KIPPO to understand their impact on performance, sensitivity, and interactions. These insights are crucial for assessing the algorithm's robustness and generalization capabilities in different conditions.

The analysis is based on 300 sets of models, each comprising four models (with different initializations) for each of the six environments, totaling 7,200 models. The hyperparameters explored, along with their values and ranges, are shown in Table 4.8.

To enable meaningful comparisons across environments with different reward scales, a standardized performance measure is used. This measure is based on the mean and standard deviation of the final episodic returns from the baseline models.

In the following sections, "Mean" refers to the average across environments of the means across initializations for the standardized final returns. Similarly, "Std." represents the average across environments of the standard deviations across initializations for the standardized final returns. This approach emphasizes both average performance and consistency across different environments and initializations.

The analysis is structured into three main parts:

1. Section 4.3.1 examines the relative importance of individual hyperparameters using a random forest regressor, providing insights into which parameters have the most significant impact on performance and variability.

2. Section 4.3.2 investigates the effects of individual hyperparameters on KIPPO's performance using box plots, revealing trends and optimal ranges for each parameter.

3. Section 4.3.3 explores the interactions between pairs of hyperparameters using contour plots, uncovering complex relationships and interdependencies that affect the algorithm's behavior.

The plots in Sections 4.3.2 and 4.3.3 use a consistent color scale to aid interpretation of the results compared to the baseline, as detailed in Table 4.9.

This comprehensive analysis aims to provide a deep understanding of behavior across different hyperparameter configurations, guiding future applications and improvements of the algorithm.

4.3.1 Hyperparameter Importance

This section explores the importance of individual hyperparameters in determining the two key performance metrics: mean final returns and their standard deviation.

A random regressor is trained to predict the standardized performance metrics based on the hyperparameters used. This is a common approach to derive importance scores for hyperparameters in machine learning models. By analyzing how much each hyperparameter reduces variance across the forest's decision trees, one can identify the most influential ones. The importance scores, ranging from 0 to 1, denote how important each hyperparameter is in predicting the corresponding performance metric.

Figure 4.3 shows the importance scores for each hyperparameter, considering both mean final returns and their standard deviation.

For mean final returns, the latent-space prediction loss weight ω_2 is the most influential, with an importance score above 0.35. The latent dimension and the state-space prediction loss weight ω_3 are the next most important, with scores around 0.2.

In terms of standard deviation, all three loss weights show the highest importance scores around 0.2. The prediction horizon has score just below 0.15.

The low importance scores of the remaining hyperparameters indicate that KIPPO is relatively robust to variations in these hyperparameters within the tested ranges.

4.3.2 Effects of Individual Hyperparameters

This section examines the impact of individual hyperparameters on KIPPO's performance.

Box plots visualize the distribution of the standardized performance metrics across different hyperparameter values and ranges. Each plot integrates results from all 7,200

Table 4.4: Hyperparameter options used in the main experiments and results comparing KIPPO and the PPO baseline. The latent dimension and prediction horizon are varied per-environment for analysis, while the others are fixed.

Hyperparameter	**Value**
Latent Dim.	Varied (16, 32, 48)
Pred. Horizon H	Varied (1, 3, 5, 10)
Num. Layers	2
Layer Neurons	128
Weight of $\mathcal{L}_{\text{rec}}$ (ω_1)	0.75
Weight of $\mathcal{L}_{\text{pred-ls}}$ (ω_2)	0.1
Weight of $\mathcal{L}_{\text{pred-ss}}$ (ω_3)	0.5

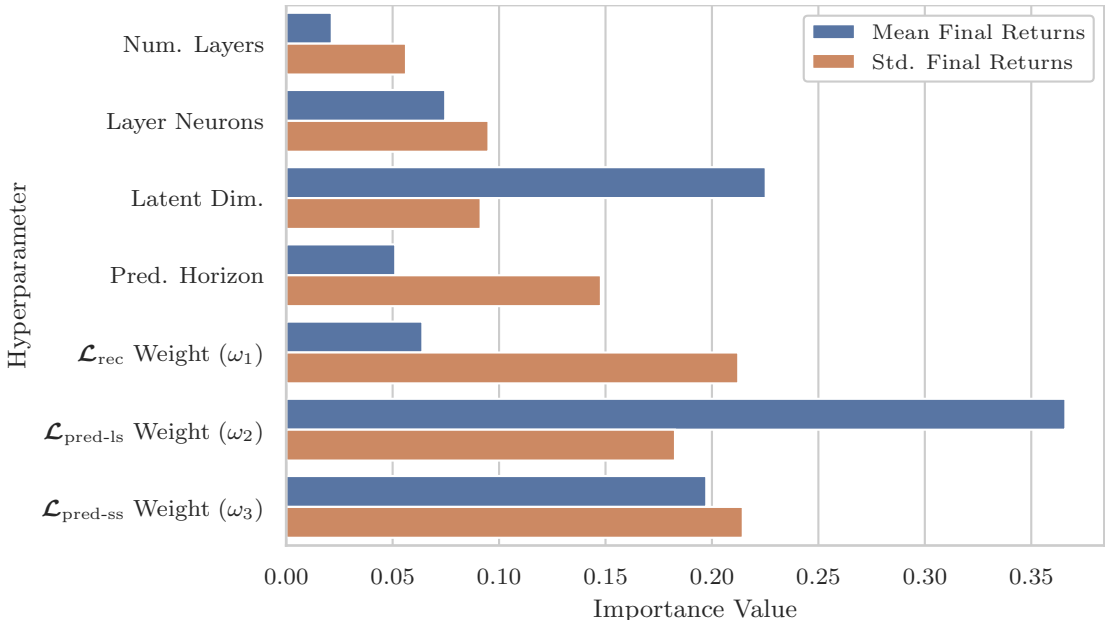

Figure 4.3: Hyperparameter importance scores derived from a random forest regressor.

Table 4.5: Per-environment comparison of latent dimension effects on final episodic returns (EWMA) values and prediction error (CTE). The baseline is shown as the first row for each environment.

Environment	**Latent Dim.**	**EWMA** Mean	Std.	**CTE** Mean	Std.
InvertedPendulum-v4	–	897.57	41.61	–	–
	16.0	969.94	48.74	**0.001**	**0.000**
	32.0	**998.18**	**3.15**	**0.001**	**0.000**
	48.0	956.21	53.01	**0.001**	**0.000**
Hopper-v4	–	2315.43	226.87	–	–
	16.0	1852.07	420.11	0.027	0.004
	32.0	2246.98	207.23	**0.017**	**0.002**
	48.0	**2520.53**	**88.78**	**0.017**	0.003
Walker2d-v4	–	3126.73	450.30	–	–
	16.0	1728.42	382.02	0.083	**0.003**
	32.0	2565.40	276.60	**0.053**	0.004
	48.0	**3325.74**	**254.65**	0.054	**0.003**
HalfCheetah-v4	–	1927.59	1030.66	–	–
	16.0	2601.19	968.05	0.147	0.028
	32.0	1434.47	**575.15**	0.117	0.012
	48.0	**3089.20**	1053.22	**0.100**	**0.009**
LunarLanderCont.-v2	–	208.38	11.59	–	–
	16.0	257.73	18.28	**0.032**	**0.004**
	32.0	253.35	12.09	0.038	0.008
	48.0	**280.81**	**7.32**	0.036	0.006
BipedalWalker-v3	–	235.09	19.03	–	–
	16.0	235.70	**6.48**	0.099	**0.003**
	32.0	254.85	13.27	0.082	0.005
	48.0	**255.91**	12.31	**0.075**	0.006

Table 4.6: Per-environment comparison of prediction horizon effects on final episodic returns (EWMA) values and prediction error (CTE). The baseline is shown as the first row for each environment.

Environment	Pred. Horizon	EWMA		CTE	
		Mean	Std.	Mean	Std.
InvertedPendulum-v4	–	897.57	41.61	–	–
	1.0	967.81	39.14	**0.001**	**0.000**
	3.0	**998.18**	**3.15**	**0.001**	**0.000**
	5.0	936.10	67.85	**0.001**	**0.000**
	10.0	966.83	39.91	0.003	**0.000**
Hopper-v4	–	2315.43	226.87	–	–
	1.0	2248.73	497.23	**0.014**	**0.001**
	3.0	**2520.53**	**88.78**	0.017	0.003
	5.0	1824.19	251.75	0.021	0.002
	10.0	1789.49	132.21	0.025	0.002
Walker2d-v4	–	3126.73	450.30	–	–
	1.0	**3325.74**	**254.65**	**0.054**	**0.003**
	3.0	2727.15	434.72	0.066	0.013
	5.0	2985.69	353.57	0.079	0.007
	10.0	2965.41	340.65	0.106	0.007
HalfCheetah-v4	–	1927.59	1030.66	–	–
	1.0	1868.40	697.62	**0.056**	0.008
	3.0	1818.68	**524.06**	0.080	0.020
	5.0	2035.49	1009.81	0.076	**0.007**
	10.0	**3089.20**	1053.22	0.100	0.009
LunarLanderCont.-v2	–	208.38	11.59	–	–
	1.0	242.33	28.06	**0.030**	0.007
	3.0	272.58	**7.21**	0.031	**0.005**
	5.0	**280.81**	7.32	0.036	0.006
	10.0	240.80	48.79	0.045	0.008
BipedalWalker-v3	–	235.09	19.03	–	–
	1.0	253.96	15.15	**0.059**	**0.004**
	3.0	**255.91**	12.31	0.075	0.006
	5.0	242.71	9.10	0.089	**0.004**
	10.0	233.00	**8.24**	0.101	**0.004**

Table 4.7: Effect of the loss function components on final episodic returns (EWMA) values and prediction error (CTE) The baseline is shown as the first row for each environment.

Environment	$\mathcal{L}_{\text{rec}}$	$\mathcal{L}_{\text{pred-ls}}$	$\mathcal{L}_{\text{pred-ss}}$	**EWMA**		**CTE**	
				Mean	Std.	Mean	Std.
InvertedPendulum-v4	–	–	–	897.57	41.61	–	–
	✓	–	–	897.21	57.50	0.927	0.137
	✓	✓	–	911.27	93.60	0.015	0.003
	✓	✓	✓	**998.18**	**3.57**	**0.001**	**0.000**
Hopper-v4	–	–	–	2315.43	226.87	–	–
	✓	–	–	2355.68	383.37	1.112	0.099
	✓	✓	–	2119.36	139.03	0.066	0.012
	✓	✓	✓	**2520.53**	**100.36**	**0.017**	**0.003**
Walker2d-v4	–	–	–	3126.73	450.30	–	–
	✓	–	–	2983.97	374.91	1.325	0.211
	✓	✓	–	2293.21	549.54	0.138	0.006
	✓	✓	✓	**3325.74**	**287.85**	**0.054**	**0.004**
HalfCheetah-v4	–	–	–	1927.59	1030.66	–	–
	✓	–	–	1852.74	811.86	1.021	0.047
	✓	✓	–	1869.47	**188.03**	0.269	**0.010**
	✓	✓	✓	**3089.20**	1203.42	**0.100**	0.011
LunarLanderCont.-v2	–	–	–	208.38	11.59	–	–
	✓	–	–	220.72	16.06	1.407	0.048
	✓	✓	–	198.33	52.01	0.054	0.017
	✓	✓	✓	**280.81**	**8.27**	**0.036**	**0.007**
BipedalWalker-v3	–	–	–	235.09	19.03	–	–
	✓	–	–	233.98	19.58	1.325	0.115
	✓	✓	–	214.13	38.04	0.186	0.026
	✓	✓	✓	**255.91**	**13.91**	**0.075**	**0.007**

Table 4.8: Hyperparameter values and ranges explored in the analysis.

Hyperparameter	**Values/Ranges**
Latent Dimension	16, 32, 48, 64
Prediction Horizon	1, 3, 5, 10
Number of Layers	1, 2, 3
Neurons per Layer	64, 128, 192, 256
Weight of $\mathcal{L}_{\text{rec}}$ (ω_1)	0.00, 0.05, ..., 1.00
Weight of $\mathcal{L}_{\text{pred-ls}}$ (ω_2)	0.00, 0.05, ..., 1.00
Weight of $\mathcal{L}_{\text{pred-ss}}$ (ω_3)	0.00, 0.05, ..., 1.00

Table 4.9: Color scale interpretation for box plots and contour plots comparing the effects of hyperparameters and their interactions.

Metric	**Color**	**Value**	**Compared to baseline**
Mean Final Returns	Blue	> 0	Better performance
	White	≈ 0	Similar
	Red	< 0	Worse performance
Std. Final Returns	Green	< 1	Lower variability
	White	≈ 1	Similar
	Purple	> 1	Higher variability

trained models, grouped by the specific hyperparameter being analyzed. The color scale, consistent with Table 4.9, indicates relative performance compared to the baseline.

An analysis of interactions between pairs of hyperparameters is presented in Section 4.3.3 using contour plots, providing additional insights into the complex relationships between them.

Representation Loss Weights

This section examines the hyperparameters controlling the relative importance of the three components in the representation loss function: ω_1, ω_2, and ω_3. Each weight ranges from 0 to 1.0 in steps of 0.05.

Figure 4.4 investigates how prioritizing reconstruction accuracy, governed by ω_1, impacts the mean and standard deviation of final returns. KIPPO demonstrates robustness to this hyperparameter, as all values achieve median mean returns and interquartile ranges above the baseline. Values between 0.25 and 0.5, however, show the most compact distribution of mean returns, suggesting an optimal range. Excessively emphasizing reconstruction does not lead to significant performance differences.

Figure 4.5 delves into the effects of varying the emphasis on latent-space prediction, controlled by ω_2. Values below 0.5 consistently outperform the baseline in mean returns, and those as low as around 0.25 achieve the best performance. In terms of std., values between 0.5 and 0.75 show slight improvements over the baseline, while values below 0.25 are comparable to the baseline.

Figure 4.6 explores the impact of state-space prediction loss weight, controlled by ω_3. Values above 0.25 consistently outperform the baseline in terms of mean, but retain similar variability to the baseline. Values around or below 0.25 show similar mean returns to the baseline, but with slightly increased variability. Note that the numerical scales and ranges of the state space and latent space are different.

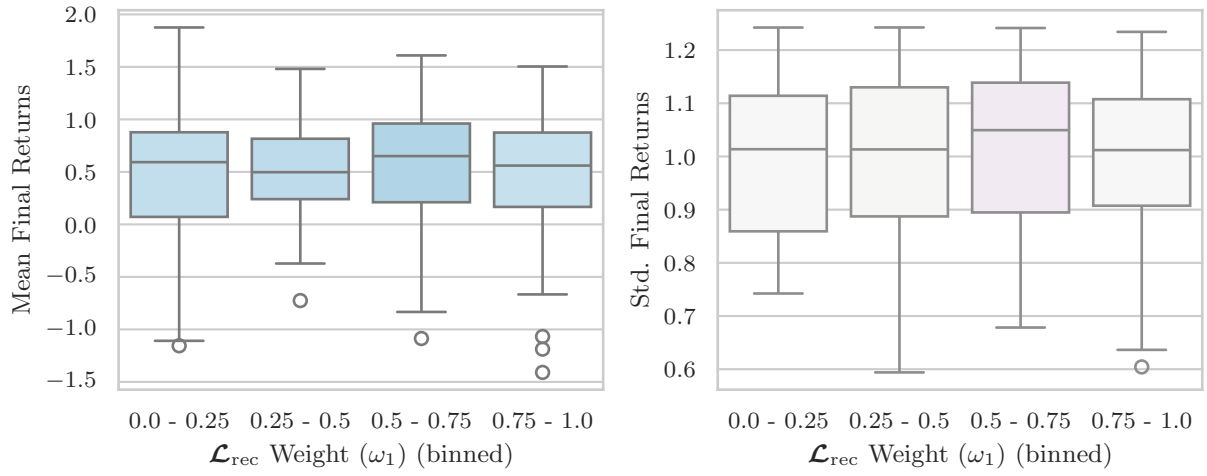

Figure 4.4: Box plots visualizing the impacts of the reconstruction loss weight ω_1 on the mean and std. of final returns, showing generally consistent performance across the range of values.

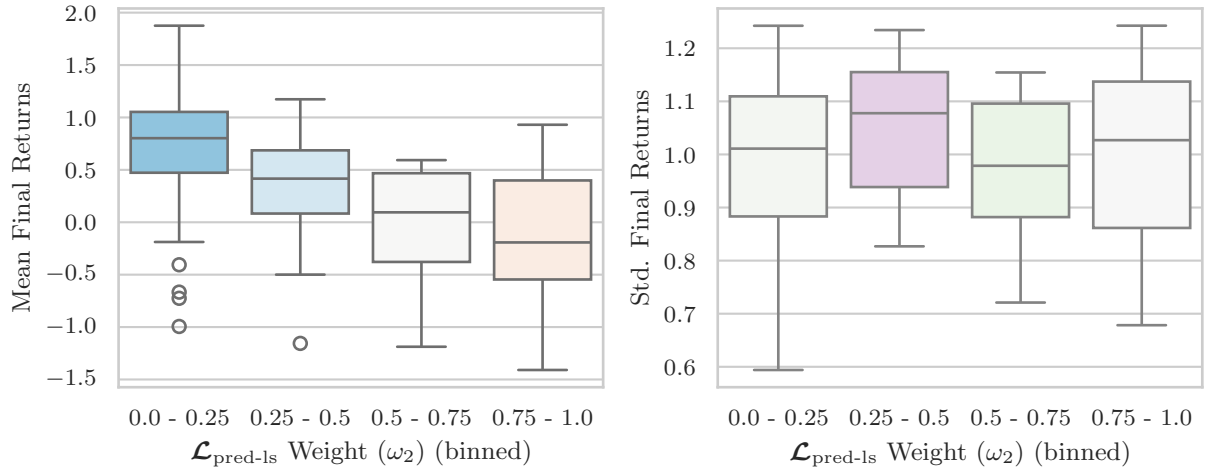

Figure 4.5: Box plots showing the impacts of the latent-space prediction loss weight ω_2 on the mean and std. of final returns, revealing that values below 0.5 outperform the baseline in mean returns.

Latent Dimension and Prediction Horizon

This section examines two key hyperparameters that directly influence the learned representation: the latent dimension (16 to 64) and prediction horizon (1 to 10).

Figure 4.7 analyzes the effects of varying the latent dimension on KIPPO's performance. Values at 32 or above consistently outperform the baseline in mean returns. A value of 48 shows significant improvements in both mean returns and variability across all environments. Performance degrades with the lowest latent dimension of 16, indicating insufficient capacity to learn effective representations.

Figure 4.8 investigates the impact of the prediction horizon H. All values achieve median mean returns above the baseline. The compact distribution at a horizon of 3 makes this value a good starting point for optimization. KIPPO's performance variability appears relatively insensitive to this hyperparameter. The lack of a clear trend could be due to specific values being environment-specific, or additional interactions with other hyperparameters masking the independent effects of the prediction horizon.

State Autoencoder and Action Encoder Architecture

This section examines the architectural hyperparameters of the state autoencoder and action encoder: the number of layers (1 to 3) and neurons per layer (64 to 256).

Figure 4.9 explores the impact of network depth, varied by the number of layers. The consistent performance across different numbers of layers indicates that KIPPO is relatively robust to moderate variations in network depth.

Figure 4.10 delves into the influence of network width, determined by the number of neurons per layer. Higher neuron counts consistently achieve better performance, indicating the importance of adequate model capacity for representation learning. A trade-off emerges between 192 and 256 neurons: 192 achieves lower variability at the cost of a wider distribution in mean returns. The significantly worse performance at 64 neurons implies insufficient capacity to learn effective representations.

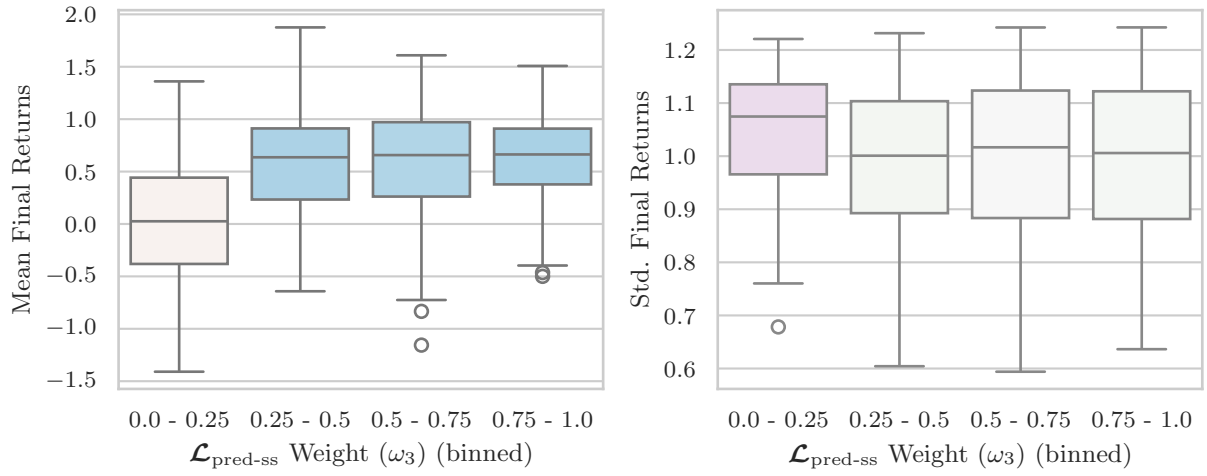

Figure 4.6: Box plots visualizing the effects of the state-space prediction loss weight ω_3 on the mean and std. of final returns, showing consistent improvements over the baseline for values above 0.25.

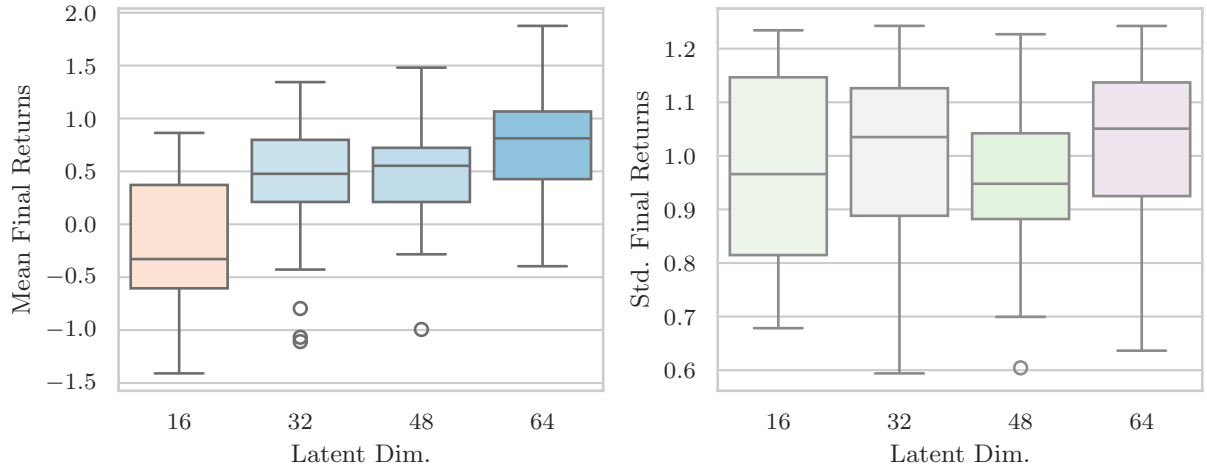

Figure 4.7: Box plots showing the impact of the latent dimension on the mean and std. of final returns, showing that all values above 32 provide improved means, and a value of 48 showing improvements in both aspects.

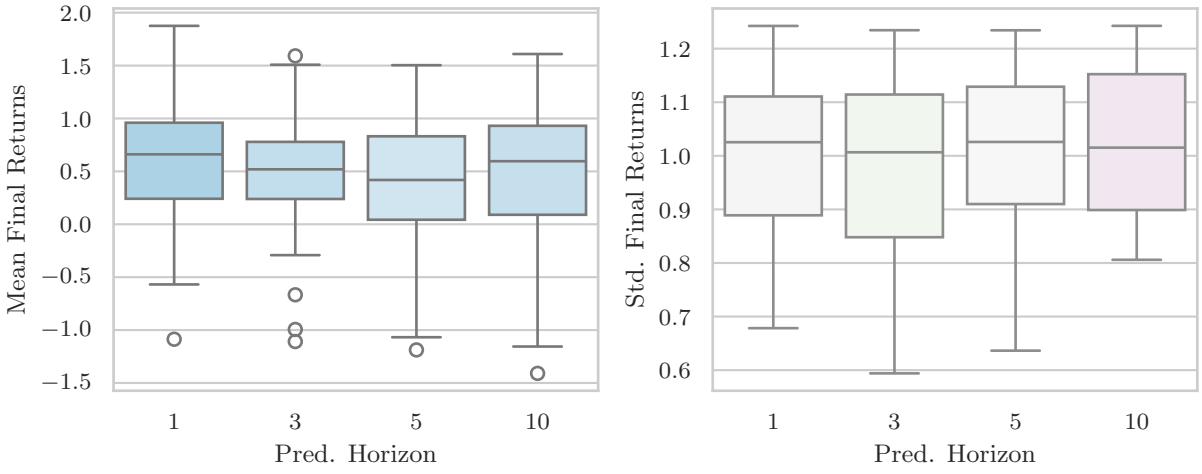

Figure 4.8: Box plots visualizing the effects of the prediction horizon H on the mean and std. of final returns, showing no significant differences in performance across values.

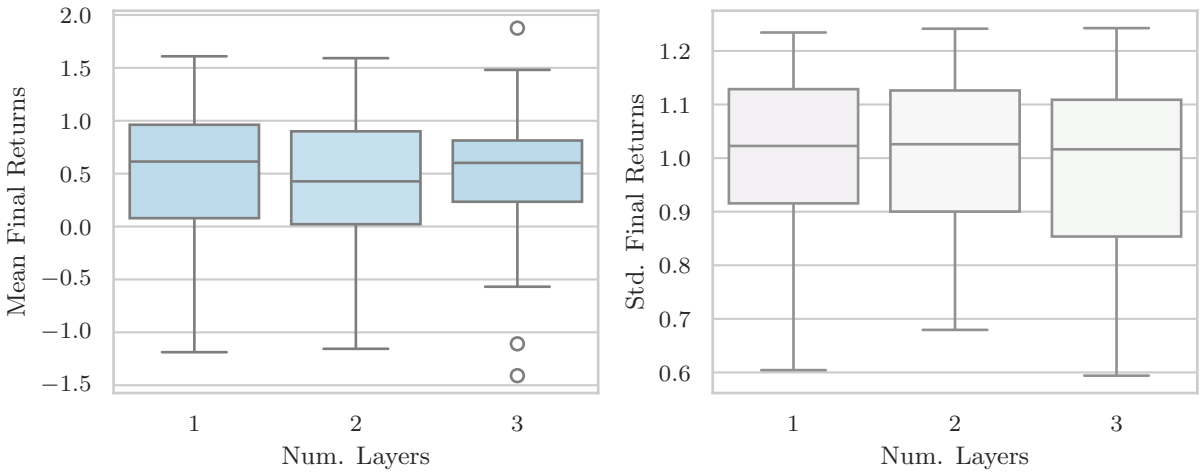

Figure 4.9: Box plots showing the impact of network depth on the mean and std. of final returns, showing relatively consistent performance across all values.

4.3.3 Analysis of Hyperparameter Interactions

While examining individual hyperparameters provides valuable insights, understanding their interactions is crucial for effective optimization of the KIPPO algorithm. This section uses contour plots to visualize these interactions, revealing trends and patterns that might not be apparent from the individual box plots presented in Section 4.3.2.

The contour plots display the joint effects of pairs of hyperparameters on the standardized performance metrics: mean final returns and standard deviation of final returns. As with the previous section, these plots integrate results from all 7,200 models trained across the six environments and four initializations. The color scale used in these plots, indicating relative performance compared to the baseline, is consistent with that described in Table 4.9.

Interactions between Representation Loss Weights

This section analyzes pairwise interactions between the hyperparameters weighting the three components of the representation loss function: ω_1, ω_2, and ω_3. Each weight ranges from 0 to 1.0 in steps of 0.05.

Figure 4.11 investigates the interplay between the reconstruction loss weight ω_1 and the latent-space prediction loss weight ω_2. Notably, the majority of the area below a ω_2 value of 0.4 outperforms the baseline in mean returns, irrespective of ω_1. Over-emphasizing both losses simultaneously leads to suboptimal performance, as seen especially in the upper-right region of the plot. There is no clear trend trend in standard deviation, though the same area shows roughly equal areas of improvement and degradation.

Figure 4.12 explores the combined effects of the reconstruction loss weight ω_1 and the state-space prediction loss weight ω_3. Most of the area with ω_3 values above 0.3 outperforms the baseline in mean returns, regardless of the ω_1 value. Moderate values between 0.25 and 0.75 for both weights lead to considerable improvements in mean returns. Values in the same ranges also have a high probability of achieving lower variability compared to the baseline.

Figure 4.13 analyzes the interplay between the latent-space prediction weight (ω_2) and state-space prediction loss weight ω_3. Notably, the upper-left region with ω_2 values below 0.4

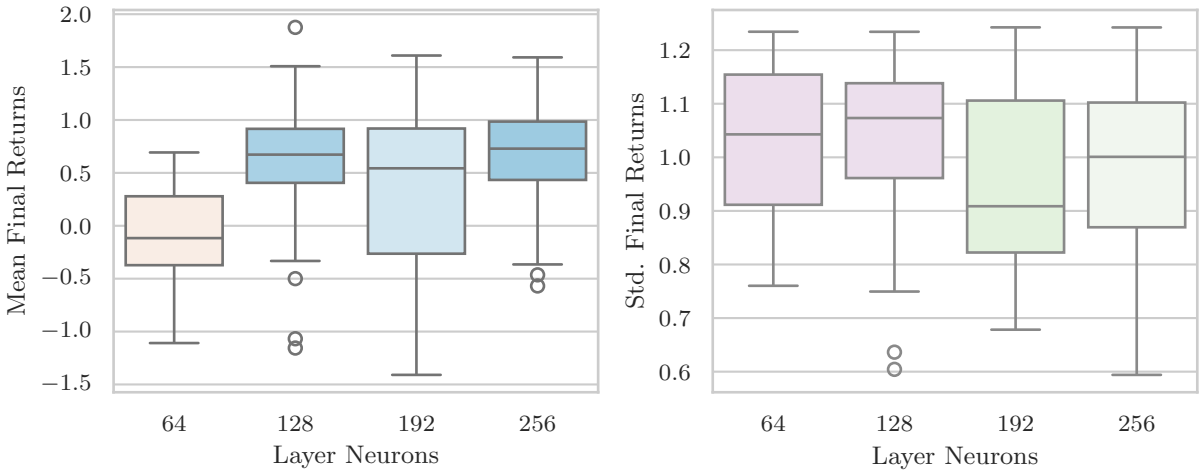

Figure 4.10: Box plots showing the effects of network width on the mean and std. of final returns, where values between 192 and 256 neurons show improved means and a value of 192 also shows lower variability.

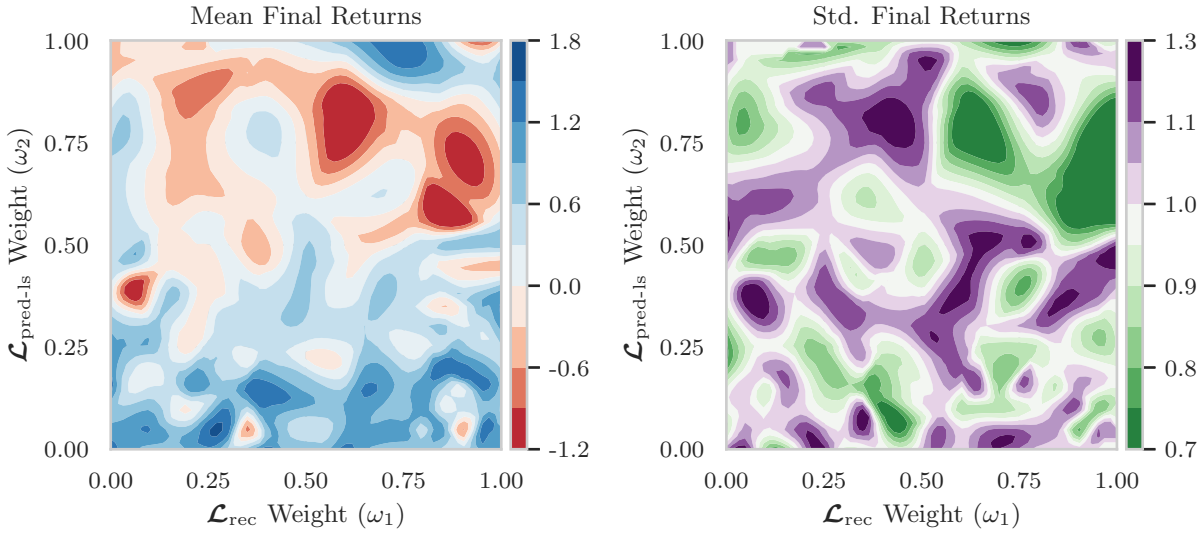

Figure 4.11: Contour plots visualizing the interaction between loss weights for reconstruction ω_1 and latent-space prediction ω_2 on the mean and std. of final returns, revealing consistent improvements with ω_2 below 0.4 regardless of ω_1.

and ω_3 values above 0.3 lead to considerable improvements in mean returns, which consistent with the insights from the previous plots. The same area on the standard deviation plot shows improvements in variability, though there are still some areas with slight increases On the other hand, the lower-right region with ω_2 values above 0.6 and ω_3 values below 0.25 consistently underperform the baseline in mean returns.

Interactions between Latent Dimensions and Loss Weights

This section investigates interactions between each representation loss weight and the latent dimension (ranging from 16 to 64). Loss weights range from 0 to 1.0 in steps of 0.05.

Figure 4.14 explores the relationship between the reconstruction loss weight ω_1 and the latent dimension. Latent dimensions above 16 are relatively consistent in outperforming the baseline in mean returns, with the best results generally achieved at 48. A value of 48 also shows the most area with improved standard deviations, indicating better consistency across different initializations.

Figure 4.15 investigates the combined effects of the latent-space prediction loss weight ω_2 and the dimensionality of the latent space. The results show that ω_2 values below 0.4 consistently yield better mean returns, especially for latent dimensions at or above 32. Using a latent dimension of 48 with a range of ω_2 values below 0.25 achieves improvements in both mean returns and variability.

Figure 4.16 examines the interplay between the state-space prediction loss weight ω_3 and the latent space dimensionality. Similar to previous results, areas above 0.3 for ω_3 consistently achieve better mean returns compared to baseline for all latent dimensions other than 16. A latent dimension of 48 achieves the most consistent improvements in variability across the entire range of ω_3 values. A notable shared range of optimal performance in both metrics is observed between ω_3 values of 0.3 and 0.55 and latent dimensions 48. Overall, these insights support that a latent dimension of 48 is the best choice across all six environments evaluated.

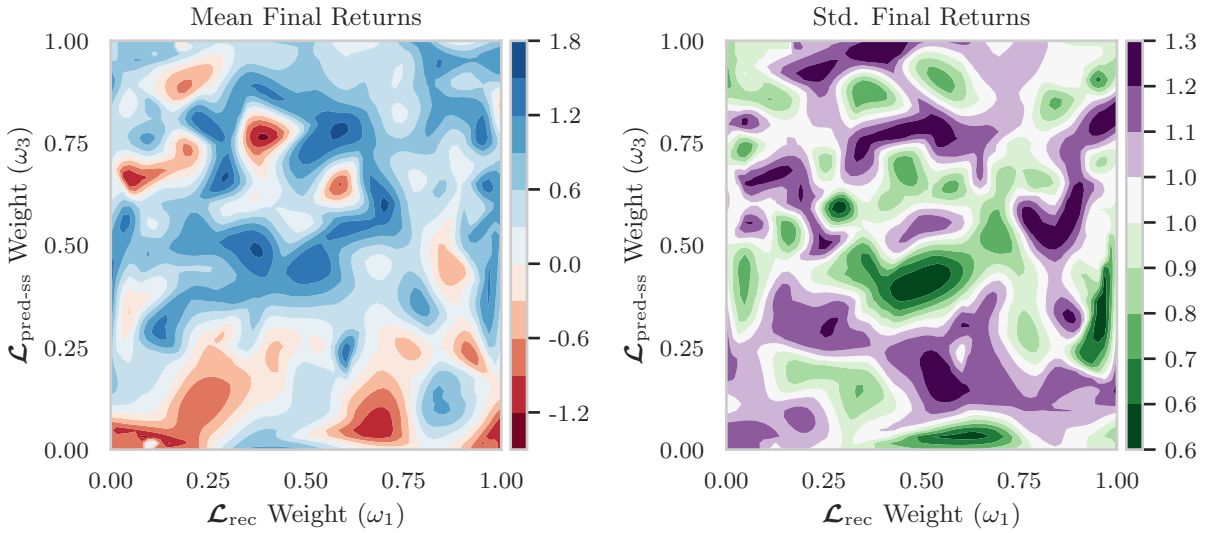

Figure 4.12: Contour plots visualizing the interaction between loss weights for reconstruction ω_1 and state-space prediction ω_3 on the mean and std. of final returns, suggesting that moderate values between 0.25 and 0.75 for both weights generally leads to considerable improvements.

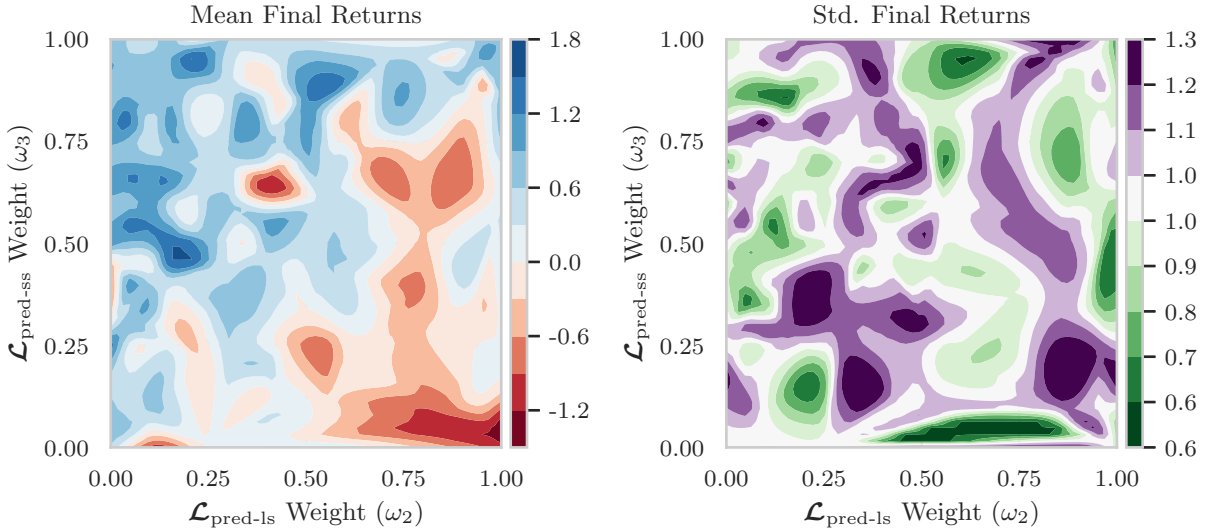

Figure 4.13: Contour plots visualizing the interaction between loss weights for prediction in the latent-space ω_2 and state-space ω_3 on the mean and std. of final returns, showing ω_2 below 0.4 and ω_3 above 0.3 lead to considerable improvements.

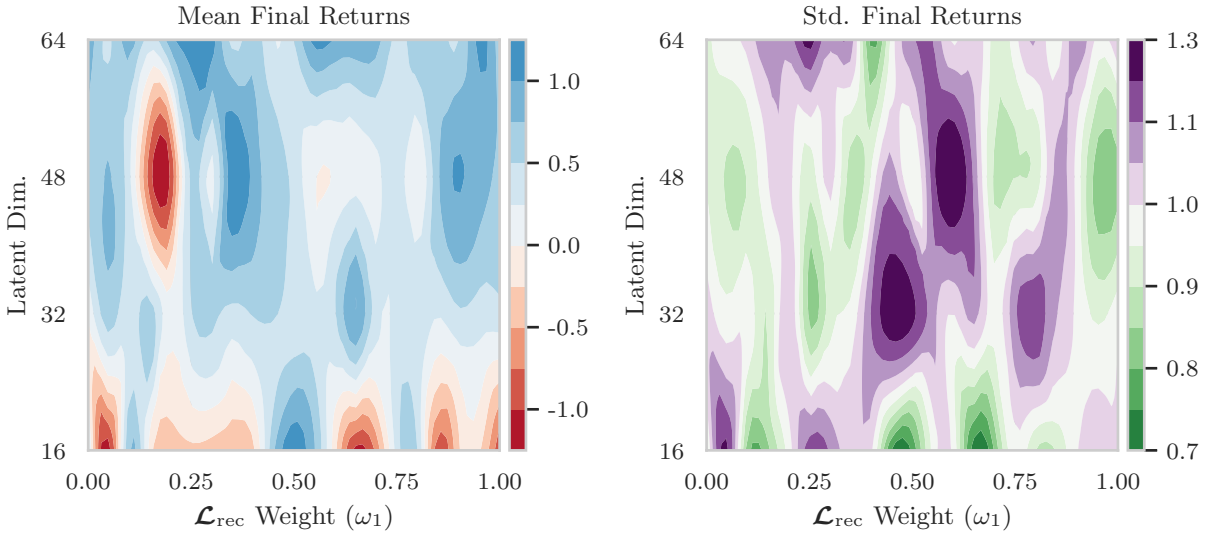

Figure 4.14: Contour plots visualizing the interaction between the reconstruction loss weight ω_1 and latent dimension on the mean and std. of final returns, revealing consistent improvements with higher latent dimensions and moderate to high reconstruction weight.

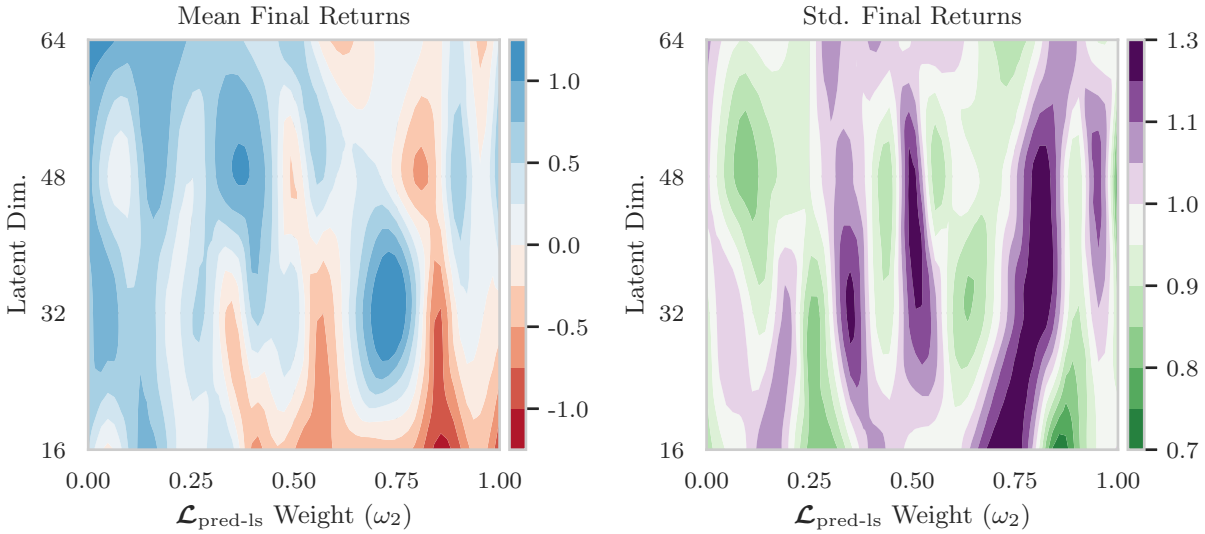

Figure 4.15: Contour plots visualizing the interaction between the latent-space prediction loss weight ω_2 and latent dimension on the mean and std. of final returns, demonstrating improved means with ω_2 below 0.4 and improved variability with a latent dimension at or above 32.

Interactions between Prediction Horizons and Loss Weights

This section analyzes interactions between each representation loss weight and the prediction horizon (ranging from 1 to 10). Loss weights range from 0 to 1.0 in steps of 0.05.

Figure 4.17 investigates the interplay between the reconstruction loss weight ω_1 and the length of future predictions (prediction horizon). The analysis reveals that horizons between 1 and 3 demonstrate the most consistent improvements in standardized mean returns, regardless of ω_1. Interestingly, a horizon of 3 exhibits the most areas with improved standard deviations, though still with some exceptions. Low weights of the reconstruction loss below 0.25 coupled with horizons of 5 or above consistently underperform the baseline in both metrics. However, increasing the reconstruction loss weight above 0.5 leads to more consistent improvements in mean returns. These findings emphasize the importance of the reconstruction loss on long-term prediction accuracy.

Figure 4.18 analyzes the combined influence of the latent-space prediction loss weight ω_2 and the prediction horizon. The results show that lowering either ω_2 below 0.4 or the horizon below 5 consistently improves mean returns. An interesting observation is the shared peak in both metrics with a horizon of 3 and ω_2 between 0.55 and 0.8, indicating considerable improvements. Values of ω_2 above 0.5 with horizons above 5 significantly degrade performance.

Figure 4.19 explores the relationship between the state-space prediction loss weight ω_3 and the prediction horizon. The analysis shows that increasing ω_3 above 0.25 generally leads to improvements, particularly with a horizon of 3. While higher horizons show good improvements in mean returns, variability is inconsistent with longer horizons. Notable peaks are observed for ω_3 ranges between 0.3 and 0.75, coupled with a horizon of 3. Values of ω_3 below 0.25 generally leads to poor standard deviations, and values below 0.1 result in extremely poor mean episodic returns, regardless of horizon.

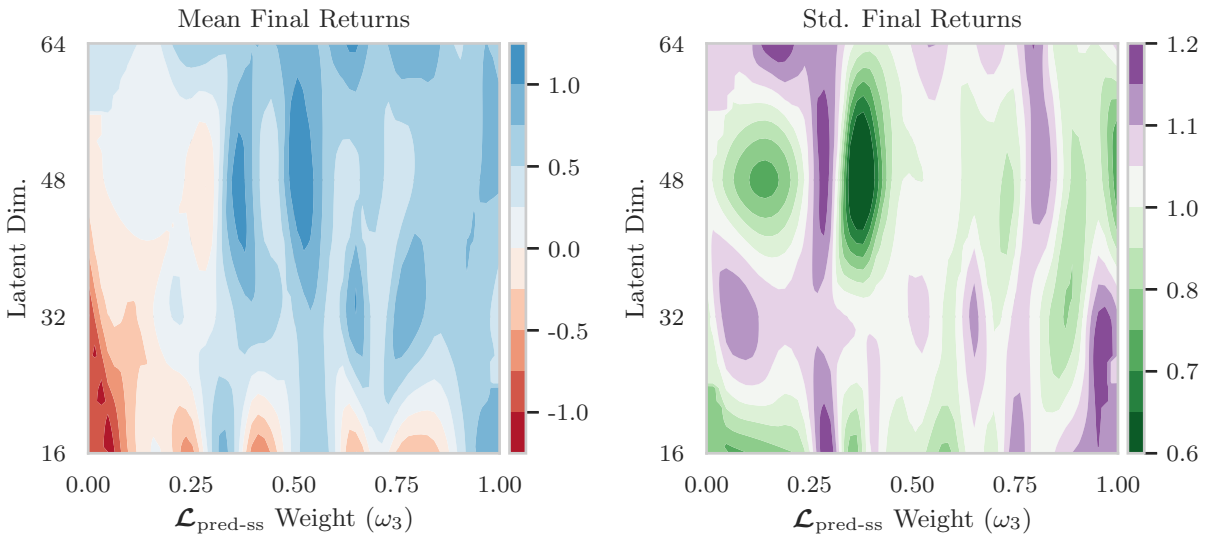

Figure 4.16: Contour plots visualizing the interaction between the state-space prediction loss weight ω_3 and latent dimension on the mean and std. of final returns, showing the benefits of higher latent dimensions and moderate to high values of ω_3.

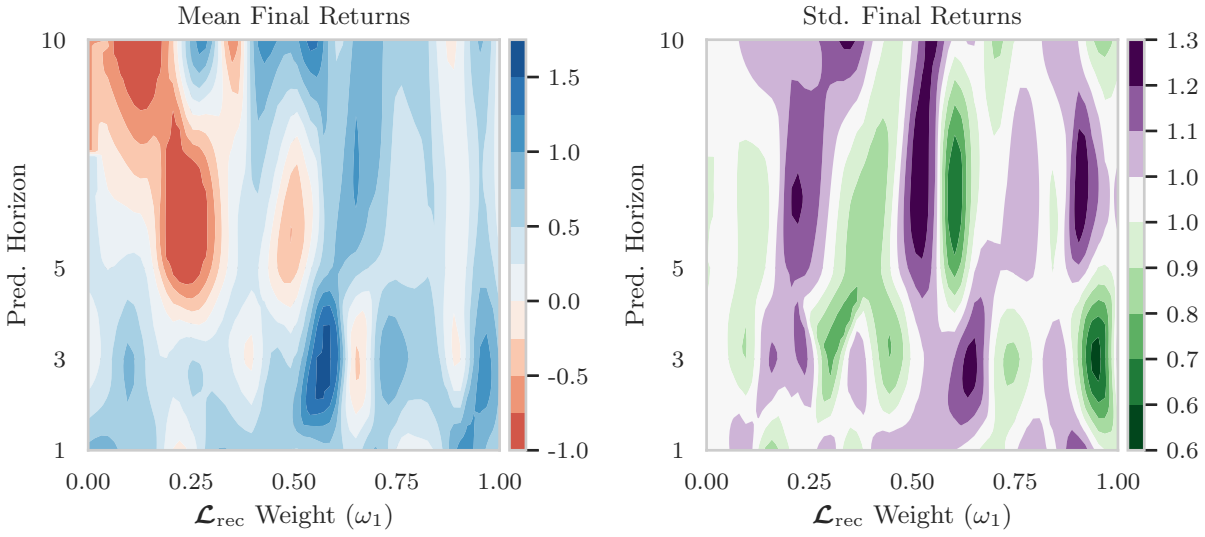

Figure 4.17: Contour plots visualizing the interaction between the reconstruction loss weight ω_1 and prediction horizon H on the mean and std. of final returns, showing that either shorter horizons or higher reconstruction loss weights lead to improved performance.

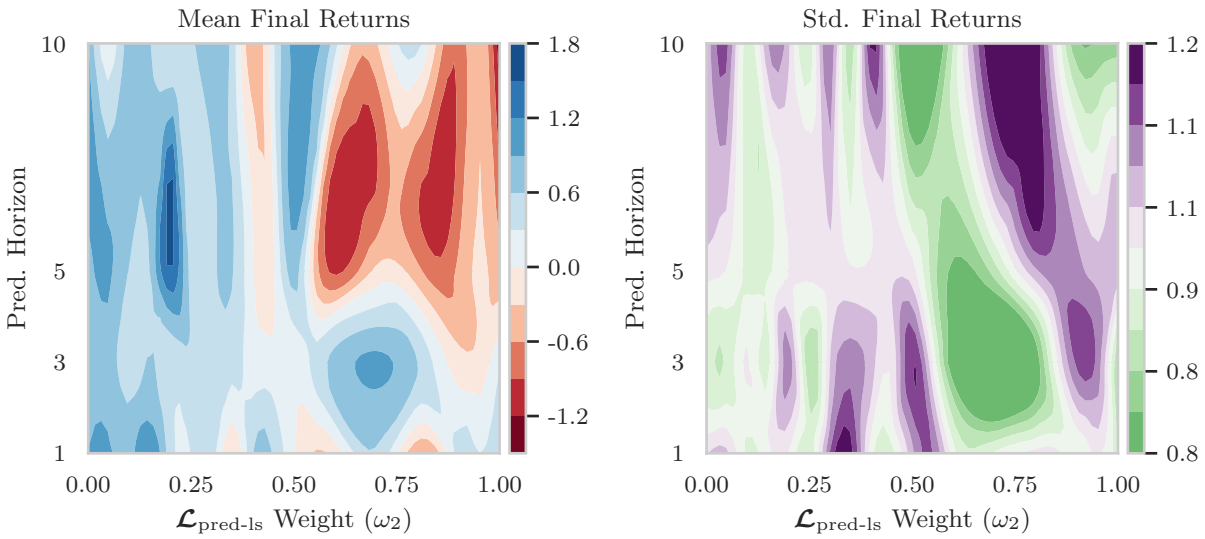

Figure 4.18: Contour plots visualizing the interaction between the latent-space prediction loss weight ω_2 and prediction horizon H on the mean and std. of final returns, which shows that either lower ω_2 or shorter horizons lead to improved performance.

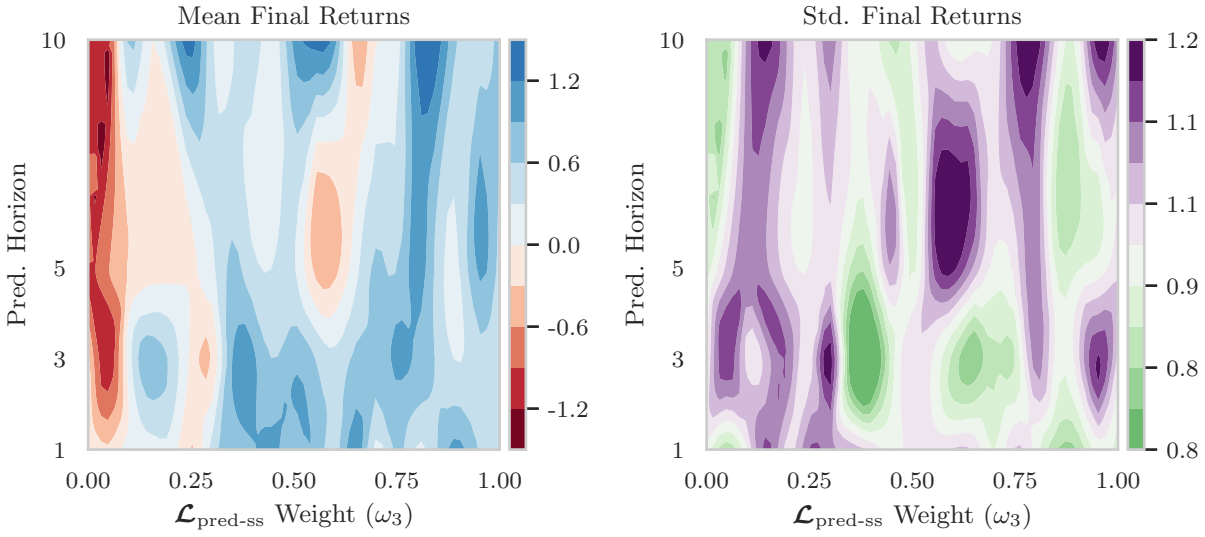

Figure 4.19: Contour plots visualizing the interaction between the state-space prediction loss weight ω_3 and prediction horizon H on the mean and std. of final returns, showing the best regions of improvements with ω_3 between 0.3 and 0.75 and a horizon of 3.

Miscellaneous Hyperparameter Interactions

This section analyzes interactions between the number of layers and neurons per layer in the state autoencoder and action encoder, the latent dimension, and the prediction horizon. The number of layers ranges from 1 to 3, neurons per layer from 64 to 256, and prediction horizon from 1 to 10.

Figure 4.20 delves into the interplay between network depth, determined by the number of layers, and the prediction horizon. The analysis reveals that a prediction horizon of 3 with 2 or 3 layers yields the best results across both metrics. Interestingly, mean performance appears relatively invariant to the number of layers, while standard deviation shows a slight degradation with 1 layer. Increasing the horizon above 5 generally leads to worse variability especially with higher numbers of layers.

Figure 4.21 examines the combined effects of network width, controlled by the number of neurons per layer, and the prediction horizon. The results show that neuron counts above 64 consistently improve mean returns, regardless of the prediction horizon. Notably, 192 neurons consistently show improved standard deviation and mean returns regardless of horizon, indicating a potentially optimal network width. The analysis reveals a shared peak with 192 neurons and a horizon of 3, showing considerable improvements in both metrics.

Figure 4.22 analyzes the interplay between the prediction horizon and the latent dimension. The results show that a 48-dimensional latent space achieves improvements in both metrics, relatively invariant to the prediction horizon. This suggests that a sufficiently expressive latent space can maintain good performance across various prediction horizons. However, for lower dimensions of the latent space, higher horizons (5-10) show considerably worse variability.

Figure 4.23 explores the relationship between network depth (number of layers) and width (neurons per layer). The analysis reveals that all neuron counts above 64 show improved mean returns, reinforcing the importance of sufficient network capacity. Interestingly, the standard deviation plot shows a split: values above 192 improve variability, while values below 128 worsen it, relatively independent of the number of layers. This suggests that

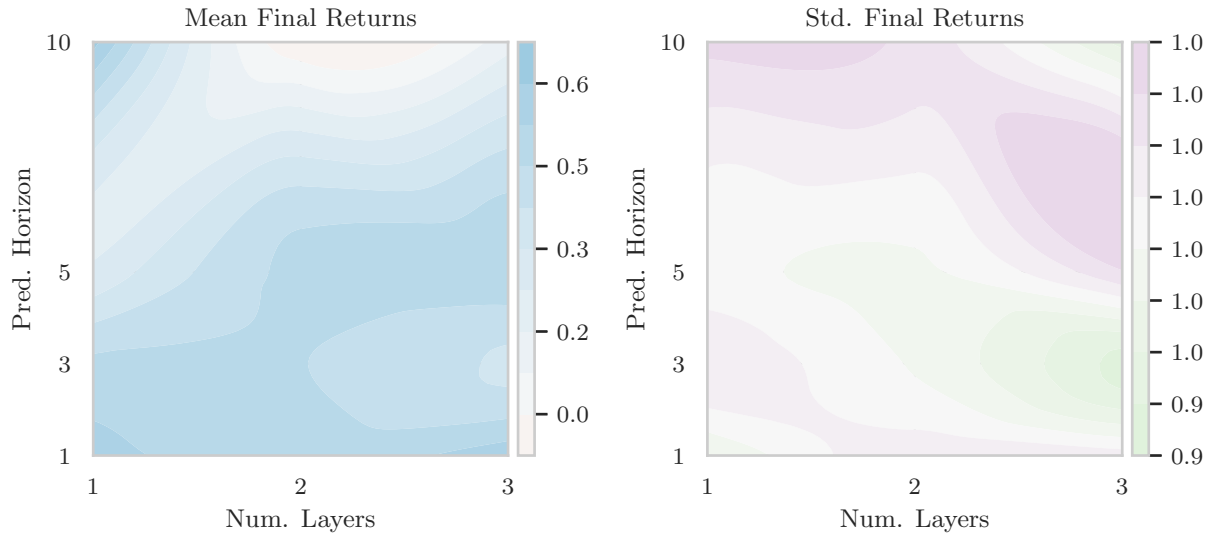

Figure 4.20: Contour plots visualizing the interaction between the number of layers and prediction horizon H on the mean and std. of final returns, demonstrating that network depth has minor impacts on performance, while relatively short horizons lead to best results across all environments.

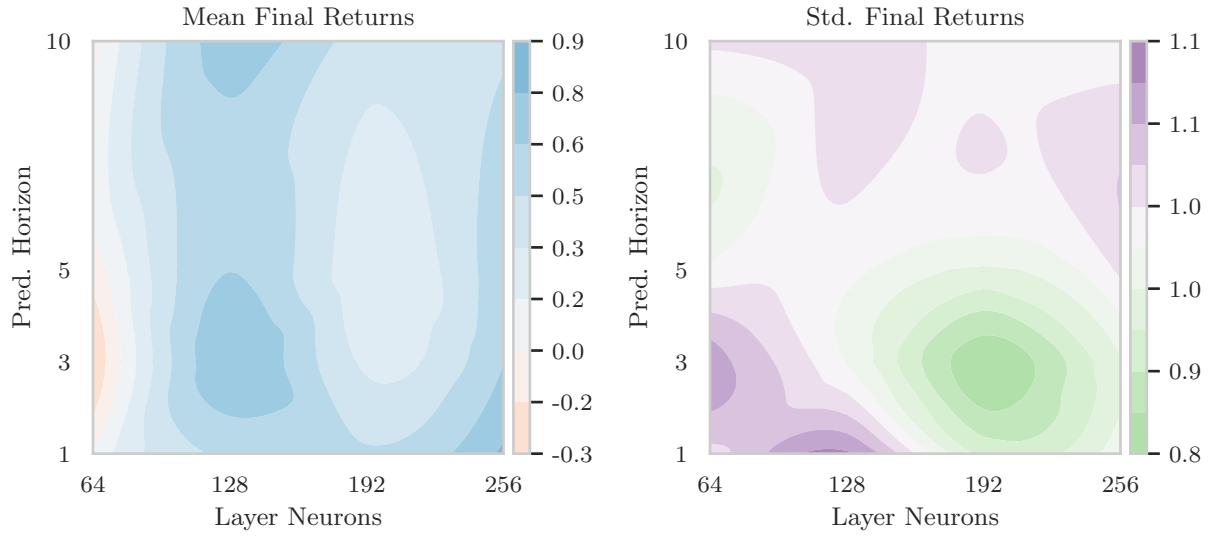

Figure 4.21: Contour plots visualizing the interaction between the number of neurons per layer and prediction horizon H on the mean and std. of final returns, revealing the benefits of increased network widths with moderate prediction horizons.

network width plays a more crucial role in performance variability than depth. The findings indicate that prioritizing network width over depth may be a more effective strategy for improving both performance and consistency.

4.4 Summary and Discussion

This chapter evaluated KIPPO, an algorithm integrating Koopman-inspired representation learning with PPO. The experiments tested the hypothesis that leveraging Koopman theory to simplify complex environment dynamics can lead to more effective and robust policy optimization.

Experiments were conducted on six diverse continuous control environments from the Gymnasium library, using MuJoCo and Box2D physics engines. These environments ranged from simple balancing tasks to complex locomotion challenges. The results show that KIPPO achieved:

- Higher mean performance in all environments, with improvements ranging from 6.36% to 60.26%.
- Lower standard deviation in five out of six environments, with reductions ranging from 26.89% to 91.43%.

The ablation study revealed that integrating all three loss components—reconstruction, latent-space prediction, and state-space prediction—yielded the best mean EWMA and lowest standard deviation in five out of six environments. The mean CTE consistently decreased as more loss components were incorporated, validating the quality of the learned representation and the effectiveness of the multi-objective approach.

A key finding of this study is that KIPPO can achieve consistent improvements in both mean and standard deviation of final returns across all six evaluation environments with a single set of hyperparameters. The best hyperparameter ranges identified are:

- Latent-space prediction loss weight around or below 0.25 and state-space prediction loss weight between 0.3 and 0.85

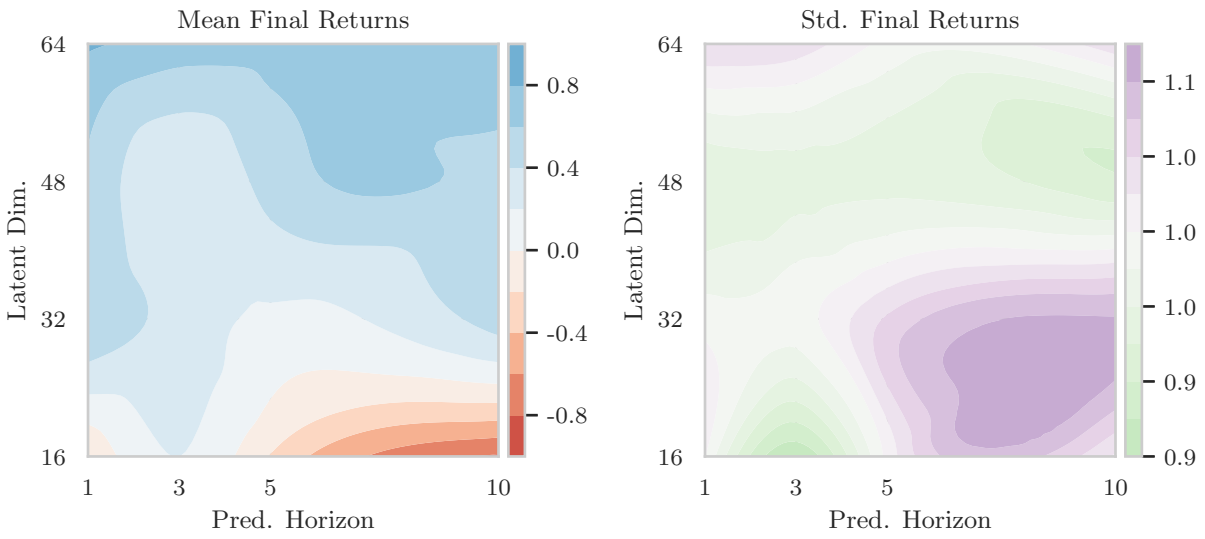

Figure 4.22: Contour plots visualizing the interaction between the prediction horizon H and latent dimension on the mean and std. of final returns, showing the robustness of higher latent dimensions across different prediction horizons.

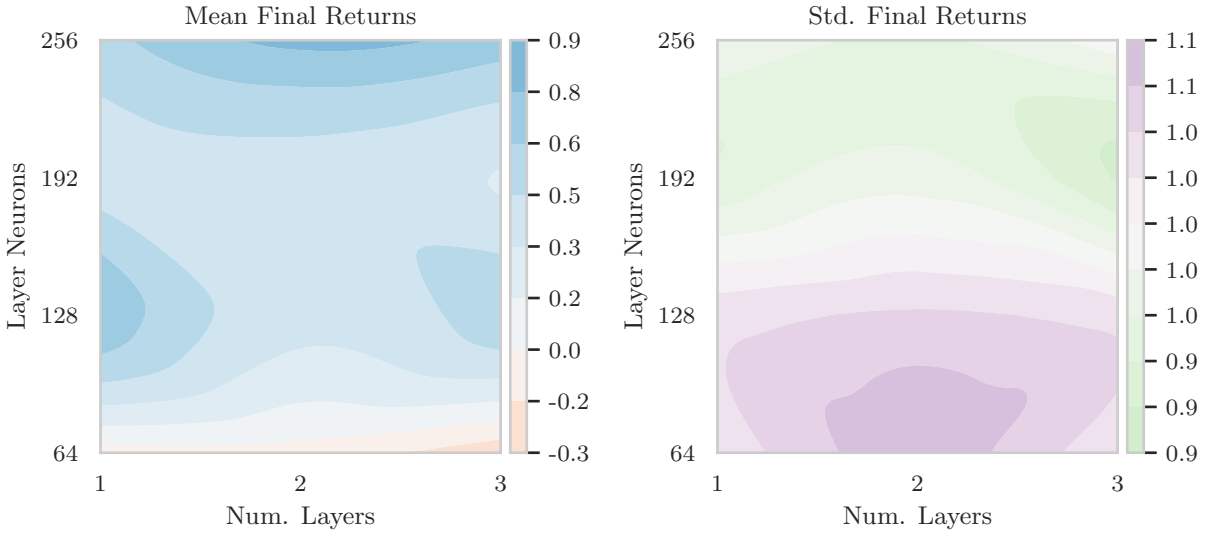

Figure 4.23: Contour plots visualizing the interaction between the number of layers and the number of neurons per layer on the mean and std. of final returns, highlighting the importance of network width over depth for both performance and variability.

- Reconstruction loss weight between 0.3 and 0.75
- Latent dimension of 48 and a prediction horizon of 3
- Network architecture with 2 or 3 layers and between 128 and 192 neurons per layer

This consistency across diverse environments demonstrates the robustness and versatility of the KIPPO approach. It suggests that the algorithm can effectively learn meaningful representations and policies across a range of control tasks without requiring extensive task-specific tuning.

- The multi-objective optimization approach, balancing reconstruction, latent-space prediction, and state-space prediction, consistently benefits representation learning across diverse environments.
- The identified hyperparameter ranges provide good starting points for applying KIPPO to new tasks, potentially reducing the need for extensive hyperparameter tuning.
- The robustness to certain architectural choices, such as the number of layers, may simplify the deployment of KIPPO in various applications.

By demonstrating improvements in both performance and robustness across a range of continuous control tasks with a consistent set of hyperparameters, KIPPO represents a significant step forward in developing effective and generalizable reinforcement learning algorithms for complex control problems. Its ability to maintain strong performance across diverse environments with a single hyperparameter configuration highlights its potential for broad applicability in real-world scenarios. Future research directions building on these promising findings are discussed in the conclusion chapter.

Chapter 5

Conclusions and Future Works

5.1 Overall Summary

This thesis introduces Koopman-Inspired Proximal Policy Optimization (KIPPO), a novel framework that integrates Koopman-inspired representation learning with the Proximal Policy Optimization (PPO) algorithm to address the challenges of policy optimization in environments with complex, nonlinear dynamics. By learning a latent-space representation that simplifies the underlying system dynamics while retaining essential information, KIPPO enables more stable and effective policy learning.

The proposed approach incorporates auxiliary components, including a state autoencoder, action encoder, and linear state-transition and control matrices, to learn a Koopman-inspired latent-space representation. The learning process is guided by carefully designed objectives and constraints, such as the latent-space prediction loss, state-space prediction loss, and reconstruction loss, which ensure a balance between simplifying the environment representation and preserving crucial information for policy optimization.

Comprehensive experiments on a diverse set of continuous control tasks from the MuJoCo and Box2D environments demonstrate the effectiveness of KIPPO in improving policy optimization performance and variability. Compared to the state-of-the-art PPO baseline, the proposed framework consistently achieves higher mean returns and lower variability, with

improvements ranging from 6% to 60% in terms of average performance and a reduction in variability ranging from 26% to 91%.

The ablation study of the loss function components highlights the importance of the proposed multi-objective optimization approach. The results demonstrate that integrating all three loss components consistently yields the best performance and highest consistency in terms of returns, emphasizing the significance of balancing the objectives of reconstruction, simplification, and consistency with the true dynamics for learning a meaningful and effective latent-space representation.

The extensive hyperparameter analysis, including the examination of individual hyperparameter effects and their interactions, provides valuable insights into the behavior of KIPPO. This analysis reveals the relative importance of different hyperparameters, with the latent-space prediction loss weight and latent dimension being particularly influential. Increasing the latent dimension generally improves returns and reduces prediction error, suggesting that more expressive representations are beneficial for capturing complex dynamics and learning effective policies. These findings provide guidance for future applications and improvements of the algorithm, contributing to a better understanding of how different parameters affect performance across various tasks.

5.2 Limitations and Future Directions

While the current set of environments provides a solid foundation for evaluation, future work could extend to a broader range of continuous control tasks and other benchmarking suites to further assess generalizability, robustness, and scalability to even more complex and high-dimensional environments. Adapting the framework to handle vision-based tasks, where the state-space is represented by high-dimensional images, would require modifications to the state autoencoder component, such as incorporating convolutional neural networks to process visual input effectively.

Future research could also extend the proposed framework to other Reinforcement Learning (RL) algorithms beyond PPO. Integration with on-policy algorithms, such as Robust

Policy Optimization (RPO), Trust Region Policy Optimization (TRPO), or Advantage Actor-Critic (A2C), would be relatively straightforward due to shared characteristics with PPO. However, incorporating off-policy algorithms like Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), or Soft Actor-Critic (SAC) may require additional considerations, particularly regarding the efficient storage and use of prediction sequences in replay buffers.

Future work could also investigate incorporating additional constraints or regularization techniques that explicitly promote interpretability, as well as methods for analyzing and visualizing the learned latent-space. Examining the robustness of the learned representation to noise and its performance in out-of-distribution scenarios could provide valuable insights into its stability and generalization capabilities.

While the proposed framework introduces some additional complexity during training, it's important to note that once the model is trained, only the state encoder needs to be retained afterward. This means that the learned representation can be efficiently used for inference without additional prediction steps, minimizing memory and computational overhead during deployment.

5.3 Implications and Concluding Remarks

The development of KIPPO and its demonstrated improvements in performance and stability have important implications for the field of RL and its potential real-world applications. By integrating insights from Koopman Operator Theory with the PPO algorithm, this work explores the potential of leveraging dynamical systems theory to address the challenges of learning effective control policies in complex environments.

The proposed framework's ability to learn latent-space representations that simplify the underlying system dynamics while preserving essential information suggests a promising approach for tackling RL problems in various domains. The enhanced stability of learned policies, as demonstrated in the experimental results, may lead to more reliable and predictable agent behavior, which is crucial for safety-critical applications.

The insights gained from the analysis of key hyperparameters and the ablation study of loss function components provide valuable guidance for future research in representation learning for RL. The findings highlight the importance of carefully designing objectives and constraints that balance the simplification of the environment representation with the preservation of crucial information for policy optimization.

In conclusion, KIPPO represents a step forward in addressing the challenges of policy optimization in environments with complex, nonlinear dynamics. By demonstrating improvements in performance and stability across a range of challenging tasks, this work showcases the potential benefits of integrating Koopman-inspired representation learning with state-of-the-art RL algorithms. The implications of this research extend to a variety of real-world problems where efficient and robust policy learning is important, contributing to the ongoing exploration of leveraging dynamical systems theory to advance RL research and its applications in real-world decision-making and control.

Bibliography

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503 – 515. 18

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684. 18

Bellman, R., Corporation, R., and Collection, K. M. R. (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press. 18

Broer, H. and Takens, F. (2010). *Dynamical Systems and Chaos*. Applied Mathematical Sciences. Springer New York. 2, 8, 9

Brunton, S. and Kutz, J. (2022). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press. 8, 9, 11, 12

Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. (2021). Modern koopman theory for dynamical systems. 3, 16, 34

Budišić, M., Mohr, R., and Mezić, I. (2012). Applied koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4). 14, 15, 32, 34

Catto, E. (2007). Box2d: A 2d physics engine for games. https://box2d.org/. Accessed: October 10, 2023. 46

Demo, N., Tezzele, M., and Rozza, G. (2018). Pydmd: Python dynamic mode decomposition. *Journal of Open Source Software*, 3(22):530. 17

Devaney, R. (2021). *An Introduction To Chaotic Dynamical Systems*. CRC Press. 8, 9

Dey, S. and Davis, E. (2023). Dlkoopman: A deep learning software package for koopman theory. 16, 17

Freidlin, M., Szucs, J., and Wentzell, A. (2012). *Random Perturbations of Dynamical Systems*. Grundlehren der mathematischen Wissenschaften. Springer New York. 8

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. 24

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR. 34

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 25

Han, Y., Hao, W., and Vaidya, U. (2020). Deep learning of koopman representation for control. 3, 29

Heij, C., Ran, A., and van Schagen, F. (2021). *Introduction to Mathematical Systems Theory: Discrete Time Linear Systems, Control and Identification*. Springer International Publishing. 2, 9

Holkar, K. and Waghmare, L. M. (2010). An overview of model predictive control. *International Journal of control and automation*, 3(4):47–63. 12

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18. 41, 49

Janner, M., Fu, J., Zhang, M., and Levine, S. (2021). When to trust your model: Model-based policy optimization. 2, 28

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press. 2

Koopman, B. O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318. 3

Kutz, J. N., Brunton, S. L., Brunton, B. W., and Proctor, J. L. (2016). *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Philadelphia, PA. 16

Lan, Y. and Mezić, I. (2013). Linearization in the large of nonlinear systems and koopman operator spectrum. *Physica D: Nonlinear Phenomena*, 242(1):42–53. 14, 15, 34

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. (2020). Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752. 2, 28

Lee, J. (2011). Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9:415–424. 12

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning. 24

Lusch, B., Kutz, J. N., and Brunton, S. L. (2018). Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1). 3, 16, 32, 34

Mezic, I. and Runolfsson, T. (2004). Uncertainty analysis of complex dynamical systems. In *Proceedings of the 2004 American Control Conference*, volume 3, pages 2659–2664 vol.3. 3

Miranda, F. (2015). *Control Theory: Perspectives, Applications and Developments*. System science series. Nova Science Publishers, Incorporated. 11

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. 24

Morari, M. and H. Lee, J. (1999). Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4):667–682. 11, 12

Pan, S., Kaiser, E., de Silva, B. M., Kutz, J. N., and Brunton, S. L. (2023). Pykoopman: A python package for data-driven approximation of the koopman operator. 17

Proctor, J. L., Brunton, S. L., and Kutz, J. N. (2016a). Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161. 15, 16, 34

Proctor, J. L., Brunton, S. L., and Kutz, J. N. (2016b). Generalizing koopman theory to allow for inputs and control. 15, 34

Qin, S. J. and Badgwell, T. A. (1997). An overview of industrial model predictive control technology. In *AIche symposium series*, volume 93, pages 232–256. New York, NY: American Institute of Chemical Engineers, 1971-c2002. 12

Rahman, M. M. and Xue, Y. (2022). Robust policy optimization in deep reinforcement learning. 2, 28

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. 35

Schmid, P. J. (2010). Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28. 3, 16

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization. 2, 25

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation. 2

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. 2, 25

Scokaert, P. and Rawlings, J. (1998). Constrained linear quadratic regulation. *IEEE Transactions on Automatic Control*, 43(8):1163–1169. 12

Shi, H. and Meng, M. Q. H. (2022). Deep koopman operator with control for nonlinear systems. 3, 29

Song, L., Wang, J., and Xu, J. (2021). A data-efficient reinforcement learning method based on local koopman operators. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 515–520. 3, 29

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA. 19

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press. 1, 23

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. 46

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium. 46

Weissenbacher, M., Sinha, S., Garg, A., and Kawahara, Y. (2022). Koopman q-learning: Offline reinforcement learning via symmetries of dynamics. 3, 29

Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. (2015). A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346. 16, 34

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256. 23

Yeung, E., Kundu, S., and Hodas, N. (2017). Learning deep neural network representations for koopman operators of nonlinear dynamical systems. 3, 16, 32, 34

Yin, H., Welle, M. C., and Kragic, D. (2022). Embedding koopman optimal control in robot policy learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13392–13399. 3, 29

Appendix

A Implementation Details

This section provides detailed information on the key components of the Koopman-Inspired Proximal Policy Optimization (KIPPO) algorithm implementation. It covers the rollout and data collection process, the future state prediction mechanism, and the optimization procedure. These details complement the high-level description provided in Chapter 3 and offer insights into the practical aspects of implementing the KIPPO framework.

A.1 Rollouts and Data Collection

The rollout phase, detailed in Algorithm 1, involves the agent interacting with the environment for T steps, collecting essential data for training the KIPPO framework. This phase efficiently gathers data for both the Proximal Policy Optimization (PPO) algorithm and Koopman-inspired representation learning, enabling joint optimization of the policy and latent-space representation.

At each timestep t, the agent observes the current state x_t, which is then transformed into the latent representation y_t by the state encoder $\boldsymbol{\varphi}_x$. The policy π_θ, conditioned on y_t, samples an action u_t. Upon executing this action in the environment, a reward r_t, next state x_{t+1}, and termination flag d_t are obtained. The algorithm stores this information along with the action's log probability and estimated state value.

Several specialized data structures are utilized to manage the collected data efficiently. Trajectory data, comprising standard Reinforcement Learning (RL) elements (x_t, u_t, r_t, d_t,

$\log \pi_t$, V_t), are used for policy optimization and value function learning. Circular buffers ($\widehat{\mathbf{x}}$, $\widehat{\mathbf{u}}$, $\widehat{\mathbf{d}}$) maintain a sliding window of recent experiences, with a fixed size equal to the prediction horizon H. This design allows efficient memory management by automatically overwriting the oldest data when the buffer is full. Storage tensors ($\mathbf{x}$, $\mathbf{u}$, $\mathbf{m}$) accumulate sequences of states, actions, and binary masks from the recent history at each timestep. These tensors preserve the temporal structure of the collected data and are used during the optimization phase for computing prediction losses over variable-length sequences.

A binary mask $\mathbf{m}$ is employed to handle variable-length episodes. Computed using the cumulative product of inverted done flags, it effectively marks valid steps within each sequence. The mask maintains a value of 1 for all steps until encountering the first termination flag, after which it becomes 0, signaling the episode's end. This mechanism allows for managing sequences spanning multiple episodes and focusing optimization on relevant parts of the collected data.

When an episode concludes (d_t is True), the environment resets to its initial state. This process continues until the specified number of environment steps T is reached.

A.2 Future State Prediction

Algorithm 2 describes the future state prediction process, which takes as input an initial state x_0, and an action sequence $u_{0:H}$. It utilizes the state encoder $\boldsymbol{\varphi}_x$, action encoder $\boldsymbol{\varphi}_u$, and learned system matrices $\mathbf{K}$ and $\mathbf{B}$ to predict future states over the horizon H specified as a hyperparameter.

The prediction process begins by encoding the initial state x_0 into the latent-space using the state encoder $\boldsymbol{\varphi}_x$. This step maps the original state into the latent representation. The algorithm then iterates over the prediction horizon H, predicting the next latent state at each timestep h. The prediction is performed by applying the state-transition matrix $\mathbf{K}$ to the current latent state $\hat{y}_h$ and adding the effect of the control matrix $\mathbf{B}$ applied to the encoded action $\boldsymbol{\varphi}_u(u_h)$.

Algorithm 1 Rollouts Phase

Require:

Environment with state-space $\mathcal{S}$ and action space $\mathcal{A}$

Policy π_θ; Value function V^{π_θ}; State encoder $\boldsymbol{\varphi}_x$

Max steps (batch size) T; Prediction horizon H

Initialize storage for trajectory data $\{x_t, u_t, r_t, d_t, \log \pi_t, V_t\}_{t=0}^{T-1}$

Initialize temporary circular buffers

$\{\widehat{\mathbf{x}}_h\}_{h=0}^{H+1} \leftarrow 0$ ▷ Recent history of states, initialized to zeroes

$\{\widehat{\mathbf{u}}_h\}_{h=0}^{H} \leftarrow 0$ ▷ Recent history of actions, initialized to zeroes

$\{\widehat{\mathbf{d}}_h\}_{h=0}^{H} \leftarrow 1$ ▷ Recent history of done flags, initialized to ones

Initialize storage for the corresponding histories at each step $\{\mathbf{x}_t, \mathbf{u}_t, \mathbf{m}_t\}_{t=0}^{T-1}$

$x_0 \leftarrow$ env.reset() ▷ Initialize the environment and get initial state

Add x_0 to the circular buffer $\widehat{\mathbf{x}}$

for $t \leftarrow 0$ to T **do** ▷ Interact with the environment for T steps

  $y_t \leftarrow \boldsymbol{\varphi}_x(x_t)$ ▷ Encode the current state to obtain the latent representation

  $u_t \sim \pi_\theta(\cdot|y_t)$ ▷ Sample an action from the policy

  $\log \pi_t \leftarrow \log \pi_\theta(u_t|y_t)$ ▷ Compute the action's log probability

  $V_t \leftarrow V^{\pi_\theta}(y_t)$ ▷ Estimate the latent state value

  $x_{t+1}, r_t, d_{t+1} \leftarrow$ env.step(u_t) ▷ Execute the action in the environment

  Add x_{t+1}, u_t, d_{t+1} to the circular buffers $\widehat{\mathbf{x}}$, $\widehat{\mathbf{u}}$, and $\widehat{\mathbf{d}}$, respectively

  $\mathbf{x}_t, \mathbf{u}_t \leftarrow \widehat{\mathbf{x}}, \widehat{\mathbf{u}}$ ▷ Store recent history as prediction sequences

  $\mathbf{m}_t \leftarrow \prod_{h=0}^{H-1}(1 - \widehat{\mathbf{d}}_h)$ ▷ Store cumulative product of recent inverted done flags

end for

return All stored data

The future state prediction process yields a sequence of predicted latent states $\hat{y}_{1:H+1}$, representing the evolution of the system in the latent-space under the given action sequence. These predicted latent states can be further decoded back to the original state-space using the state decoder ${\varphi_x}^{-1}$ (not shown in the algorithm) to obtain the predicted states $\hat{x}_{1:H+1}$.

Algorithm 2 Future State Prediction for H Steps in Latent Space

Require:

State encoder φ_x; Action encoder φ_u; System matrices: $\mathbf{K}, \mathbf{B}$

Prediction horizon H; Initial state x_0; Action sequence $u_{0:H}$

$\hat{y}_0 \leftarrow \varphi_x(x_0)$ ▷ Encode the initial state

for $h \leftarrow 0$ to H **do** ▷ Predict H steps into the future

  $\hat{y}_{h+1} \leftarrow \mathbf{K}\hat{y}_h + \mathbf{B}\varphi_u(u_h)$ ▷ Predict next state in latent-space

end for

return $\hat{y}_{1:H+1}$

A.3 Optimization Process

The optimization phase updates all networks and trainable parameters using the data collected during the rollout phase. Algorithm 3 presents the detailed pseudocode for this phase, providing an overview of the optimization process and showcasing the integration of KIPPO's objectives with the PPO algorithm.

For each mini-batch, the algorithm computes the following losses:

- **Reconstruction loss ($\mathcal{L}_{\text{rec}}$):** The Mean Squared Error (MSE) between the original states and the reconstructed states obtained by encoding and decoding the states using the state autoencoder – Eq. (3.3).
- **Latent-space prediction loss ($\mathcal{L}_{\text{pred-ls}}$):** The MSE between the predicted latent states $\hat{y}_h$ and the encoded true states $\varphi_x(x_h)$, averaged over the prediction horizon and weighted by the binary mask – Eq. (3.4).

- **State-space prediction loss ($\mathcal{L}_{\text{pred-ss}}$):** The MSE between the predicted states $\hat{x}_h$ and the true states x_h, averaged over the prediction horizon and weighted by the binary mask – Eq. (3.6).

- **Total PPO loss ($\mathcal{L}_{\text{PPO}}$):** Computed using the encoded states $\varphi_x(x)$ and other relevant information from the mini-batch, such as actions, rewards, and value estimates.

The actor and critic networks utilize the latent-space representation during optimization. These networks operate on the encoded states, a basis for learning the policy and value function in the simplified latent-space. The use of latent representations instead of the original states is a modification to the PPO algorithm, enabling the algorithm to leverage the benefits of the latent-space while maintaining its core architecture and objectives. The $\not\nabla$ operator in the pseudocode denotes the detachment of the PPO loss gradients from the computation graph of the auxiliary components, ensuring that the state representations are optimized independently.

The algorithm performs future state prediction and loss computation using vector operations across the entire mini-batch and sequence length dimensions to efficiently compute the latent-space and state-space prediction losses. Although the pseudocode in Algorithm 3 presents the process as a for loop over each sequence for clarity, these operations are performed in a vectorized manner in practice.

The weighted sum of the representation losses $\mathcal{L}_{\text{KI}}$ (Eq. (3.7)) provides a means to balance the different objectives. The total loss $\mathcal{L}_{\text{KIPPO}}$ is then calculated as the sum of the weighted losses $\mathcal{L}_{\text{KI}}$ and the total PPO loss $\mathcal{L}_{\text{PPO}}$, combining the additional objectives with those of the PPO algorithm.

Finally, the gradients of the total loss with respect to the model parameters are computed using automatic differentiation. The gradients are then used to update the parameters using the Adam optimizer via backpropagation. This process is repeated for each mini-batch and the specified number of epochs, allowing KIPPO to iteratively refine the latent-space representation, the policy, and the value function based on the collected experience.

Algorithm 3 Optimization Phase

Require:

Policy π_θ; Value function V^{π_θ}; Update epochs N_{epochs}, Prediction horizon H

State encoder $\boldsymbol{\varphi}_x$; State decoder ${\boldsymbol{\varphi}_x}^{-1}$; Action encoder $\boldsymbol{\varphi}_u$; System matrices $\mathbf{K}, \mathbf{B}$

Collected data $\{x_t, u_t, r_t, d_t, \log \pi_t, V_t, \mathbf{x}_t, \mathbf{u}_t, \mathbf{m}_t\}_{t=0}^{T-1}$ from rollouts (Algorithm 1)

Hyperparameters $\omega_2, \omega_3, \omega_1$ ▷ PPO-specific hyperparameters omitted for brevity

for epoch $\leftarrow 1$ to N_{epochs} **do**
  Randomly shuffle *data* and split into minibatches
  for each $\mathcal{B} \in$ *data* **do**
    Initialize $\boldsymbol{\mathcal{L}}_{\text{pred-ls}}, \boldsymbol{\mathcal{L}}_{\text{pred-ss}} \leftarrow 0$
    for each $\{\mathbf{x}_t, \mathbf{u}_t, \mathbf{m}_t\} \in \mathcal{B}$ **do**
      $x_0 \in \mathbf{x}_t$ ▷ Initial state for prediction
      $u_{0:H} \in \mathbf{u}_t$ ▷ Actions to take for H steps
      $\hat{y}_{1:H+1} \leftarrow \text{LatentSpacePrediction}(x_0, u_{0:H})$ ▷ H-step prediction (Algorithm 2)
      $x_{1:H+1} \in \mathbf{x}_t$ ▷ Target state sequence
      $m_{1:H+1} \in \mathbf{m}_t$ ▷ Binary mask accounts for episode boundaries
      $\boldsymbol{\mathcal{L}}_{\text{pred-ls}} \leftarrow \boldsymbol{\mathcal{L}}_{\text{pred-ls}} + \frac{1}{H}\sum_{h=1}^{H} m_h(\hat{y}_h - \boldsymbol{\varphi}_x(x_h))^2$ ▷ Latent-space pred. loss
      $\boldsymbol{\mathcal{L}}_{\text{pred-ss}} \leftarrow \boldsymbol{\mathcal{L}}_{\text{pred-ss}} + \frac{1}{H}\sum_{h=1}^{H} m_h({\boldsymbol{\varphi}_x}^{-1}(\hat{x}_h) - x_h)^2$ ▷ State-space pred. loss
    end for
    $\boldsymbol{\mathcal{L}}_{\text{pred-ls}} \leftarrow \frac{1}{|\mathcal{B}|}\boldsymbol{\mathcal{L}}_{\text{pred-ls}}$
    $\boldsymbol{\mathcal{L}}_{\text{pred-ss}} \leftarrow \frac{1}{|\mathcal{B}|}\boldsymbol{\mathcal{L}}_{\text{pred-ss}}$
    $\boldsymbol{\mathcal{L}}_{\text{rec}} \leftarrow \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}}({\boldsymbol{\varphi}_x}^{-1}(\boldsymbol{\varphi}_x(x)) - x)^2$ ▷ Reconstruction loss
    $\boldsymbol{\mathcal{L}}_{\text{KI}} \leftarrow \omega_1\boldsymbol{\mathcal{L}}_{\text{rec}} + \omega_2\boldsymbol{\mathcal{L}}_{\text{pred-ls}} + \omega_3\boldsymbol{\mathcal{L}}_{\text{pred-ss}}$ ▷ Weighted sum of loss components
    $\boldsymbol{\mathcal{L}}_{\text{PPO}} \leftarrow \text{ComputePPOLoss}(\{\boldsymbol{\varphi}_x^{\not\propto}(x), u, ...\}_{\mathcal{B}})$ ▷ As per original implementation
    $\boldsymbol{\mathcal{L}}_{\text{KIPPO}} \leftarrow \boldsymbol{\mathcal{L}}_{\text{KI}} + \boldsymbol{\mathcal{L}}_{\text{PPO}}$ ▷ Total loss
    Compute gradients of $\boldsymbol{\mathcal{L}}_{\text{KIPPO}}$ with respect to model parameters
    Update model parameters with gradient descent
  end for
end for

B Glossary

A advantage function, quantifying the relative advantage of taking a specific action u in a given state x compared to the average action under the current policy π. 24–26

G return, the discounted cumulative reward obtained by the agent from a given state onwards in a specific trajectory. 19, 50

H prediction horizon, a hyperparameter specifying the number of timesteps over which the latent dynamics $\mathbf{K}$ and $\mathbf{B}$ are unrolled to generate predicted states $\hat{x}$. x–xii, 35, 38, 39, 50, 60, 67, 69, 76, 77, 79, 81, 94–96, 98, 101, 102

J expected return, the average return obtained by the agent over all possible trajectories when following a policy π. 20, 23

$N_{\mathbf{epochs}}$ number of training epochs, i.e., the number of times the learning algorithm iterates through the entire training dataset. 53, 98

P probability measure, assigning a value between 0 and 1 to events in a sample space, satisfying non-negativity, normalization, and countable additivity. 19, 20

Q action-value function, estimating the expected return starting from a given state x, taking a specific action u, and following a specific policy π thereafter. 20, 24

R reward function, mapping states, actions, or state-action pairs to scalar rewards. 19, 20, 23, 104

T total number of timesteps or interaction steps between the agent and the environment, either during a single episode (for episodic tasks) or over the entire training process (for continuing tasks). 19, 23, 53, 93–95, 98, 102

V state-value function, estimating the expected return starting from a given state x and following a specific policy π thereafter. 20, 24, 94, 95, 98

α learning rate, a hyperparameter scaling the magnitude of gradient updates applied to model parameters during optimization. 23, 53

$\mathcal{K}$ Koopman operator, an infinite-dimensional linear operator describing the evolution of observables g of a dynamical system. 14, 102, 103

$\mathcal{L}$ loss function in machine learning or reinforcement learning, quantifying the discrepancy between predicted and target values, serving as an objective to be minimized during training. 25, 37–40, 43, 57, 60, 63, 64, 96–98, 100, 101

φ_u action encoder, an MLP with tanh activation functions mapping actions u from the original action space $\mathcal{A}$ to the latent action space. ix, 32, 33, 35, 94, 96, 98

φ_x state encoder, an MLP with tanh activation functions mapping states x from the original state-space $\mathcal{S}$ to the latent-space. ix, 32, 33, 35, 37–39, 43, 93–98, 101

$\not\nabla$ notation indicating that gradient computation and backpropagation are disabled for a specific operation or component. 97, 98

ϵ clipping parameter in PPO, limiting the ratio of the new policy to the old policy, ensuring conservative and stable policy updates. 25, 26, 53

γ discount factor, a value between 0 and 1 determining the present value of future rewards in the return calculation. 19, 20, 53

ω_1 hyperparameter for the weight of the reconstruction loss $\mathcal{L}_{\text{rec}}$. ix–xi, 40, 60, 64–66, 70–76, 98, 101

ω_2 hyperparameter for the weight of the latent-space prediction loss $\mathcal{L}_{\text{pred-ls}}$. ix–xi, 40, 59, 60, 64–66, 70–75, 77, 98, 101

ω_3 hyperparameter for weight of the state-space prediction loss $\mathcal{L}_{\text{pred-ss}}$. x, xi, 40, 59, 60, 64, 65, 68, 70, 72, 73, 75–77, 98, 101

$\mathcal{L}_{\mathbf{KIPPO}}$ total loss for the KIPPO framework, computed as the sum of the weighted Koopman-inspired loss $\mathcal{L}_{\text{KI}}$ and the standard PPO loss $\mathcal{L}_{\text{PPO}}$, combining the objectives of representation learning and policy optimization. 43, 97, 98

$\mathcal{L}_{\mathbf{KI}}$ weighted sum of the reconstruction loss $\mathcal{L}_{\text{rec}}$, latent-space prediction loss $\mathcal{L}_{\text{pred-ls}}$, and state-space prediction loss $\mathcal{L}_{\text{pred-ss}}$, where the weights ω_1, ω_2, and ω_3 control the relative importance of each term. 40, 43, 97, 98, 101

$\mathcal{L}_{\mathbf{PPO}}$ total loss function used in the PPO algorithm. 43, 97, 98, 101

$\mathcal{L}_{\mathbf{pred\text{-}ls}}$ latent-space prediction loss, the MSE between the predicted latent states $\hat{y}$ and the encoded target states $\varphi_x(x)$ over the prediction horizon H. 38, 40, 57, 60, 63, 64, 96, 98, 100, 101

$\mathcal{L}_{\mathbf{pred\text{-}ss}}$ state-space prediction loss, the MSE between the decoded predicted states ${\varphi_x}^{-1}(\hat{y})$ and the true target states x over the prediction horizon H. 39, 40, 57, 60, 63, 64, 97, 98, 100, 101

$\mathcal{L}_{\mathbf{rec}}$ reconstruction loss, the MSE between the original states x and their reconstructions $\tilde{x}$ obtained by encoding and decoding the states using the state autoencoder. 37, 40, 57, 60, 63, 64, 96, 98, 100, 101

${\varphi_x}^{-1}$ state decoder, an MLP with tanh activation functions mapping latent states y back to the original state-space $\mathcal{S}$. ix, 32, 33, 35, 37, 39, 96, 98, 101

$\hat{y}$ predicted state observable, the latent representation of a predicted future state $\hat{x}$, obtained by applying the learned system matrices $\mathbf{K}$ and $\mathbf{B}$ to the current latent state and action. 35, 38, 39, 94, 96, 98, 101

$\hat{x}$ sequence of predicted future states, obtained by decoding the predicted state observables $\hat{y}$ back to the original state-space $\mathcal{S}$ using the state decoder ${\varphi_x}^{-1}$. 35, 50, 51, 96, 98, 99, 101

λ an eigenvalue of the Koopman operator $\boldsymbol{\mathcal{K}}$, a scalar value λ associated with an eigenfunction ϕ. 14, 15, 102

$\mathbb{E}$ expectation operator, computing the average value of a random variable. 20, 23

$\mathbb{R}$ set of real numbers, including all rational and irrational numbers. 12, 14, 15

$\mathbf{B}$ control matrix representing the effect of control inputs in the finite-dimensional approximation of the Koopman operator, capturing how actions v influence the evolution of observables y. ix, 15, 32–35, 38, 39, 94, 96, 98, 99, 101

$\mathbf{C}$ system matrix in a linear dynamical system, capturing the linear dynamics of the system states without control inputs. 10, 12

$\mathbf{D}$ control matrix in a linear dynamical system, representing the effect of control inputs on the system states. 12

$\mathbf{F}$ state transition function, mapping the current state x_t and action u_t to the next state x_{t+1} in a discrete-time dynamical system. 11, 12, 14

$\mathbf{K}$ Koopman state-transition matrix, a finite-dimensional approximation of the Koopman operator $\boldsymbol{\mathcal{K}}$ for autonomous systems, capturing the linear dynamics of the observables y without control inputs. ix, 15, 32–35, 38, 39, 94, 96, 98, 99, 101

$\mathbf{u}$ storage tensor of shape (T, H, *), used to store at each timestep the past H actions from $\overset{\frown}{\mathbf{u}}$; used as inputs for computing prediction losses during optimization. 94, 95, 98

$\mathbf{m}$ storage tensor of shape (T, H+1), used to store the binary masks m associated with the collected state sequences $\mathbf{x}$, indicating the validity of each state in the sequence. 94, 95, 98

$\mathbf{x}$ storage tensor of shape (T, H+1, *), used to store at each timestep the past H+1 states from $\overset{\frown}{\mathbf{x}}$; used as targets for computing prediction losses during optimization. 94, 95, 98, 102

$\mathcal{A}$ action space, the set of all possible actions that an agent can take in the environment. 15, 18, 95, 100, 104

$\mathcal{B}$ mini-batch, a subset of training data used for a single iteration of gradient descent during optimization. 98

$\mathcal{S}$ state-space, the set of all possible states that the environment can be in. 14, 15, 18, 95, 100, 101, 104, 105

∇ gradient operator, computing the vector of partial derivatives of a scalar-valued function. 23

ω loss weight, a hyperparameter balancing the contributions of different loss components in multi-objective optimization. ix–xi, 40, 59, 60, 64–66, 68, 70–77, 98, 100, 101

$\overset{\curvearrowright}{\mathbf{u}}$ circular buffer used to store the most recent actions during rollouts, providing a sliding window of the agent's decisions for constructing prediction inputs, allowing efficient storage and retrieval of recent actions without the need for frequent memory reallocation. 94, 95, 102

$\overset{\curvearrowright}{\mathbf{d}}$ circular buffer used to store the most recent termination flags d during rollouts, used to compute the binary masks m for handling variable-length trajectories, allowing efficient storage and retrieval of recent termination flags without the need for frequent memory reallocation. 94, 95

$\overset{\curvearrowright}{\mathbf{x}}$ circular buffer used to store the most recent states during rollouts, providing a sliding window of the agent's experience for constructing prediction targets, allowing efficient storage and retrieval of recent states without the need for frequent memory reallocation. 94, 95, 102

ϕ an eigenfunction of the Koopman operator $\mathcal{K}$, a function ϕ that, when acted upon by the Koopman operator, yields a scalar multiple of itself. 102, 103

π policy, either deterministic or stochastic, mapping states x to actions u or probability distributions over actions, representing the agent's decision-making strategy. 19–21, 23–25, 93–95, 98, 99

$\prod$ cumulative product operator, computing the product of a sequence of elements: $\prod_{i=1}^{n} a_i = a_1 \times a_2 \times \cdots \times a_n$. 95

ρ_0 initial state distribution, a probability distribution over the state-space $\mathcal{S}$ specifying the likelihood of the environment starting in each possible state. 19

τ trajectory or rollout, a sequence of states, actions, and rewards generated by the agent's interaction with the environment over a finite number of timesteps. 19, 20, 23, 104

θ trainable model parameters in machine learning or reinforcement learning, typically including weights and biases of neural networks, adjusted during learning to minimize the objective function. 23, 25, 26, 93, 95, 98

d binary flag indicating whether the current episode or rollout has terminated. 93–95, 98, 103

f action observable function, mapping actions u from the original action space $\mathcal{A}$ to the lifted action observable space. 15, 35, 105

g state observable function, mapping states x from the original state-space $\mathcal{S}$ to the lifted observable space: $y = g(x)$. 14, 15, 32, 35, 100, 104, 105

m binary mask that indicates the validity of states in a trajectory τ, used to handle variable-length trajectories, with $m_t = 1$ denoting a valid state and $m_t = 0$ denoting an invalid or terminal state. 38, 39, 94, 95, 98, 102–104

r reward, a scalar value provided by the environment to the agent based on the current state and action, computed using the reward function R. 19, 93, 95, 98

u an action, an element of the action space $\mathcal{A}$, representing a decision made by the agent at a given timestep. ix, 12, 15, 18–21, 23–25, 32, 33, 35, 93–96, 98–100, 102–105

v an action observable, the result of applying the action observable function f to an action u. 15, 32, 102

x state, an element of the state-space $\mathcal{S}$, representing a complete description of the environment at a given timestep. ix, 10–12, 14, 15, 18–21, 23–25, 32, 33, 35, 37–39, 43, 50, 51, 93–105

y state observable, the result of applying the state observable function g to a state x. ix, 15, 32, 33, 35, 38, 39, 93–96, 98, 101, 102, 104

C Acronyms

A2C Advantage Actor-Critic. 24, 85

CTE Cumulative Temporal Error. 50, 51, 55–57, 61–63, 80

DDPG Deep Deterministic Policy Gradient. 24, 85

DKRL Deep Koopman Reinforcement Learning. 3, 29

DMD Dynamic Mode Decomposition. 3, 16, 17

DMDc Dynamic Mode Decomposition with Control. 16

DPG Deterministic Policy Gradient. 24

EWMA Exponentially Weighted Moving Average. 50, 51, 55, 57, 61–63, 80

GMM Gaussian Mixture Model. 3

KFC Koopman Forward (Conservative) Q-Learning. 3, 29

KIPPO Koopman-Inspired Proximal Policy Optimization. iv, 3, 4, 31, 32, 36–38, 40–42, 44–47, 49, 51, 52, 55, 56, 58, 59, 65, 67, 70, 80, 82–86, 93, 96, 97

KL Kullback-Leibler. 25

LQR Linear Quadratic Regulator. 3, 12, 13, 29

LTI Linear Time-Invariant. 9, 10, 15, 34

MBPO Model-Based Policy Optimization. 2, 28

MDP Markov Decision Process. 18

MLP Multi-Layer Perceptron. 34

MPC Model Predictive Control. 3, 12, 13, 29

MSE Mean Squared Error. 96, 97

PID Proportional-Integral-Derivative. 12, 13

PPO Proximal Policy Optimization. iv, 2–5, 25, 26, 28, 31, 41–45, 49, 51, 52, 55, 57, 80, 83–85, 93, 96, 97

RL Reinforcement Learning. iv, 1, 3, 7, 8, 18, 20, 22, 26–29, 31, 49, 84–86, 93

RPO Robust Policy Optimization. 2, 28, 29, 84

SAC Soft Actor-Critic. 24, 85

SLAC Stochastic Latent Actor-Critic. 2, 28

TD3 Twin Delayed DDPG. 24, 85

TRPO Trust Region Policy Optimization. 2, 25, 26, 85

Vita

Andrei Cozma was born in Haţeg, Romania, on June 25, 1999. At 13, he relocated to the United States, embarking on a journey rich with diverse cultural experiences and opportunities for growth.

In 2018, Andrei began his engineering career at The University of Tennessee, Knoxville. Throughout his undergraduate years, he actively pursued expertise through rigorous coursework, research initiatives, industry internships, personal projects, and dedicated self-study. In May 2022, Andrei graduated summa cum laude with a Bachelor of Science in Computer Science, complemented by minors in Cybersecurity and Business Administration.

Immediately following, Andrei commenced his graduate studies at the same institution. His research interests span the cutting-edge fields of Deep Learning, Reinforcement Learning, Signal and Image Processing, and Computer Vision. During this time, he has engaged in innovative research projects, collaborating with faculty and peers to advance intelligent systems and machine learning.

Andrei's academic journey reflects a strong commitment to both theoretical foundations and practical applications. His thesis work synthesizes rigorous research with real-world applications, mirroring his diverse background and experiences. Through his pursuits, Andrei aims to contribute meaningful advancements to computer science, developing innovative solutions to complex computational challenges.