RIO

PUC

Dissertação de Mestrado

Meta Reinforcement Learning Applied on Quadrupedal Robots for Blind Locomotion and Fast Adaptation on Unknown Terrains

Pedro Leon Fontes Cardoso Bazan

Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Engenharia Mecânica
Rio de Janeiro, 10 de Outubro de 2025

Pontifícia
Universidade
Católica do
Rio de Janeiro

Dissertação de mestrado

Meta Reinforcement Learning Applied on Quadrupedal Robots for Blind Locomotion and Fast Adaptation on Unknown Terrains

Pedro Leon Fontes Cardoso Bazan

Orientação: Prof. Marco Antonio Meggiolaro
Coorientação: Prof. Wouter Caarls
Coorientação: Profa. Vivian Suzano Medeiros

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Engenharia Mecânica pelo programa de Pós-Graduação em Engenharia Mecânica, no Departamento de Engenharia Mecânica.

Rio de Janeiro, 10 de Outubro de 2025

Pontifícia
Universidade
Católica do
Rio de Janeiro

Meta Reinforcement Learning Applied on Quadrupedal Robots for Blind Locomotion and Fast Adaptation on Unknown Terrains

Pedro Leon Fontes Cardoso Bazan

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Engenharia Mecânica. Aprovada pela Comissão examinadora abaixo:

Prof. Marco Antonio Meggiolaro
Orientador
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Wouter Caarls
Coorientador
Departamento de Engenharia Elétrica – PUC-Rio

Profa. Vivian Suzano Medeiros
Coorientadora
Universidade de São Paulo – USP

Prof. Marcelo Becker
Universidade de São Paulo – USP

Dr. João Carlos Virgolino Soares
Istituto Italiano di Tecnologia – IIT

Prof. Allan Nogueira de Albuquerque
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

Rio de Janeiro, 10 de Outubro de 2025

Pontifícia
Universidade
Católica do
Rio de Janeiro

Pedro Leon Fontes Cardoso Bazan
Graduado em Engenharia de Controle e Automação pela Pontifícia Universidade Católica do Rio de Janeiro em 2023.

Bibliographic data

Bazan, Pedro Leon Fontes Cardoso

Meta reinforcement learning applied on quadrupedal robots for blind locomotion and fast adaptation on unknown terrains / Pedro Leon Fontes Cardoso Bazan ; advisor: Marco Antonio Meggiolaro ; co-advisors: Wouter Caarls, Vivian Suzano Medeiros. – 2025.

91 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Mecânica, 2025.

Inclui bibliografia

1. Engenharia Mecânica – Teses. 2. Quadrúpede. 3. Meta aprendizado por reforço. 4. Adaptação rápida. 5. Locomoção cega. I. Meggiolaro, Marco Antonio. II. Caarls, Wouter. III. Medeiros, Vivian Suzano. IV. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Mecânica. V. Título.

To my family and friends, for all the support and encouragement.

Acknowledgments

To my family, my deepest gratitude. You have always supported me and stood by my side through both the sad and the happiest moments.

To my siblings, Boris and Natali, for always being there to support me and help me whenever I needed, and to my parents, Boris and Deolinda, who raised us with love, care, and encouragement to pursue education, having guided (so far) three masters and three future PhDs. This path was first paved by my mother, who earned her PhD in Electrical Engineering, in the field of Telecommunications, at PUC-Rio.

To my girlfriend, Luísa Ramoa, who supported me during difficult times and has been my anchor over the past two years, helping me achieve this great accomplishment. Thank you so much—I already consider you part of the family.

To my dear friend Diana Moreira, who has brought so many good things into my life, introduced me to my girlfriend, and send me the opportunity to join the LabRob.

To my advisors and co-advisors, Marco Meggiolaro, Vivian Suzano Medeiros, and Wouter Caarls, who guided me throughout this journey. A special thanks to Prof. Wouter, who assisted in the overall development of the algorithm and provided access to the computers at the Intelligent Control Lab (LCI).

To LabRob and all its members, where I completed my undergraduate research, TCC, master's, and am now pursuing my PhD.

To PUC-Rio and all the professors who have accompanied me throughout this journey.

To my great friends Artur Schiavo, Leonardo Migueis, Victor Chamusca, Clara Lúcio, Marcelo Campanelli, Pedro Guedes, Leonardo Trote, Luigi Gosling, João Guilherme Benevenuto, Igor Minichetti Butô, Thiago Marsiglia, and Wilson Junior—friends with whom I have shared my best moments.

To my college friends from PUC-Rio, for all the memories and support.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior- Brasil (CAPES)- Finance Code 001.

This work was also funded by the Brazilian National Research Council (CNPq).

Abstract

Fontes Cardoso Bazan, Pedro Leon; Meggiolaro, Marco Antonio (Advisor); Caarls, Wouter (Co-Advisor); Medeiros, Vivian Suzano (Co-Advisor). **Meta Reinforcement Learning Applied to Quadrupedal Robots for Blind Locomotion and Fast Adaptation on Unknown Terrains**. Rio de Janeiro, 2025. 91p. Master's Dissertation – Department of Mechanical engineer, Pontifícia Universidade Católica do Rio de Janeiro.

Blind locomotion refers to the challenge of navigating varied terrains without prior knowledge or exteroceptive data. Although quadruped robots often use external sensors, these can be unreliable in low-light or resource-constrained settings and cannot anticipate disturbances such as slippage. In such scenarios, quadrupeds must rely exclusively on proprioceptive feedback, using internal measurements — joint positions, velocities, and contact forces — to adapt their locomotion strategies. While slip detection and terrain-estimation methods exist, leveraging proprioceptive information offers advantages across many applications, partly because exteroceptive sensors generally operate at lower acquisition frequencies than proprioceptive sensors. This work explores Meta-Reinforcement Learning (Meta-RL) to enhance policy robustness and rapid adaptation for quadruped robots during blind locomotion on challenging terrain, with the goal of achieving zero-shot generalization — i.e., enabling the agent to perform effectively in unseen environments without additional training. It builds on the RL^2 algorithm, integrating recurrent neural networks into Proximal Policy Optimization (PPO) to implicitly encode task-specific information from experience. Two novel RL^2-based architectures are proposed and evaluated in simulation with the ANYmal C quadruped robot across diverse terrain conditions, focusing on flat surfaces with stochastic slip and highly unstructured terrains. Results show that recurrent policies significantly outperform standard PPO, improving both adaptability and robustness under unpredictable ground dynamics and thereby advancing the state of blind quadrupedal locomotion in challenging simulated environments, with implications for real-world deployment.

Keywords

Quadrupedal Robots; Meta-Reinforcement Learning; Blind Locomotion; Fast Adaptation; Unknown Terrains.

Resumo

Fontes Cardoso Bazan, Pedro Leon; Meggiolaro, Marco Antonio; Caarls, Wouter; Medeiros, Vivian Suzano. **Meta Aprendizado por Reforço Aplicado a Robôs Quadrupedes para Locomoção Cega e Rápida Adaptação em Terrenos Desconhecidos**. Rio de Janeiro, 2025. 91p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

A locomoção às cegas refere-se ao desafio de navegar por terrenos variados sem conhecimento prévio ou dados exteroceptivos. Embora robôs quadrúpedes frequentemente utilizem sensores externos, esses podem ser pouco confiáveis em condições de baixa luminosidade ou de recursos limitados, não conseguindo antecipar perturbações como escorregamento. Nesses cenários, quadrúpedes devem confiar exclusivamente em feedback proprioceptivo, utilizando medições internas — posições articulares, velocidades e forças de contato — para adaptar suas estratégias de locomoção. Embora existam métodos de detecção de escorregamento e de estimativa da topografia do terreno, o uso de informações proprioceptivas oferece vantagens em diversas aplicações, em parte porque sensores exteroceptivos geralmente operam em frequências de aquisição mais baixas do que sensores proprioceptivos. Este trabalho investiga o Meta-Aprendizado por Reforço (Meta-RL) para aumentar a robustez das políticas e a adaptação rápida de robôs quadrúpedes durante a locomoção às cegas em terrenos desafiadores, com o objetivo de alcançar generalização zero-shot — isto é, permitir que o agente atue de forma eficaz em ambientes não vistos sem treinamento adicional. O algoritmo RL^2 é tomado como base, integrando redes neurais recorrentes ao Proximal Policy Optimization (PPO) para codificar implicitamente, a partir da experiência, informações específicas de cada tarefa. Duas novas arquiteturas baseadas em RL^2 são propostas e avaliadas em simulação com o robô quadrúpede ANYmal C em diferentes condições de terreno, com foco em superfícies planas com escorregamento estocástico e em terrenos altamente não estruturados. Os resultados mostram que políticas recorrentes superam significativamente o PPO padrão, aumentando tanto a adaptabilidade quanto a robustez sob dinâmicas imprevisíveis do solo e, assim, avançando o estado da locomoção quadrúpede às cegas em ambientes simulados desafiadores, com implicações para aplicação no mundo real.

Palavras-chave

Robôs Quadrúpedes; Meta Aprendizado por Reforço; Locomoção Cega; Rápida Adaptação; Terrenos Desconhecidos.

Table of contents

List of figures

List of tables

List of algorithms

List of Abreviations

GRF – Ground Reaction Forces

DoF – Degrees of Freedom

CoM – Center of Mass

MPC – Model Predict Control

WBC – Whole Body Control

IK – Inverse Kinematics

MDP – Markov Decision Process

RL – Reinforcement Learning

PODMP – Partial Observable Markov Decision Process

MLP – Multilayer Perceptron

RNN – Recurrent Neural Network

GRU – Gated Recurrent Unit

FI – Friction Index

The limitation of robotics lies in its understanding of the human soul. Simply because robotics excels at measuring and quantifying everything, while the human soul cannot be measured — only felt.

Luiz Tardelli, .

1
Introduction

Thanks to their articulated morphology and dynamic leg coordination, quadruped robots demonstrate superior stability and balance compared to bipedal designs [1], greater terrain adaptability than wheeled platforms [2], and enhanced ability to traverse complex obstacles. These advantages make them uniquely suited for real-world applications in challenging environments.

However, quadruped locomotion involves a high number of degrees of freedom, floating-base dynamics, and interaction with variable terrains (e.g., slippage, uneven surfaces), which demand advanced control strategies and accurate system modeling [3]. For this reason, Reinforcement Learning (RL) [4] has emerged as a promising approach, enabling robots to learn locomotion policies directly through interaction with the environment, without the need for an explicit analytical model. This is particularly advantageous in dynamic, stochastic, and hard-to-model settings.

To support the development of such adaptive locomotion strategies, robots also rely on exteroceptive perception to gather information about the environment. This is typically emulated using sensors [5] which provide terrain information or exteroceptive information to detect slippage events [6]. With such sensory data, robots can build internal representations of the terrain and learn more precise and adaptive locomotion strategies, enabling them to navigate obstacles like stairs, slopes, and random rough terrains.

However, in the absence of such sensory information, most RL-based policies tend to learn more generic behaviors—sufficient to handle a variety of terrains, but may lack precision and adaptability in novel conditions, as shown in 4.2. Among these challenges, slip terrains deserve particular attention. Even in seemingly simple environments, unpredictable variations in ground friction can destabilize the robot and severely degrade performance. Unlike geometric irregularities, slip is harder to detect and compensate for because it is not directly observable through onboard proprioceptive sensors. This makes the problem of blind locomotion on slippery terrains especially relevant, as it directly tests the controller's ability to adapt to changes in ground dynamics without relying on external sensors or explicit friction estimation [6].

While standard reinforcement learning methods [7, 8] have shown strong performance in handling slippage and irregular terrains, their specialization for a specific task limits their generalization. This often renders a policy trained for one task ineffective on another, necessitating extensive retraining for

adaptation to new environments, a key limitation demonstrated in the Results section (4). To improve sample efficiency and enable faster generalization across different terrains, this project adopts a Meta-RL framework [9], which extends traditional RL by incorporating the ability to "learn how to learn". Instead of training a policy for a single task, Meta-RL extracts patterns across multiple tasks during training, enabling rapid adaptation to new environments by leveraging prior experience.

Specifically, we adopt the Fast Reinforcement Learning via Slow Reinforcement Learning (RL^2) paradigm [10], where a recurrent neural network (RNN) processes entire episodes to capture temporal dependencies and implicitly encode task-specific information. We build on this framework using Proximal Policy Optimization (PPO) [11] as the base algorithm, and propose two novel RL^2-inspired architectures for quadrupedal locomotion.

These approaches are evaluated against standard PPO—a well-established baseline in legged robot control—and validated in simulation across a broad set of scenarios. The evaluation includes flat terrains with different levels of slippage as well as unstructured environments containing irregular obstacles, slopes, and box-shaped terrain. The emphasis is placed on flat slip terrains, while complementary experiments on irregular surfaces serve to assess generalization beyond the training distribution. The results demonstrate that recurrent policies outperform standard PPO, offering enhanced adaptability and robustness in the face of unpredictable ground dynamics.

1.1 Related Work

Over the years, extensive research has focused on advancing quadrupedal robot locomotion by improving stability, efficiency, and adaptability across diverse terrains and dynamic conditions. A key aspect of these advancements lies in refining locomotion control strategies while developing novel approaches to enhance environmental awareness. By integrating real-time terrain perception and adaptive decision-making, robots can better understand their surroundings, adjust their gait accordingly, and navigate complex environments autonomously.

1.1.1 Control Strategies Based on Inverse Kinematics and Dynamics

Controlling a quadrupedal robot poses significant challenges primarily due to its complex morphology. Maintaining balance is particularly demanding, as the robot's center of mass (CoM) is elevated above the ground, increasing the

risk of instability during motion. Moreover, the high number of DoF inherent to its structure adds substantial complexity to the control problem, requiring advanced strategies for coordinated actuation, dynamic stability, and real-time adaptation to uneven terrains or external disturbances.

One common approach used for addressing these challenges is kinematics-based control. Csillag et al. [12] explain that inverse kinematics (IK) can be employed to generate predefined foot trajectories based on the robot's geometric model, ensuring that the feet follow desired paths while satisfying kinematic constraints. However, solving the IK problem can be computationally expensive and numerically unstable, especially in complex robotic systems. To mitigate these difficulties, Csillag et al. [12] propose the use of feedforward artificial neural networks, such as Radial Basis Function (RBF) networks and Multilayer Perceptrons (MLPs), to approximate the IK mapping. These models are capable of learning the relationship between end-effector positions and joint configurations, enabling faster and more robust trajectory generation.

Alternatively, Hidayat et al. [13] present a simpler, model-free strategy to bypass IK computations altogether. They employ predefined sinusoidal patterns to model the end-effector trajectories, effectively generating foot motions through wave-like curves. This method reduces the reliance on geometric modeling and simplifies the control architecture, making it suitable for low-cost or resource-constrained robotic platforms.

Although kinematics-based approaches offer a simplified framework for trajectory generation, they often neglect the dynamic interactions between the robot and its environment. To overcome this limitation, several studies have investigated dynamics-based control strategies. Among these, Model Predictive Control (MPC) [14] has emerged as one of the most widely adopted methods in the literature due to its ability to generate dynamically consistent trajectories by optimizing control inputs over a prediction horizon.

Kang et al. [15] apply a nonlinear MPC strategy to simultaneously optimize the base trajectory and the sequence of stepping locations. In their approach, the optimal control problem is solved using a second-order numerical solver combined with a Sparse Gauss-Newton (SGN) method, enabling efficient computation and real-time feasibility.

Exploring a different perspective, Villarreal et al. [16] implement MPC on the HyQReal quadruped robot to achieve terrain-aware locomotion using solely on-board sensing and computation. Their framework integrates two key components: a convolutional neural network (CNN) that identifies safe foothold locations from terrain data, and an MPC module that plans and controls foot placements based on these predictions, ensuring adaptive and

robust movement across uneven environments.

An alternative approach to leverage the robot's full-body dynamics in quadrupedal locomotion is the Whole-Body Control (WBC) framework. This method coordinates all joints and limbs of the robot within a unified control structure, enabling the simultaneous execution of multiple motion tasks—such as body posture regulation, foot placement, and manipulation—while satisfying dynamic constraints including contact forces, joint torques, and friction limits.

Lu et al. [17] propose a control framework for quadruped robots traversing rough terrain by integrating WBC with Virtual Model Control (VMC), ensuring both precise trajectory tracking and compliant interaction. Their method combines a trajectory planner based on the normalized energy stability criterion with a probabilistic contact detection strategy for event-based leg state switching. Experiments with the SDUQuad-144 robot, which climbs high steps and stairs without prior environmental knowledge, highlight the robustness and adaptability of the approach.

Expanding on this strategy, Bellicoso et al. [18] employ WBC to execute dynamic gaits—such as trot, pace, and dynamic lateral walk—and enable smooth transitions between them. Their architecture couples a motion planner with a hierarchical WBC that optimizes motion and contact forces by solving a cascade of prioritized tasks. This hierarchy ensures that essential physical constraints—such as floating-base dynamics, torque limits, and contact stability—are enforced before pursuing lower-priority objectives, yielding robust and versatile locomotion.

Another common approach in quadrupedal locomotion control is the integration of Model Predictive Control (MPC) with Whole-Body Control (WBC) to leverage both high-level planning and low-level dynamic tracking. Amanzadeh et al. [19] proposed a hierarchical control framework that combines MPC with a gradient-descent-based adaptive updating law to enable robust locomotion under unknown payload conditions. In this approach, a high-level controller estimates the parameters of a reduced-order model, which informs the MPC for trajectory planning. These planned trajectories are then executed by a nonlinear WBC, allowing the system to adapt online to varying payloads and maintain stability even in uncertain and dynamic environments.

The work of Grandia et al. [20] proposes a complete perception, planning, and control pipeline for dynamic locomotion on rough terrain, optimizing the robot's full degrees of freedom in real-time. Their method combines steppability classification, plane segmentation, and signed distance fields with convex local constraints embedded in a model predictive controller. Optimiza-

tion is performed via multiple-shooting, real-time iteration, and filter-based line search, enabling high-frequency, reliable performance. The approach is validated in simulation and on the ANYmal quadruped, achieving robust locomotion over gaps, slopes, and stepping stones, setting a new state of the art in dynamic climbing.

1.1.2 Learning-Based Control methods

1.1.2.1 Reinforcement Learning Approaches

While kinematics- and dynamics-based methods have demonstrated strong performance under controlled conditions, they often rely on accurate models and detailed terrain estimation, which limits their robustness and adaptability when facing unstructured or unpredictable environments. To address these challenges, the field has increasingly shifted toward learning-based control methods, which allow robots to acquire locomotion policies directly from data or through interaction with the environment, thereby reducing reliance on explicit modeling.

On the learning approaches, Reinforcement Learning (RL) has emerged as a promising framework for quadrupedal locomotion due to its capacity for autonomous learning and adaptation in complex environments without the necessity of modeling the environment. Gurram et al. [21] present a comprehensive review of the state of the art in RL-based locomotion controllers, highlighting the versatility of these approaches in overcoming the limitations of traditional model-based methods. Their study covers a wide range of methodologies, including the design of learning algorithms, reward functions, training curricula, and Sim-to-Real transfer techniques, all of which are fundamental to enabling robust and adaptive locomotion on real robotic platforms.

In contrast, one of the drawbacks of the work of Gurram et al. [21], the authors implicitly assume that to achieving robust locomotion with reinforcement learning its necessarily requires the integration of exteroceptive signals—such as LiDAR or depth cameras—during both training and deployment. Although they argue that proprioceptive information alone may limit the learning process, this assumption introduces an important limitation: the resulting controllers become dependent on perception pipelines that are susceptible to noise, latency, occlusions, and outright failure in visually degraded or slippery environments.

In the work of Chen et al. [22], the authors explore a novel reinforcement

learning paradigm for quadrupedal locomotion that goes beyond the conventional position-based control framework. Traditionally, RL policies output low-frequency target joint positions, which are then converted into joint torques using high-frequency proportional-derivative (PD) controllers. Inspired by the shift in model-based control from position-based to torque-based strategies, this study proposes a torque-based RL framework in which the policy directly predicts joint torques at a high frequency, bypassing the need for PD controllers.

While the approach demonstrates promising results on flat and structured terrains, several limitations become evident when extending the discussion to more dynamic or unpredictable environments. Directly predicting torques at high frequency increases the controller's sensitivity to abrupt ground–robot interaction forces and contact inconsistencies, making it more difficult to recover from unexpected perturbations. Without the stabilizing contribution of PD controllers, the policy must implicitly learn compliance and recovery behaviors—skills that are difficult to acquire solely through training. Furthermore, the study does not examine performance under non-stationary dynamics or partial observability, leaving open the question of whether torque-based RL can maintain stability in rapidly changing conditions.

Lee et al. [23] proposed a novel approach to address the Sim-to-Real gap in quadrupedal robot locomotion by leveraging Constrained Markov Decision Processes (CMDPs). Recognizing that conventional reinforcement learning in simulation often overlooks critical physical constraints—such as joint torque limits, velocity boundaries, and safety requirements—the authors demonstrated how CMDPs provide a structured framework to incorporate these real-world limitations during policy training explicitly.

A major limitation of this approach is its restricted validation to flat or discretized environments, which casts doubt on its performance in complex, dynamically changing terrains where underlying constraints—such as allowable torques or velocities—must adapt rapidly to changing physical conditions. In particular, CMDPs rely on predefined and stationary constraint thresholds, which may be overly restrictive in real scenarios requiring aggressive corrective actions or momentary deviations from nominal safety bounds. This structural rigidity limits the controller's ability to adapt online when the terrain dynamics deviate significantly from the training distribution.

Building upon recent advances in learning-based locomotion control, Han and Zhao [24] present a hybrid control architecture that combines reinforcement learning with model-based control to address the challenge of high-speed movement on uneven terrain. The proposed framework decouples the control

loop into two distinct layers: a low-frequency RL policy that generates joint-level control commands and a high-frequency model-based controller that ensures precise execution. This novel approach maintains the adaptability of learning-based methods while leveraging the responsiveness of traditional control systems, effectively balancing computational efficiency with dynamic performance requirements.

Despite its results, the method still exhibits several limitations, many of which are acknowledged by the authors themselves. First, the framework employs linear feedback control laws, whose capacity to stabilize the robot under highly nonlinear ground interactions—such as foot slippage or partial footholds—may be limited. Second, the stability of the controller under sensor noise, actuator disturbances, or partial failures is not analyzed, even though such factors are common in real terrains with low friction or irregular contact. The authors also note that the stiffness and damping parameters of the joint controllers were selected empirically, leaving unexplored how variations in these parameters might affect stability and adaptability under rapidly changing terrain dynamics. Finally, the robustness of the controller under sensor or actuator faults remains untested, and its performance under noisy proprioceptive inputs—a critical element for blind or partially observable locomotion—was not examined.

1.1.2.2 Meta Reinforcement Learning Approaches

Conventional RL algorithms typically achieve strong performance when operating within environments that remain consistent with their training distribution. However, their ability to generalize and adapt quickly to entirely new situations is often limited. While effective in stochastic environments, where variations follow known probability distributions, they often struggle when those distributions change over time, such as during abrupt terrain changes or unexpected slippage, which require rapid and flexible adaptation.

To address these limitations, meta-reinforcement learning (meta-RL) has been proposed as a promising framework that enables agents not only to learn behaviors but also to acquire the ability to learn efficiently across tasks. As discussed in the state-of-the-art survey by Beck et al. [9], meta-RL casts the design of more efficient RL algorithms as a machine learning problem itself. Instead of training a policy for a single task, the objective is to learn a policy that can quickly adapt to new tasks sampled from a distribution, using limited interaction data. This approach addresses two of the main challenges in RL: poor data efficiency and limited generalization. By organizing meta-RL

research around the notions of task distributions and adaptation budgets, the authors provide a structured overview of existing methods and outline key open problems that must be addressed to make meta-RL a viable tool for real-world challenges such as adaptive robot locomotion.

Building on the approach proposed by Di Giuro et al. [25], significant progress has been made in applying meta-RL to achieve universal locomotion control for quadrupedal robots. Their method trains a control policy capable of zero-shot generalization, eliminating the need for robot-specific fine-tuning when deployed on new platforms. This generalization capability is enabled by training the agent on a procedurally generated set of diverse quadruped morphologies. A key component of their approach is the integration of a memory unit, which not only improves sample efficiency but also allows the policy to rapidly adapt to changes in robot properties such as mass, size, or actuator dynamics. Despite these advantages, the authors highlight an important limitation: the proposed policy is constrained to robots that share a fixed morphological template with 12 degrees of freedom. As a result, the framework does not extend naturally to quadrupeds with alternative leg structures, such as x-shaped or three-link legs, limiting its applicability to broader robot designs.

In line with the objective of achieving robust zero-shot generalization in legged locomotion, Zhao et al. [26] introduced a novel architecture designed to overcome the limitations of traditional teacher-student frameworks. Their approach leverages a Recurrent Proximal Policy Optimization (RPPO) algorithm that directly trains recurrent neural networks in partially observable environments, making the training process more stable and effective. By incorporating domain randomization, their method enhances the robustness of the learned policy across simulation-to-reality gaps, avoiding the significant performance drop often seen in student policies during deployment.

However, although the approach demonstrates generalization to terrains such as slippery surfaces, grass, and stairs, it remains dependent on exteroceptive perception—specifically LiDAR or depth cameras—to anticipate upcoming terrain features. This dependence limits its applicability in scenarios where perception may be noisy, delayed, or unavailable. In addition, while the experiments include some slippery surfaces, the work does not focus on terrains with variable or discontinuous slip conditions, such as mixed-friction patches, abrupt changes in traction, or surfaces with intermittent loss of support. These situations require rapid adaptation based mainly on proprioceptive cues and the ability to infer friction changes online, aspects that are not explicitly addressed in the proposed framework.

Belmonte et al. [27] address joint tuning and optimization of kinematic and actuator parameters in quadrupedal robots. Rather than relying on human intuition or manual tuning, the authors propose a model-free meta reinforcement learning framework that enables the co-optimization of both design and control. In this approach, a meta-learned locomotion policy is trained to quickly adapt to different robot designs, allowing it to serve as an evaluation tool during the design optimization process. This eliminates the need for predefined control rules or motion templates, enabling end-to-end design.

One of the limitations of this framework is the choice of cost functions. Standard metrics like torque minimization or mechanical power may not fully capture critical real-world factors such as energy efficiency, thermal management, or durability. Furthermore, despite leveraging meta-learning for rapid adaptation, the initial training of the meta-policy remains computationally intensive, requiring 72 hours of wall-clock time, which could hinder its scalability to more complex design spaces or multi-objective scenarios.

Unlike conventional reinforcement learning approaches that tend to produce task-specific policies with limited generalization, Kuzhamuratov et al. [28] investigate the use of meta-reinforcement learning, specifically employing the PEARL algorithm, to train a single, versatile policy. This meta-trained agent is designed to adapt to a distribution of tasks, including varied ground friction, surface inclines, and even inverted actuator controls, thereby enabling robust quadrupedal locomotion without task-specific fine-tuning. This paradigm directly addresses the core issue in robot learning where policies, once transferred to new conditions, typically exhibit a significant performance drop compared to their counterparts trained from scratch on a specific task.

A notable aspect of this meta-learning framework is its inherent temporal requirement, which introduces a critical latency period. The PEARL agent demands several episodes—approximately 160 steps each—to successfully identify the task and refine its policy. This multi-episode warm-up period represents a fundamental limitation for real-world applications where immediate response to environmental changes is essential, such as navigating rapidly shifting terrains or reacting to sudden physical perturbations. The framework also faces performance dependencies on the quality of simulation-to-real transfer, and while it outperforms MAML in real-world tests, its success is not guaranteed under significant sim-to-real gaps.

1.1.3 Blind Locomotion

A significant aspect shared by most state-of-the-art quadrupedal locomotion methods is their strong reliance on exteroceptive sensors, such as cameras or LiDAR, to perceive the environment and guide the control policy [29, 30]. As demonstrated by Dey et al., these sensors can anticipate terrain features ahead of the robot, enabling proactive gait adjustments before physical contact occurs. However, this reliance on exteroception introduces substantial challenges in real-world deployments. Beyond their vulnerability to environmental degradation—such as camera failures under low-light conditions or LiDAR degradation in fog, rain, and dust—there is a more fundamental limitation related to acquisition frequency. While cameras and LiDAR typically operate at only 10–30 Hz, proprioceptive sensors such as IMUs and joint encoders sample at 500–1000 Hz, creating a temporal mismatch that effectively increases perceptual latency within the control loop. This discrepancy reduces the controller's ability to react to fast disturbances, making exteroceptive-dependent locomotion fragile in highly dynamic scenarios.

In contrast, blind locomotion relies primarily on proprioceptive sensing, including joint encoders, IMUs, and force–torque sensors. These signals remain reliable under degraded perceptual conditions and provide high-frequency measurements that support rapid and stable control loops. However, proprioception alone lacks the ability to anticipate upcoming terrain variations, limiting the robot to purely reactive behaviors.

Building on the motivation to reduce dependence on exteroceptive information, Marcus et al. [31] propose a walking algorithm that combines a stable blind gait—operating without any visual cues—with deep reinforcement learning to enhance mobility across unstructured terrains. By prioritizing proprioceptive feedback, their approach enables quadrupedal robots to perform hazardous tasks in environments where wheeled robots are ineffective. The method focuses on precise control of leg movements and posture, demonstrating strong performance in virtual evaluations, particularly in overcoming obstacles such as stairs.

In contrast, a key limitation of the approach is that the algorithm was trained on a single stair height (10 cm), which restricts its ability to generalize. While performance remained high for slightly lower steps, it dropped substantially for higher ones—for instance, achieving only 65% success on 13 cm stairs—indicating a marked sensitivity to conditions outside the training distribution. A second important limitation lies in the use of a quasi-static blind gait: although this strategy improves stability, it inherently

constrains locomotion speed and prevents the system from leveraging more dynamic and energy-efficient gait patterns.

Beyond terrain perception, a key challenge in legged locomotion is handling unpredictable ground slip, which can lead to instability and is hard to anticipate. To address this, Nisticò et al. [32] propose a slip detection approach based exclusively on proprioceptive information, avoiding reliance on exteroceptive sensors or state estimation in the inertial frame. This choice is crucial since proprioceptive measurements, such as joint encoders and torque sensors, are not affected by drift and integration errors that typically arise when using IMU signals or world-frame estimations. Moreover, unlike force/torque or contact sensors mounted at the feet, which are prone to damage and signal discontinuities due to repeated ground impacts, proprioceptive sensing offers a more robust and reliable solution for detecting slippage. By operating solely in the body frame, their method ensures higher robustness across different gaits and environments, particularly in unstructured terrains where exteroceptive feedback can be unreliable.

Nonetheless, the approach also exhibits several important limitations. Because slip estimation is obtained by contrasting the commanded body velocity with the velocity reconstructed through forward kinematics, the method becomes highly sensitive to inaccuracies in the robot's dynamic and kinematic models. Errors arising from parameter uncertainties, actuator wear, or encoder offsets can therefore compromise detection robustness. The evaluation was also conducted under constrained conditions, with predefined gaits and moderate locomotion speeds, which restricts the applicability of the proposed proportional-derivative control strategy to more dynamic behaviors such as rapid trotting or agile maneuvers. Moreover, the method is limited to detecting slip events and does not incorporate any mechanism for online compensation or adaptive control, leaving the problem of how the robot should react to the detected slip unresolved.

Building upon the need for accurate slip detection to ensure stable locomotion, Carvalho et al. [6] conducted a comparative study of slip detection methods based on thresholding foot velocity and acceleration during the contact phase. Implemented on a simulated quadruped robot using a Model Predictive Control (MPC) framework and a Whole Body Controller (WBC), their work evaluates multiple proprioception-based methods across different friction conditions. Additionally, they introduce a novel approach that filters slip occurrences during the stance-to-swing transition, leveraging the relationship between tangential and orthogonal foot velocities at ground contact.

However, the proposed methods rely on foot-velocity and acceleration

thresholds empirically tuned for specific gaits and friction settings; as a result, their performance may degrade when the robot operates outside these conditions or when transitioning between different contact regimes. The slip-filtering strategy introduced for the stance-to-swing transition, although effective in reducing false positives, is tightly coupled to the locomotion pattern used in the experiments and may not generalize to more dynamic or non-periodic motions. Finally, the evaluation focuses exclusively on detection accuracy and does not address how slip information could be integrated into the MPC or WBC layers to enable adaptive compensation or recovery behaviors, leaving the control implications of slip unexamined.

Complementing the efforts to detect and predict slippage, Teng et al. [33] introduce a state estimation framework designed specifically for legged robots navigating slippery environments. The approach employs an Invariant Extended Kalman Filter (InEKF) that fuses inertial measurements, velocity data from a tracking camera, and leg kinematics constraints to accurately estimate the robot's state despite challenging conditions.

Although, the proposed InEKF relies on external velocity measurements from a tracking camera, which constrains its applicability in outdoor or cluttered environments where visual tracking is unreliable or unavailable. Moreover, although the filter incorporates leg-kinematic constraints, it still assumes accurate knowledge of contact events; misclassification of contact states—common under high slip or uneven terrain—can introduce significant estimation errors. The method is validated primarily in controlled laboratory conditions using planar gaits and moderate speeds, leaving its performance under highly dynamic motions or complex 3D terrains largely unexplored. Additionally, the framework focuses solely on state estimation and does not address how the improved estimates could be integrated into a locomotion controller to achieve slip-aware stabilization or recovery. Finally, the algorithm's computational cost is non-negligible due to its reliance on Lie group operations, which may limit deployment on embedded hardware with strict real-time constraints.

In summary, blind locomotion must depend primarily on proprioceptive cues to detect and adapt to slip, making this a challenging and practically relevant problem in quadrupedal locomotion.

1.2 Contributions

A review of the existing literature shows that quadrupedal locomotion on unknown terrains remains an active research challenge, largely due to the strong dependence of many state-of-the-art methods on exteroceptive sensing

and the limited adaptability of proprioceptive-only RL approaches. Vision- and LiDAR-based pipelines can anticipate terrain features but introduce vulnerabilities such as perceptual failures, environmental sensitivity, and control-loop latency. Conversely, standard RL policies trained purely from proprioception tend to generalize only within training distributions and often suffer from sample inefficiency. Although meta-RL methods aim to improve adaptation, many rely on few-shot procedures that require several episodes to infer task dynamics, making them unsuitable for real-world scenarios where friction and terrain conditions may change abruptly from one timestep to the next.

Motivated by these limitations, this work proposes a Meta-Reinforcement Learning approach for blind quadrupedal locomotion that enables zero-shot adaptation, fast response to abrupt terrain changes, and robust performance in slippery and unstructured environments. Relying solely on proprioceptive feedback, the proposed method leverages two RL^2-inspired recurrent variants based on PPO, enabling continuous task inference at every timestep. This allows the policy to extract temporal regularities that encode latent terrain properties and evolving slip dynamics, enabling immediate corrective actions rather than depending on multi-episode warm-up phases, as required by many few-shot meta-learning approaches. As a result, the recurrent policy improves robustness to both geometric irregularities and sudden traction changes, addressing key gaps in existing proprioceptive and meta-RL locomotion methods.

To evaluate these contributions, we assess the two RL^2-based variants against standard PPO across environments with varying levels of terrain complexity and slippage. Our results demonstrate improved adaptation and strong generalization to unseen and challenging terrain–slip combinations, showcasing the effectiveness of the proposed meta-RL formulation for blind locomotion. Notably, this research led to the paper "Meta Reinforcement Learning Applied to Quadrupedal Robots for Blind Locomotion and Fast Adaptation on Unknown Terrains", which was accepted and awarded Best Paper at the 2025 IEEE International Conference on Advanced Robotics (ICAR).

1.3 Outline

The remainder of this thesis is organized as follows. Chapter 1 introduces the related work, covering control strategies for locomotion, reinforcement learning approaches, and the specific context of blind quadrupedal locomotion. Chapter 2 presents the theoretical foundations of reinforcement learning and meta-reinforcement learning, including the main algorithmic paradigms,

policy optimization methods, and recurrent neural architectures. Chapter 3 details the methodology, describing the robot model, control architecture, reinforcement learning formulation, and the simulation environments considered in this work. Chapter 4 reports the simulation results, comparing the proposed recurrent PPO variants with the baseline PPO across different terrain conditions. Chapter 5 concludes the thesis, highlighting the main contributions and discussing directions for future work.

2
Theoretical Foundations

This chapter provides the theoretical background necessary to understand the methods applied throughout this work. It first introduces the fundamental concepts of Reinforcement Learning (RL), covering classical solution methods, value function approximation, and modern policy optimization techniques. Following this, the discussion extends to Meta-Reinforcement Learning, presenting its connection to Partially Observable Markov Decision Processes (POMDPs), recurrent neural networks, and recent approaches for fast adaptation in dynamic environments. Together, these foundations establish the framework upon which the proposed strategies for quadrupedal locomotion are built.

2.1
Reinforcement Learning Background

This chapter introduces the core principles of Reinforcement Learning (RL), which form the basis for the control strategies developed in this work for quadrupedal locomotion.

2.1.1
Fundamentals of Reinforcement Learning

Reinforcement Learning (RL) [4] is a machine learning technique for solving sequential decision-making problems. It allows an agent to learn optimal policies through continuous interaction with a stochastic environment—one in which the outcomes of actions are uncertain or inherently variable. Within the RL framework, these sequential decision-making processes in stochastic environments are formally modeled as a Markov Decision Process (MDP), which serves as the foundational theoretical basis of this field.

An MDP is defined by a tuple $< S, A, P, R >$, formed by: a set of states $s \in S$, representing all possible situations that the system can be in; a set of actions $a \in A$, which includes all possible actions the agent can take; a transition function $P(s'|s, a)$, which defines the probability of transitioning from state s to next state s' given that action a was taken. This reflects the Markov property, indicating that the next state depends only on the current state and the current action; a reward function (R), which specifies the value the agent receives when taking an action—this is the signal the agent seeks to optimize.

In the context of Reinforcement Learning, the MDP is applied in a continuous trial-and-error cycle, where the agent interacts with the environment—which includes both the robot and the scenario in which it operates. During this interaction, the agent performs actions and observes their effects on the environment. Rather than having prior knowledge of the system's dynamics, such as the transition and reward functions, the agent learns through interaction, gradually adjusting a policy based on the received return. This iterative process of interaction between the agent and the environment is fundamental for learning in dynamic and uncertain scenarios: the agent observes the current state, selects an action based on its policy, executes the action in the environment, receives a reward, and observes the resulting new state. This cycle is illustrated in Figure 2.1.

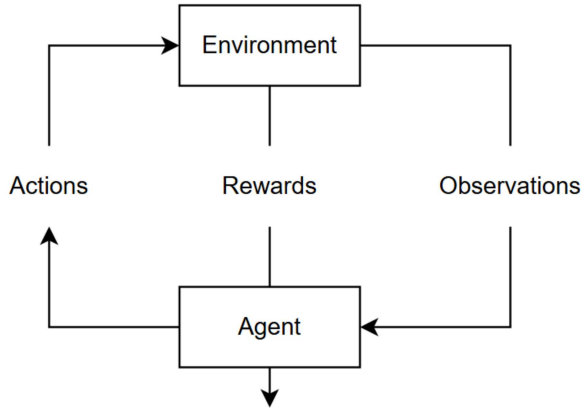

Figure 2.1: Interaction cycle between the agent and the environment in RL

The policy $\pi(a|s)$, which represents a strategy or rule that defines the actions the agent should take in each state of the environment, guides the agent's behavior throughout episodes within an MDP. These episodes begin in initial states sampled from a distribution P_0, which defines the probability distribution over initial states. From these starting points, episodes evolve as the agent takes actions and transitions between states according to the environment's dynamics, described by the transition function $P(s'|s, a)$. Each transition is associated with a reward $r_t(s_t, a_t)$, and after T transitions, the episode ends, allowing a new one to begin from another sampled initial state.

The data collected throughout an episode, $\tau = \{(s_t, a_t, r_t, s_{t+1})\}$, is often referred to as a trajectory, representing the sequence of interactions between the agent and the environment [9].

Each action taken by the agent yields a reward or penalty and results in a change in the system's state, providing valuable information for the continuous update of the policy. Over time, the agent aims to refine its decision-making by

exploring different actions and learning strategies that maximize the expected return, which corresponds to the accumulated reward over time, weighted by a discount factor $\gamma \in [0, 1]$.

Formally, the return R_t at time step t is defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2-1}$$

where r_{t+k+1} represents the future rewards received by the agent. The discount factor controls the importance of future rewards relative to immediate ones, with $\gamma = 0$ prioritizing only immediate rewards, while values of γ close to 1 assign greater weight to long-term rewards.

Having formalized the concept of rewards, we can define the main objective of Reinforcement Learning: obtain an optimal policy π^* that maximizes the expected return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} R_t \tag{2-2}$$

2.1.1.1 State Value Estimation

Given that the goal of Reinforcement Learning (RL) is to find an optimal policy π^* that maximizes the expected return, it becomes essential to define a metric capable of quantifying the quality of the agent's states and actions. One of this metric is expressed by the value function $V^{\pi}(s)$ [4], which represents the expected return (R) when following a policy π from a given state s. This function provides a fundamental abstraction for evaluating the relevance of being in a particular state, regardless of the specific actions taken at that moment. The value function $V^{\pi}(s)$ is defined as

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\{R_t | s_t = s\} \\ &= \mathbb{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}. \end{aligned} \tag{2-3}$$

This notation allows for generalization and simplification of the analysis, focusing on the expected return without directly depending on the internal structure of the policy. As a result, $V^{\pi}(s)$ becomes more useful in many situations because it provides a clear and quantitative metric for evaluating states, while the policy π is concerned only with the actions taken. Based on this, one can use the definition in Equation 2-2 to derive the optimal value function $V^{\pi^*}(s)$:

$$\pi^* = \arg\max_{\pi} V^{\pi} \tag{2-4}$$

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \tag{2-5}$$

To make the value function suitable for use in RL algorithms, it is often expressed through a recursive definition that relates the value of a state to the immediate reward and the expected value of the subsequent states:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s')\middle| s_t = s\right\} \end{aligned} \tag{2-6}$$

A straightforward approach to computing $V^{\pi}(s)$ is the tabular method, where the value function is stored as an array or table. This method is highly effective for problems with small, discrete state and action spaces, as it provides a simple and exact solution for finite Markov Decision Processes (MDPs) [4].

However, tabular methods face significant limitations that hinder their practicality for many real-world problems. The primary issue is the curse of dimensionality: the size of the table grows exponentially with the number of state or action variables, making it computationally infeasible for environments with high-dimensional or continuous spaces. Furthermore, tabular representations lack a mechanism for generalization; each state is treated as entirely unique, and any learning must occur independently for each one. This is inefficient, as many states will never be visited.

It is precisely these limitations that justify the use of function approximation [4]. By employing a parameterized function $\hat{V}(s, w) \approx V^{\pi}(s)$—where w is a weight vector—an agent can generalize from a limited set of experiences to estimate the value of unseen, but similar, states. This approach provides critical advantages, including scalability to handle high-dimensional or continuous state spaces that are intractable for tabular methods, generalization to allow knowledge learned in one state to inform predictions about similar states, thereby drastically improving sample efficiency, and enhanced capability in partially observable environments.

2.1.1.2 Reinforcement Learning in the Real World

When applying Reinforcement Learning (RL) in real-world scenarios, the environment model is often unknown, which prevents access to information about the transition function $P(s'|s, a)$ and the rewards r. In such cases, the agent must learn an optimal policy through direct interaction with the environment, collecting experience samples over time. Methods that enable learning policies without requiring an explicit model of the environment are known as model-free methods.

When facing a model-free problem, a new challenge arises: since the agent learns the policy through interactions with the environment, it is necessary to explore the environment in an efficient manner, where the algorithm must visit states strategically in order to discover the most rewarding ones. Therefore, it becomes essential to balance two concepts: the exploration phase, which enables the agent to try new actions and discover potentially better rewards in each state; and the exploitation phase, in which the agent chooses actions that have previously yielded the highest rewards [4].

This balance is crucial because, without adequate exploration, the agent may become stuck in suboptimal solutions, ignoring actions or states that could lead to higher rewards. On the other hand, excessive exploration can cause the agent to waste resources on actions that do not improve performance, compromising learning efficiency. A proper balance between exploration and exploitation ensures that the agent can learn effectively, leveraging acquired knowledge while still seeking new opportunities to optimize its policy [4].

Considering model-free methods, there are two distinct strategies that define how the agent interacts with the environment and optimizes its policy: on-policy and off-policy [4] [34]. In on-policy learning, the policy that guides the agent's actions is the same policy being evaluated and improved during training. This approach allows the agent to iteratively adjust its behavior based on the outcomes obtained directly from its own interactions. In contrast, in off-policy learning, the policy used to generate actions (the behavior policy) differs from the policy being evaluated and optimized (the target policy). This separation makes off-policy methods more flexible, as the agent can explore the environment using one policy while simultaneously improving another.

Based on these concepts, it is possible to develop methods for estimating value functions and discovering optimal policies without relying on explicit information about the environment's model, using only past experiences to improve the agent's decision-making process.

2.1.2 Core Algorithmic Paradigms

In order to obtain an optimal policy that maximizes the RL objective, it is essential to employ a method or algorithm capable of efficiently estimating or computing the optimal value function. In this section, we describe some of the most important methods for solving RL problems: Dynamic Programming, Monte Carlo, Temporal Difference learning, Policy Gradient Methods, Actor-Critic Method and Proximal Policy Optimization [4]. Our focus is on the core ideas behind each approach, highlighting their fundamental principles.

2.1.2.1 Dynamic Programming

Among the algorithms used for this purpose, the main ones are based on dynamic programming (DP) [4], a problem-solving technique that decomposes complex problems into smaller subproblems. In the context of Reinforcement Learning (RL), DP computes value functions and optimal policies assuming full access to the model of the environment, which includes both state transitions and rewards. This makes DP particularly powerful in planning scenarios, where the environment dynamics are fully known, but less suitable for direct learning from raw experience.

A key principle underlying DP is the concept of bootstrapping. Instead of waiting for all transitions and rewards to be observed and calculated exactly, bootstrapping updates current estimates of the value function using estimates of successor states. In other words, the value of a state is refined incrementally by combining immediate rewards with the current approximation of the value of the next states, as in Eq. (2-6). This approach allows learning to proceed step by step, speeding up the process and reducing dependence on exact computations.

Although DP itself is rarely applied in practical RL due to its reliance on a complete and explicit model of the environment, many modern algorithms inherit its theoretical foundation. In particular, they adopt the principle of bootstrapping while addressing DP's limitations, such as high computational cost and the requirement of full knowledge of the environment dynamics.

2.1.2.2 Monte Carlo

Monte Carlo methods [4, 34] estimate value functions and optimal policies in episodic tasks, where each episode has a clear beginning and end, regardless of the agent's actions. They are particularly useful when the environment

model is unknown, since they rely solely on sampled experience gathered through interaction, without requiring prior knowledge of state transitions.

The central idea is to approximate the value function directly from complete episodes. Each episode provides a sequence of visited states, chosen actions, and observed rewards, from which returns are computed. Updates are then applied incrementally at the end of the episode, once the full return is available. This distinguishes Monte Carlo from bootstrapping methods, which update estimates at every step using predictions of successor states.

Formally, given a state s_t, the value estimate is updated toward the observed return G_t as:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[G_t - V(s_t) \right], \tag{2-7}$$

where G_t is the actual return accumulated after time t, and α is the learning rate that controls both the influence of new experiences and the convergence of the algorithm.

Monte Carlo offers a crucial advantage: independence from an explicit model of the environment. By relying exclusively on sampled trajectories, Monte Carlo avoids the need for transition dynamics and reward models, making it particularly suitable for complex or unknown environments. This property provides flexibility and scalability, while mitigating the computational burden typically associated with model-based approaches [34]. Additionally, by relying on the forward definition of the value function (2-3), it avoids the bias introduced by bootstrapping methods.

2.1.2.3 Temporal Difference Method

The Temporal Difference (TD) method [4, 34] is characterized by a synthesis of Monte Carlo and Dynamic Programming techniques. Like Monte Carlo, it learns directly from experience without requiring an explicit model of the environment. At the same time, it inherits from Dynamic Programming the use of bootstrapping—that is, updating value estimates based on other learned estimates, without waiting for the end of an episode. In this way, instead of relying on cumulative rewards from complete episodes, TD methods use the immediate rewards received at each time step, enabling continuous updates and generally faster convergence of the learning process.

The key difference between Monte Carlo and TD lies in the timing of the updates. While Monte Carlo requires the completion of an entire episode to compute value estimates, TD can update the value function V after each transition, using the immediate reward r and the estimate of the next state

$V(s_{t+1})$. A typical TD update is given by:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right], \tag{2-8}$$

where γ is the discount factor, determining the importance of future rewards relative to immediate ones, and α is the learning rate parameter. This formulation highlights how the estimate of the value function $V(s_t)$ is adjusted incrementally, characterizing the process of bootstrapping.

A central concept in the TD method is the TD error (or prediction error), which measures the discrepancy between the current estimate of a state's value and the sum of the observed reward plus the estimated value of the next state. Formally, it is defined as [34]:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{2-9}$$

The TD error δ_t serves as a feedback signal, indicating how well the current value function matches the agent's actual performance. Updates to V are driven by this error, progressively reducing the gap between predictions and reality. Over time, minimizing the prediction error leads the value estimates to converge toward the true expected values, ensuring continuous improvement of the agent's policy.

2.1.2.4 Policy Gradient Methods

In traditional reinforcement learning approaches, function approximation is typically applied to estimate value functions, from which the agent derives its behavior. This process generally involves two stages: first, estimating the value of each action or state, and then selecting actions according to a decision rule based on these estimates.

An alternative is to parameterize the policy directly [4], representing it as a function that maps states to a probability distribution over actions. This approach removes the need to store or query value functions, allowing the agent to learn the state–action mapping in a single stage. In many scenarios, it is also easier to approximate a policy than a value function, especially in high-dimensional or continuous action spaces.

Policy-based methods are particularly effective in continuous domains, where the number of possible actions is infinite. Instead of discretizing the action space—an often infeasible task—the policy is modeled as a continuous parameterized function, frequently represented by a probability distribution. The learned parameters then describe the distribution's statistics, such as its mean and standard deviation, enabling compact representation, efficient

exploration, and controlled variability around the central policy.

These properties make policy parameterization more sample-efficient, capable of faster convergence, and better suited for incorporating prior knowledge about the task to guide learning toward optimal behavior. A parameterized policy can be expressed as:

$$\pi(a|s,\theta) = P\{a_t = A|s_t = S, \theta_t = \theta\} \tag{2-10}$$

which denotes the probability of selecting action a in state s, given parameters θ at time t.

As with the previously discussed methods, the goal remains to maximize the expected return. However, in this case, we define a performance measure $J(\pi)$, which evaluates the expected performance of a policy π. The exact definition of $J(\pi)$ may vary depending on the specific objectives of the problem.

From this definition, the aim is to find a parameterized policy π_θ that maximizes $J(\pi_\theta)$, which is equivalent to determining the optimal parameters θ^* that yield the highest expected return:

$$\theta^* = \arg\max_\theta J(\pi_\theta) \tag{2-11}$$

To optimize $J(\pi_\theta)$, we employ gradient ascent given by

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\pi_\theta)}, \tag{2-12}$$

where α is a learning rate and $\widehat{\nabla J(\pi_\theta)}$ is a stochastic estimate of the true gradient $\nabla_\theta J(\pi_\theta)$, computed from sampled trajectories. Due to the complexity of the environment and the stochastic nature of RL, computing the exact gradient is often infeasible; instead, sampling-based estimates are used. There are several methods to compute this gradient estimate, with the Policy Gradient Theorem (PGT) [4] being one of the fundamental approaches, providing the analytical expression for parameter updates. In practice, automatic differentiation frameworks such as PyTorch or TensorFlow are used to efficiently compute the gradient $\widehat{\nabla J(\pi_\theta)}$ from $J(\pi_\theta)$ via backpropagation, abstracting away the complexity of manual calculation. Over time, the average of these estimates converges to the true gradient, enabling effective policy optimization.

2.1.2.5 Actor-Critic Method

Aiming to enhance the efficiency and stability of gradient-based methods in reinforcement learning, a class of approaches has been developed that employs a dual structure composed of an Actor and a Critic [35]. In this structure, the policy $\pi(a|s;\theta)$ represents the Actor, responsible for selecting

actions, while the approximation of the value function $\hat{V}(s, w)$ represents the Critic, tasked with evaluating the actions taken based on their estimates. The Critic provides a measure of the quality of the selected actions, allowing the policy parameters of the Actor to be updated in a guided manner, leading to actions that improve overall performance.

Thus, the actor, which defines the policy, and the critic, which estimates the value function, together provide a balance between policy-based and value-based methods: the Actor directly optimizes the policy in a way that supports computing continuous actions without the need to acess and optimize an value function. On the other hand, the Critic reduces the variance of gradient estimates by providing a learned baseline, improving convergence speed and stability. This synergy makes Actor–Critic algorithms well-suited for high-dimensional, continuous control problems, and has led to their adoption in several state-of-the-art methods.

2.1.2.6 Proximal Policy Optimization - PPO

The Proximal Policy Optimization (PPO) algorithm [11] is an on-policy reinforcement learning method that aims to optimize an agent's policy using only first-order optimization, i.e., relying solely on the gradient of the objective function for policy updates. Its main goal is to achieve high computational efficiency while maintaining training stability and improving the reliability of the data used for learning.

PPO follows the actor-critic framework, where the actor represents the policy π_θ, responsible for selecting actions, while the critic estimates the value function V_w that guides policy updates. To optimize the policy, the method alternates between collecting samples from the policy and performing optimization based on the collected samples. Although PPO reuses this data for several gradient steps, the updates are confined to the current batch, preserving the algorithm's on-policy nature and improving sample efficiency.

To achieve this, PPO employs an objective function known as the clipped surrogate objective, which enables multiple updates on minibatches drawn from the same set of recently collected trajectories before acquiring new data. This approach allows PPO to extract more learning signal from the same data, thereby accelerating training without compromising stability.

This sample reuse is achieved through a reformulation based on importance sampling (IS) [36], which makes it possible to estimate an expectation under a target distribution $p(x)$ from samples obtained from a different distribution $q(x)$. This relationship is expressed as:

$$\mathbb{E}_{x\sim p(x)}\{f(x)\} = \mathbb{E}_{x\sim q(x)}\left\{\frac{p(x)}{q(x)}f(x)\right\}. \tag{2-13}$$

In the context of PPO, this technique is applied through the probability ratio ρ, which quantifies the difference between the current policy π_θ and the previous policy π_β used to collect the samples. This ratio is defined as:

$$\rho = \frac{\pi_\theta(a_t|s_t)}{\pi_\beta(a_t|s_t)} \tag{2-14}$$

This term adjusts the influence of each sample in learning, taking into account the difference between the probability of the new policy π_θ selecting action a_t and the probability with which the previous policy π_β selected that same action. In this way, the agent can efficiently reuse previously collected data, assigning higher weight to actions the new policy deems more advantageous and reducing the importance of less favorable actions. This leads to the construction of the surrogate objective function:

$$\begin{aligned} J_{is}(\theta) &= \mathbb{E}_{\pi_\beta}\left\{\frac{\pi_\theta(a|s)}{\pi_\beta(a|s)}A^{\pi_\theta}(s,a)\right\}, \\ &= \mathbb{E}_{\pi_\beta}\left\{\rho A^{\pi_\theta}(s,a)\right\}, \end{aligned} \tag{2-15}$$

where $A^{\pi_\theta}(s,a)$ is the advantage function, estimated under the old policy π_θ. The advantage function measures whether an action a in state s is better or worse than the policy's average action. By reweighting this advantage by the probability ratio $\rho(\theta)$, $J_{IS}(\theta)$ estimates the performance of the new policy π_θ using data collected from π_β.

A common method for estimating the advantage is Generalized Advantage Estimation (GAE) [37], which provides an approach that balances bias and variance in advantage calculation by combining multiple temporal-difference (TD) error estimates with a smoothing factor. GAE is defined as:

$$\hat{A}^{\pi_\theta}(s,a) = \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}, \tag{2-16}$$

where γ determines the weight given to future reward contributions, while λ controls the trade-off between bias and variance, allowing adjustment between a low-bias/high-variance estimator and a high-bias/low-variance estimator. In this formulation, δ represents the temporal-difference (TD) error, computed as the difference between the estimated value of the current state and the estimated value of the next state. Setting $\lambda = 0$ makes the method equivalent to Temporal Difference (TD), while $\lambda = 1$ makes it equivalent to the Monte Carlo estimation method.

However, directly optimizing $J_{is}(\theta)$ often results in excessively large policy updates, leading to a collapse in the agent's performance. To mitigate this problem, PPO introduces a mechanism to control policy change through clip-

ping [11] in the surrogate objective, limiting the probability ratio within a predefined range. This mechanism prevents the policy from changing abruptly between consecutive updates, ensuring training stability and avoiding undesirable oscillations in performance. The PPO objective function with clipping is defined as:

$$J_{\text{clip}}(\theta) = \mathbb{E}_{\pi_\beta}\left\{\min\left[\rho A^{\pi_\theta}(s,a), \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A^{\pi_\theta}(s,a)\right]\right\}, \tag{2-17}$$

where ϵ is a hyperparameter that controls the allowable range for policy updates. The clip operator restricts ρ to the interval $[1-\epsilon, 1+\epsilon]$, removing the incentive for ρ to move excessively outside this region. If the ratio ρ exceeds the limits of the interval, the policy update will not result in an additional increase in the objective function. In this way, the algorithm balances the exploration of new policies with training stability, ensuring that updates are not so large as to degrade the performance of the learned policy.

Algorithm 1: Proximal Policy Optimization - PPO

for *iteration = 1, 2, ...* **do**
  for *actor = 1, 2, ..., N* **do**
    Run policy π_β in the environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_T$
  end for
  Optimize the surrogate objective J with respect to θ,
  using K epochs and a minibatch size $M \leq N \times T$ $\theta_\beta \leftarrow \theta$
end for

2.2 Fundamentals of Meta-Reinforcement Learning

Meta-reinforcement learning (Meta-RL) is an advanced approach within reinforcement learning that seeks to learn a new policy capable of adapting quickly to novel tasks using prior experience and minimal data. The central idea is "learning to learn", where, during meta-training, the agent learns to identify and reuse common patterns across tasks. This enables the agent to start new tasks with a good initialization of the policy parameters or a shared representation of the solution space, thus reducing the amount of exploration required. In this way, policies are developed with the capacity to adapt efficiently to new conditions or environments, even when only limited data or interactions are available.

Formally, it is assumed that tasks M^i are sampled from a distribution $p(M)$ over MDPs, where each MDP defines a distinct task. During meta-

training, the agent is exposed to a wide variety of tasks sampled from M, allowing it to develop an internal adaptation mechanism designed to generalize to unseen situations and adjust its behavior efficiently. To ensure this generalization capability, the agent is encouraged to learn an initial policy that is robust across the support of $p(M)$, which facilitates rapid adaptation to new tasks.

This policy facilitates adaptation to new tasks in two main ways: in Few-Shot Meta-Learning, the agent adapts to a new task with only a few additional training iterations; while in Zero-Shot Meta-Learning, the agent leverages an internal memory or latent representation that summarizes information from past interactions, enabling immediate adaptation to new tasks without further training.

Because of its ability to identify and reuse patterns in new tasks, Meta-RL agents can explore environments in a more guided and less random manner, making better use of each collected data sample. This results in a significant improvement in sampling efficiency. However, this advantage comes at two main costs. First, meta-training requires a substantial amount of data, since the agent must be exposed to diverse states, rewards, and dynamics across many tasks, which increases both the computational effort and the demand for environment interactions. Second, overfitting to the meta-training tasks may reduce the agent's ability to generalize to tasks outside the training distribution, since the model may specialize in task-specific patterns rather than learning more general strategies.

The core idea of Meta-RL is that, while a conventional RL algorithm learns a policy π, Meta-RL learns the RL algorithm f that generates this policy. In this framework, Meta-RL is structured around two levels: the inner loop, which handles fast, task-specific adaptation, and the outer loop, which performs Meta-training and optimizes the meta-parameters that condition the inner loop. The outer loop adjusts either the initial parameters of the policy or the structure of the agent so that the inner loop can adapt more efficiently to new tasks. During meta-training, the performance of the inner loop is used to update the meta-parameters of the outer loop. The overall system is illustrated in Figure 2.2.

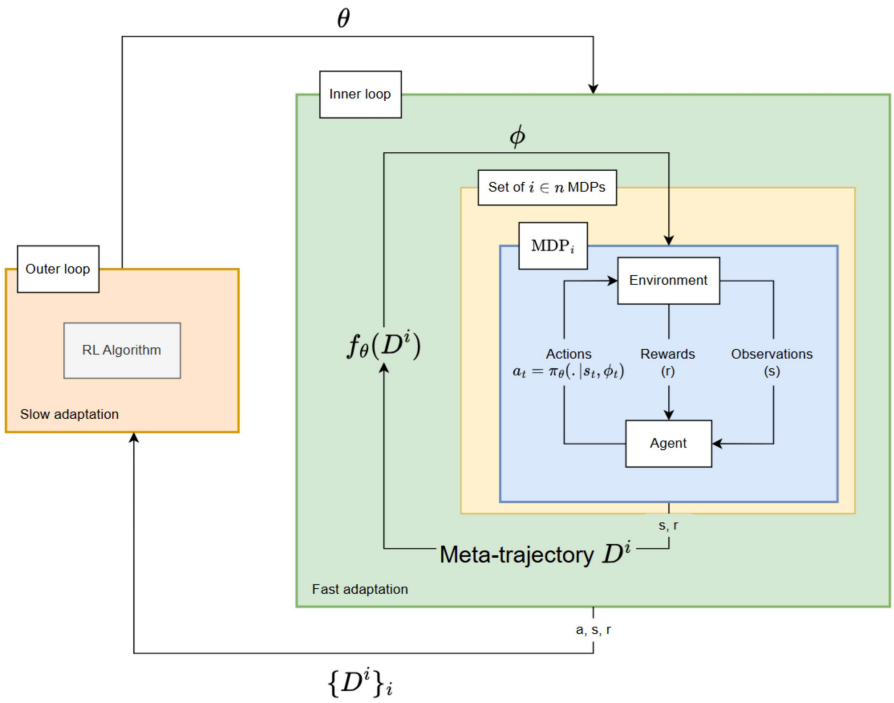

Figure 2.2: A complete Meta-Reinforcement Learning (Meta-RL) system

In summary, Meta-RL aims to meta-learn a reinforcement learning algorithm capable of rapid adaptation to new MDPs, also referred to as tasks. During meta-training, a task is sampled from the distribution $p(M)$, the inner loop algorithm is applied to solve it, and the outer loop updates the meta-parameters to improve the resulting policy. This iterative process enables efficient adaptation to novel tasks and environments.

2.2.1 Meta-Reinforcement Learning Architecture

In Meta-RL, learning occurs across two hierarchical layers: a fast inner loop and a slow outer loop. The inner loop is responsible for solving individual tasks sampled from a task distribution, efficiently adapting the policy to each specific context. In contrast, the outer loop optimizes the meta-parameters that condition the inner loop, accumulating experience across multiple tasks to guide future adaptations. This hierarchical structure combines short-term efficiency (fast task-specific adjustments) with long-term generalization (transferable knowledge across tasks).

Formally, in the inner loop, the agent follows a policy $\pi_{\theta,\phi}$, where θ represents the meta-parameters and ϕ the task-adapted parameters. These task-specific parameters ϕ are obtained through a function f_θ, which leverages the agent's interactions with the environment to produce fast adaptations. In this context, ϕ can be interpreted either as a set of locally adjusted policy

parameters or as the hidden state of a network that encodes task-specific information.

The outer loop is responsible for updating the meta-parameters θ, which remain fixed during the execution of each task but are gradually adjusted across multiple tasks using reinforcement learning algorithms. These meta-parameters determine how the inner loop constructs its task-specific representation from experience during each episode and accumulate global knowledge about the distribution of tasks, thereby enabling faster and more efficient adaptation in future tasks.

The roles of θ and ϕ are therefore complementary: ϕ allows the agent to quickly adapt its policy to a specific task, while θ ensures that this adaptation is efficient and generalizable based on experience accumulated across tasks. Typically, all tasks share the same state space s and action space a, differing only in the reward function $R(s, a)$ and transition function $P(M)$, which ensures structural consistency for the learning process across both layers.

Considering this loop structure, the main objective of Meta-RL is to evaluate and improve the performance of the inner-loop algorithm across the different tasks to which it is exposed. In this context, the main objective of Meta-RL is to optimize the ability of the inner-loop algorithm to quickly adapt to new tasks, reflecting the effectiveness of the outer-loop training. Formally, this objective in the context of Meta-RL can be expressed as follows:

$$\mathcal{J}(\phi) = \mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M})} \left[\mathbb{E}_D \left[\sum_{\tau \in \mathcal{D}_{K:H}} \sum_{t=0} \gamma^t r_t \,\middle|\, f_\theta, \mathcal{M}^i \right] \right], \tag{2-18}$$

where $\mathbb{E}_{\mathcal{M}_i \sim p(\mathcal{M})}$ denotes the expectation over tasks $\mathcal{M}_i$ sampled from a task distribution $p(\mathcal{M})$, $\mathbb{E}_D$ represents the expectation over trajectories D generated for each task, $\mathcal{D}_{K:H}$ specifies a segment of a meta-trajectory from time K to H, and $\sum_{t=0} \gamma^t r_t$ is the discounted accumulated return. This formalization captures both the fast adaptation of the inner loop and the long-term optimization of the outer loop, highlighting the complementary roles of ϕ and θ in Meta-RL.

2.2.2 Partially Observable Markov Decision Process - (POMDP)

In conventional reinforcement learning, the environment is typically modeled as a Markov Decision Process (MDP), defined by the tuple $\langle S, A, P, r \rangle$, including the states $s \in S$, actions $a \in A$, state transition functions $P(s'|s, a)$ and reward associated with the transition from state s to s', $r(s, a, s')$. However, in many real-world scenarios, the agent cannot directly observe the complete state of the environment, making it necessary to extend this formalism.

To address this limitation, the Partially Observable Markov Decision Process (POMDP) [9] is introduced. A POMDP is characterized by the tuple $\langle S, A, \Omega, P, r, O \rangle$, where Ω represents the set of possible observations, and O is the observation function defining the probability of receiving observation $o \in \Omega$ given the true state s and the previous action a_{t-1}. This formulation captures the fact that the agent interacts with the environment under partial observability and must make decisions based only on limited information.

Although the agent does not have direct access to the true state, the underlying structure of the MDP—such as the reward function and the transition dynamics—remains unchanged in the POMDP formulation. The key difference lies in the necessity to infer the hidden state based on past experience. Within the context of Meta-RL, it is natural for the task identity to be encoded in the hidden state during the process of fast adaptation.

2.2.3 Artificial neural networks

Artificial Neural Networks, also known as Feedforward Neural Networks [38] is an attempt to recreate the learning and adaptive capabilities of the biological brain. Similar to their biological counterpart, ANNs are composed of interconnected processing units called neurons. These artificial neurons are connected with each other via synaptic connections, each characterized by a weight. The core operation of a neuron is to compute the weighted sum of its inputs, in which is achieved by multiplying each input signal by the weight of its respective connection and summing these products. This aggregated sum is then passed through a non-linear activation function to produce the neuron's output.

When considering ANN categories, two architectures stand out: the conventional feedforward neural network and the recurrent neural network (RNN). In feedforward neural networks, there are no feedback connections; information flows sequentially from one layer to the next without any recurrence. information from previous time steps when processing subsequent inputs. In contrast, in RNN, the output of neurons is fed back into the network as inputs, allowing it to retain and reuse. Figure 2.3 shows a common type of ANN: a fully connected, multi-layer feedforward neural network. It is composed of four layers: an input layer, an output layer, and two hidden layers between them, which are responsible for enabling the network to extract higher-order statistics. In contrast, Figure 2.4 depicts the RNN architecture, where some neurons have recurrent connections to previous layers, allowing the network to maintain and utilize temporal information.

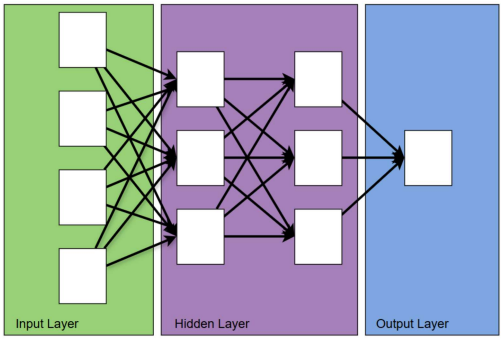

Figure 2.3: Fully Connected Multi Layer FeedForward Neural Network

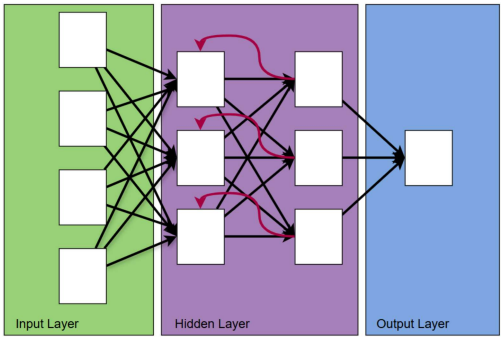

Figure 2.4: Recurrent Neural Network

Figure 2.5: Types of Artificial Neural Network

Considering both types of neural networks, their objective is to achieve their most essential capability: the ability to learn. Through the adjustment of synaptic weights during the training process, these networks gradually improve their performance, enabling them to generalize from examples and solve complex tasks.

2.2.3.1 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) [39] is a type of recurrent neural network (RNN) that introduces a more sophisticated mechanism through the use of gating functions. Unlike conventional RNNs, which resemble feedforward neural networks but with the ability to process variable-length sequential inputs via recurrent hidden states, GRUs employ update and reset gates to regulate the flow of information. These gates enable the network to decide which information to retain, update, or discard over time, making learning more efficient.

Although RNNs are effective in modeling temporal dependencies, they struggle to capture long-term dependencies due to the vanishing and exploding gradient problems inherent in gradient-based optimization methods. In practice, gradients associated with distant time steps tend to either diminish exponentially (vanishing gradients), making it difficult for the model to learn from long-range context, or grow uncontrollably (exploding gradients), destabilizing training. GRUs were proposed to address these limitations by incorporating gating mechanisms that preserve relevant long-term information while mitigating the instability issues of conventional RNNs.

In this context, GRU introduce a key improvement over traditional RNNs. Rather than completely overwriting the hidden state with new values

derived from the current input and the previous state, they retain the existing content and selectively update it with new information. This mechanism offers important advantages: it allows the network to preserve relevant features over long sequences and establishes shortcut paths through time, which facilitate backpropagation of errors without them vanishing too quickly. Furthermore, when training GRU-based models, the sequence length—that is, the number of time steps unrolled during training—plays a critical role. A longer sequence length allows the model to capture broader temporal dependencies, while a shorter one reduces computational cost and mitigates gradient instability, making the choice of this parameter highly problem-dependent.

2.2.4 Fast RL via slow RL (RL^2)

The Fast Reinforcement Learning via Slow RL (RL^2) [10] framework can be understood as a practical implementation of Meta-Reinforcement Learning. Its goal is to enhance a policy's ability to adapt quickly to new tasks by leveraging data collected during training, enabling the agent to generalize and perform effectively in unseen environments without additional retraining—a concept known as zero-shot learning. Such adaptability is particularly valuable when data collection or continuous online learning is costly, allowing the agent to handle diverse tasks efficiently based solely on prior experience [9].

In this framework, the policy depends not only on the current observation but also on the entire history of interactions with the environment, including past states, actions, and rewards. During training, the agent interacts with specific tasks M^i sampled from a task distribution $p(\mathcal{M})$, encouraging it to learn representations that capture task-specific information useful for fast adaptation [10].

In the RL^2 framework, the fast adaptation is done by a Recurrent Neural Network (RNN) [40] which processes sequential data and maintains a hidden state $\phi_t = f_\theta(s_t, \phi_{t-1})$ that acts as a compact task embedding. Rather than storing explicit parameters for each task, the RNN dynamically encodes relevant information into its hidden state, making the learning process more efficient and generalizable [41]. Additionally, the use of an RNN allows the policy to tackle POMDP tasks as well as regular MDP tasks. The slow adaptation of the network parameters θ is performed by the underlying reinforcement learning algorithm.

In this hierarchical structure, the adaptation of the hidden state ϕ is treated as a learning process with parameters ϕ. These parameters are used, together with the main network parameters (also called meta-parameters) θ to

generate the action a_t:

$$a_t \sim \pi_\theta(\cdot|s_t, \phi_t).$$

As such, f_θ generates the parameters ϕ of the policy, effectively defining a task-adaptive policy $\pi_{\theta,\phi}$ [9].

Importantly, in RL^2, the sequence of rollouts within the same task is treated as a single continuous sequence, during which the RNN's hidden state is not reset. This design allows the agent to learn about different task transitions. It is also important to mention that the task should be randomly changed within the set of tasks at each environment reset to encourage the policy to generalize robustly.

Unlike standard RL, where the goal is to maximize the expected return over a single episode, the meta-learning objective is to maximize the expected discounted return across multiple episodes sampled from different tasks, effectively learning how to be adaptable. The RNN's hidden state is carried across episodes to retain task information, enabling the policy to adapt its behavior dynamically and perform each new task in a zero-shot manner.

The meta-learning objective can be formally expressed through the following expected return function:

$$J_{RL^2}(\theta) = \mathbb{E}_{M_i \sim p(M)} \left[\mathbb{E}_{\tau \sim p(\tau|M_i,\theta)} \left[\sum_{t=0}^{HT} \gamma^t r_t \right] \right]. \tag{2-19}$$

where each τ is a trajectory of T transitions sampled from MDP M_i using the policy π_θ, and H is the number of episodes in the trial (task horizon).

Algorithm 2: RL^2 for Meta-Reinforcement Learning

Initialize meta-parameters ϕ (RNN and other neural network Parameters)
while *not done* **do**
  Sample tasks $\mathcal{M}^i \sim p(\mathcal{M})$
  for *each task* $\mathcal{M}^i$ **do**
    Initialize RNN hidden state θ_0
    Run a continuous trial consisting of multiple episodes
    for *each timestep* t *in the trial* **do**
      Observe current state s_t, previous action a_{t-1} and reward r_{t-1}
      Update RNN hidden state with input $[s_t, a_{t-1}, r_{t-1}]$ and parameter ϕ:

$$\theta_t = f_\phi(s_t, a_{t-1}, r_{t-1}, \theta_{t-1})$$

      Sample action $a_t \sim \pi_\phi(\cdot \mid s_t, \theta_t)$
      Execute action a_t and receive reward r_t
    End For
  End For
  Update meta-parameters ϕ by optimizing the expected discounted sum of reward across the entire trial
End while

3
Metodology

This chapter describes the methodology adopted to investigate the use of Meta-Reinforcement Learning (Meta-RL) for quadrupedal locomotion in blind scenarios and its ability to enable fast adaptation to previously unseen terrains.

3.1
Robot Description

This research employs the 12-DoF quadrupedal robot ANYmal C, developed by the Swiss company ANYbotics. The platform was primarily chosen for its high onboard computational capabilities, featuring two Intel i7 processors and 16 GB of RAM, which make it well-suited for real-time execution of reinforcement learning-based control pipelines. In addition to its processing power, ANYmal C is equipped with a rich set of sensors, including a LiDAR for environmental perception (e.g., obstacles and terrain profiles), an Inertial Measurement Unit (IMU), and joint encoders that provide detailed proprioceptive feedback. The robotic platform used in this study is illustrated in Figure 3.1.

Figure 3.1: Visualization of ANYmal C on IsaacLab software

ANYmal C features 12 actuated degrees of freedom, distributed symmetrically across its four legs. Each leg consists of three joints—a hip abduction/adduction (HAA) joint enabling lateral leg motion, a hip flexion/extension (HFE) joint controlling forward–backward swing, and a knee flexion/extension (KFE) joint for leg length modulation. This configuration provides the

robot with a highly versatile and adaptive locomotion structure, allowing precise control of foot placement, balance, and body posture across a wide variety of terrains. The DoF configuration is illustrated in Figure 3.2.

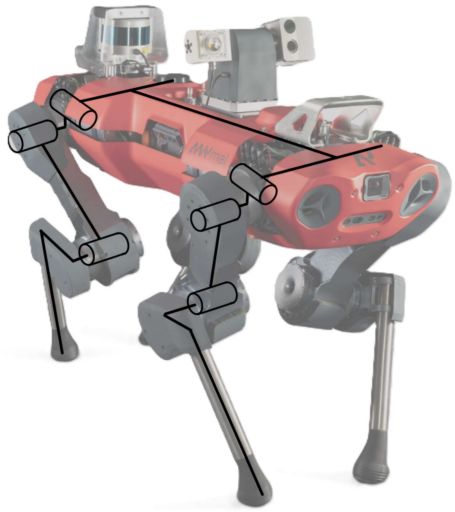

Figure 3.2: Visualization of ANYmal C with his DoF

3.2 Control Architecture

As illustrated in Figure 3.3, the control architecture consists of two hierarchical layers: a high-level RL-based controller and a low-level torque controller. The high-level controller processes observations through an Actor-Critic network. The Actor Layer outputs the desired joint positions, while the Critic approximates the value function V, which is used to compute the Generalized Advantage Estimation (GAE) with the rewards provided by the environment. The GAE serves as the basis for updating the Actor network through batch optimization.

The low-level controller converts the actions chosen by the Actor, together with the current joint states, into torques that are applied to the robot's joints. Following prior work [42], a recurrent neural network (LSTM) is used to model the real robot's actuator dynamics, implicitly encoding temporal dependencies in its hidden state — including delays and non-linearities — and is trained on data collected from the physical actuators.

To enable rapid adaptation to unseen terrains, we developed two RL^2-inspired variants based on Proximal Policy Optimization (PPO). In our architecture, the actor and critic share a common network, as described by Duan et al. [10]. As reported in their work, this adaptation leads to significant improvements in stability and reduces overfitting in the final policy. These

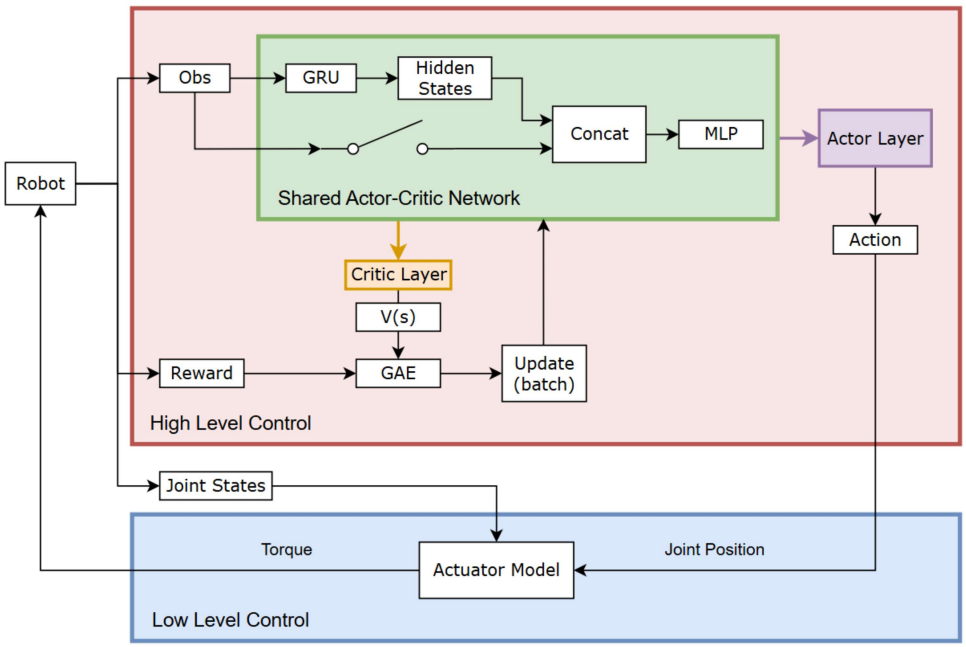

Figure 3.3: Robot control framework used in the simulation

benefits are driven by the regularization effects and better-learned features resulting from weight sharing.

The policy is trained entirely in simulation, primarily across a comprehensive set of flat terrains with varying friction conditions to ensure robustness and rapid adaptation in blind locomotion scenarios. As an additional validation of generalization, we also evaluate performance on procedurally generated rough terrains.

3.3 RL overview

In this chapter, we present the reinforcement learning (RL) fundamentals that underpin the development of the control system. The implementation is based on the SKRL library [43] in Python, which was modified and extended to incorporate the RL^2 algorithm.

3.3.1 Observation Space

The observation space consists primarily of proprioceptive data collected onboard the robot. This data includes the base angular velocity and linear acceleration measured by the IMU, the projected gravity vector that encodes the robot's orientation relative to gravity, and the commanded linear and angular velocities that the robot must track. In addition to these direct measurements, the inertial signals from the IMU (angular velocity and linear

acceleration) can be integrated to aid in estimating the robot's orientation and velocity.

Beyond these inertial cues, joint positions and joint velocities from the 12 actuated joints provide detailed information about the robot's current posture and motion. To complement these proprioceptive signals, a dense height scan from the onboard LiDAR sensor offers exteroceptive information about the surrounding terrain, used exclusively for validation and comparison on unstructured terrain. These features together form a comprehensive observation vector that the high-level policy uses for decision making, as detailed in Table 3.1.

Table 3.1: Active Observation Terms

Active Observation Terms (shape: (235))		
Index	**Name**	**Shape**
0	base lin vel	(3)
1	base ang vel	(3)
2	projected gravity	(3)
3	velocity commands	(3)
4	joint pos	(12)
5	joint vel	(12)
6	actions	(12)
7	height scan	(187)

3.3.2 Action Space

The action space of the agent consists of the target joint positions for the 12 actuated joints of the quadruped. These targets are subsequently converted into torque commands by a low-level position controller employing a learned actuator model [42].

During training, exploration in this action space is achieved through a stochastic policy. Since the robot's action space is continuous rather than discrete — as its actuators are motors that accept real-valued joint position commands — the policy outputs the parameters of a probability distribution, from which the actions are sampled. Specifically, the policy is modeled as a multivariate Gaussian distribution [4] with diagonal covariance:

$$\pi(a \mid s) = \mathcal{N}\Big(\mu_\theta(s), \mathrm{diag}(\sigma_\theta^2(s))\Big), \tag{3-1}$$

where $\mu_\theta(s)$ denotes the mean action (the nominal joint positions predicted by the network given state s), and $\sigma_\theta(s)$ represents the state-dependent standard deviations that control the amount of exploration. The diagonal co-

variance structure ensures that noise is added independently to each joint, enabling efficient and scalable sampling.

This stochastic formulation encourages the agent to balance exploration and exploitation: by sampling around the mean policy, the agent generates diverse behaviors that promote learning, while still converging toward high-reward actions as training progresses.

3.3.3 Reward terms

The main RL objective is to robustly track the commanded linear velocities (in the gravity-aligned frame) and yaw angular velocity (in the world frame) across diverse terrains. This objective is formulated using an exponential decay kernel applied to the velocity tracking error. Accordingly, the reward function is divided into two components: rewards and penalties. This follows the standard reward formulation provided by IsaacLab.

In the reward component, the tracking error for the commanded linear velocity in the xy-plane is defined as

$$r_{lin,xy} = \exp\left(- \frac{||v_{cmd,xy} - v_{body,xy}||^2}{\sigma_{lin}^2} \right), \tag{3-2}$$

where $v_{cmd,xy}$ is the commanded linear velocity on the xy-plane, $v_{body,xy}$ is the measured velocity of the robot's base, and σ_{lin} is a scaling parameter that controls the sensitivity of the exponential decay. This term encourages the robot to match its horizontal base velocity to the desired command.

The tracking error for the yaw angular velocity is expressed as

$$r_{ang,z} = \exp\left(- \frac{(w_{cmd,z} - w_{body,z})^2}{\sigma_{ang}^2} \right), \tag{3-3}$$

where $w_{cmd,z}$ is the commanded yaw rate around the vertical axis, $w_{body,z}$ is the actual yaw angular velocity of the robot, and σ_{ang} is the sensitivity parameter. This term rewards yaw tracking.

Another reward term encourages rhythmic stepping behavior, modeled through the feet air time (r_{feet}).

Penalty terms are introduced to discourage undesirable behaviors. The first one penalizes undesired vertical motion of the base:

$$c_{lin,z} = -v_{body,z}^2, \tag{3-4}$$

where $v_{body,z}$ is the vertical velocity of the robot base.

The second penalty discourages excessive angular velocities in roll and pitch:

$$c_{ang,xy} = -||\omega_{body,xy}||^2, \tag{3-5}$$

where $\omega_{body,xy}$ is the angular velocity vector of the base projected onto the x and y axes.

Joint-related costs are introduced to discourage excessive effort and abrupt dynamics. The first penalizes large torque magnitudes applied at the joints:

$$c_\tau = \sum_{k=1}^{N_{joints}} \tau_k^2, \tag{3-6}$$

while the second penalizes high joint accelerations, encouraging smoother motion:

$$c_{\ddot{q}} = \sum_{k=1}^{N_{joints}} \ddot{q}_k^2, \tag{3-7}$$

where τ_k and $\ddot{q}_k$ denote, respectively, the torque and angular acceleration at joint k, and N_{joints} is the total number of actuated joints. Together, these costs promote energy-efficient locomotion, smoother trajectories, and reduced mechanical stress on the hardware.

Additionally, an action rate penalty is introduced to penalize abrupt changes in consecutive actions, encouraging smoother control signals:

$$c_{act} = \sum_{j=1}^{N_{joints}} (a_j - a_{j-1})^2, \tag{3-8}$$

where a_j is the desired position for joint j at the current timestep, and a_{j-1} is the commanded position at the previous timestep.

Additional, a penalty term can be included for undesired contacts uc, discouraging the robot from touching unintended surfaces and helping to prevent damage or instability.

Following these definitions, our implementation adopts a generic reward formulation expressed as the weighted sum of all individual terms, scaled by the environment time step dt, as shown in Equation (3-9):

$$R_t = \sum_i w_i \, r_t^i \, dt, \tag{3-9}$$

where w_i denotes the weight associated with each reward or penalty term r_t^i.

By combining all active terms, the final reward at each timestep can be expressed as

$$
\begin{aligned}
R_t = \; dt \, (& w_{lin,xy} \, r_{lin,xy} \\
& + w_{ang,z} \, r_{ang,z} \\
& + w_{feet} \, r_{feet} \\
& - w_{lin,z} \, c_{lin,z} \\
& - w_{ang,xy} \, c_{ang,xy} \\
& - w_{\tau} \, c_{\tau} \\
& - w_{\ddot{q}} \, c_{\ddot{q}} \\
& - w_{act} \, c_{act} \\
& - w_{uc} \, uc)
\end{aligned}
\tag{3-10}
$$

where $w_{(\cdot)}$ denotes the weight associated with each term, dt is the environment time step, and the ellipsis indicates that additional terms (e.g., slip penalties or energy costs) may be included depending on the training configuration.

Table 3.2 summarizes the active reward and penalty terms used during training. Positive weights correspond to rewards that are maximized, while negative weights represent penalties (costs) to be minimized. The weight values were adopted from the default locomotion reward implementation provided in IsaacLab, ensuring compatibility and a baseline comparison consistent with prior work in the framework.

Table 3.2: Active Reward Terms

Active Reward Terms		
Index	**Name**	**Weight**
0	track linear veloc. xy (exponential)	1.0
1	track angular veloc. z (exponential)	0.5
2	base linear veloc. z (L2 norm)	-2.0
3	base angular veloc. xy (L2 norm)	-0.05
4	Joint torque (L2 norm)	-1e-05
5	Joint acceleration (L2 norm)	-2.5e-07
6	action rate (L2 norm)	-0.01
7	feet air time	0.125
8	undesired contacts	-1.0

3.3.4 Meta RL algorithm

Focusing on maximizing the reinforcement learning objective—robustly tracking angular and linear velocity commands—it is important to note that,

in this work, the robot relies on proprioceptive information. Consequently, it remains unaware of its surrounding environment.

When applying conventional reinforcement learning methods like Proximal Policy Optimization (PPO) algorithm, a key limitation arises in such proprioceptive-only settings. Since the robot cannot anticipate or perceive the terrain ahead, the learned policies tend to converge to generic locomotion strategies that are sufficiently robust to handle a range of terrains, but often lack the precision, adaptability and efficiency required in novel or rapidly changing conditions. While PPO is effective in stochastic environments—where variability follows known probability distributions—it typically struggles when these distributions shift over time. Examples include sudden terrain transitions or unexpected slippage, both of which demand rapid adaptation and flexible behavior. As a result, policies trained under these constraints are unable to quickly adjust to entirely new or out-of-distribution environments, limiting their generalization capabilities.

To address this limitation, we employ the Fast Reinforcement Learning via Slow Reinforcement Learning (RL^2) algorithm, which enables rapid adaptation to unseen environments by embedding the learning mechanism within the policy itself. Specifically, RL^2 leverages experience collected across multiple tasks during training, allowing the policy to capture temporal patterns and generalize beyond the training distribution. This is achieved through recurrent neural networks (RNNs), which maintain an internal hidden state encoding task-related information inferred from past interactions.

In this work, RL^2 is implemented to enhance proprioceptive locomotion, enabling the policy not only to act within known tasks but also to adapt online to abrupt changes such as terrain transitions or slippage. Our implementation extends the standard PPO algorithm by incorporating recurrent structures.

While conventional PPO typically employs a feedforward multilayer perceptron (MLP) [44], a fully connected, multi-layer neural network with nonlinear activation functions, its operation is purely reactive. In our shared actor-critic architecture, the shared network processes the current observation to produce a common feature representation. This representation is then used independently by the actor and critic heads to compute the action probabilities and state-value estimate, respectively, without any memory of past states. Formally, given an observation vector o_t representing the agent's current state, the MLP computes the actor's output

$$a_t = \pi(\cdot|s_t),$$

where $\pi(\cdot|s_t)$ denotes the network with parameters θ. Each layer trans-

forms its input through a linear combination followed by a nonlinear activation function, propagating information forward through the network. This structure enables the policy to approximate complex, nonlinear mappings between observations and actions.

In contrast, by introducing a recurrent structure before the feedforward MLP, temporal memory is incorporated into the policy. In our implementation, we use a Gated Recurrent Unit (GRU) preceding the MLP, as it can capture long-term dependencies with fewer parameters and lower computational cost than traditional LSTMs, while still retaining the ability to encode task-relevant information [10]. The GRU maintains a hidden state ϕ_t that evolves over time, incorporating information from previous observations. The action is then computed as

$$a_t = \pi_\theta(.|s_t, \phi_t),$$

allowing the policy to condition its decisions on the recent history of states and actions. In our architecture, this GRU-MLP structure is embedded within the Actor-Critic module, as illustrated in Figure 3.3.

In the RL^2 framework, the GRU enables fast adaptation by processing sequential data and maintaining a hidden state that evolves with the agent's interactions. This hidden state acts as an implicit memory that summarizes past observations, actions, and rewards, effectively serving as a compact task embedding. By exploiting this memory, the policy can adapt its behavior online within a given task, without requiring changes to the network parameters. The network itself is trained across tasks so that its recurrent dynamics support rapid within-episode adaptation.

In contrast, the slow adaptation is performed by the PPO algorithm, which updates the network parameters θ gradually across a distribution of tasks and their respective training episodes. Unlike the GRU's fast, within-task adaptation, PPO operates between tasks, integrating experience collected under different transition dynamics and reward structures. Through these incremental parameter updates, PPO endows the policy with general strategies that transfer across tasks. Although this process is comparatively slow, it establishes a robust foundation upon which the GRU can rapidly adapt the agent's behavior to the specifics of a new task.

The separation between fast and slow adaptation is central to the RL^2 framework: while the GRU enables rapid, online adjustment within a single task through its hidden-state dynamics, PPO drives the slow adaptation by optimizing the policy parameters across a distribution of tasks. This synergy equips the agent with both generalization across tasks and flexibility to cope

with abrupt environmental changes [41].

An important aspect of our design is that the hidden state of the GRU is preserved across sequences of rollouts rather than being reset at every episode. This allows the agent to accumulate information over multiple tasks, instead of starting the adaptation process from scratch at the beginning of each episode. As such, it can learn about task transitions.

Typically, task variability in Meta-RL arises from differences in reward functions. In our case, however, the tasks differ in their state transition dynamics $P(M)$. In both scenarios, the agent cannot directly observe the true state s_t. Instead, it receives observations o_t sampled according to the observation function

$$O(o_t \mid \overline{s}_t, a_{t-1}).$$

where $\overline{s} = (M^i, s_t)$.

Crucially, these observations do not include any explicit task identifier. As a result, the policy must rely on the recurrent hidden state to infer the underlying task from past experience and adapt its behavior accordingly, which lies at the core of the learning to learn paradigm of Meta-RL.

3.4 Simulation Environment

In this work, all experiments are conducted in simulation, providing a controlled yet realistic environment to develop and evaluate locomotion policies. The simulation framework is designed to replicate robotic dynamics, sensor feedback, and interaction with diverse terrains, enabling testing of reinforcement learning algorithms without the risks and constraints of physical hardware.

This section first introduces the IsaacLab framework, which serves as the simulation platform, offering high-fidelity physics, GPU-accelerated performance, and integration with reinforcement learning pipelines. We then describe the design and implementation of the simulated terrains used for training and evaluation, highlighting how variations in terrain characteristics challenge the robot's adaptability and test the robustness of the learned policies.

3.4.1 IsaacLab

All experiments were conducted in simulation using the IsaacLab framework [45], an open-source robotics research and reinforcement learning (RL) toolkit built on NVIDIA Isaac Sim. IsaacLab offers a structured Python-based API that streamlines the development and training of locomotion policies, in-

tegrating with popular deep learning (PyTorch) and RL libraries (RLlib). Its architecture supports GPU-accelerated, parallelized simulation, allowing thousands of environment instances to run concurrently on a single GPU. This reduces wall-clock training time and enables efficient hyperparameter tuning and large-scale policy evaluation.

Built on NVIDIA Omniverse, IsaacLab provides high-fidelity physics and sensor simulation, which is critical for modelling realistic terrain interactions and proprioceptive feedback. The framework also includes built-in support for domain randomization—varying physical parameters such as friction, motor strengths, and terrain geometry—which is essential for robust sim-to-real transfer. Furthermore, its modular design allows straightforward integration with ROS 2 and low-level C++ controllers, facilitating eventual deployment on physical hardware.

In this work, IsaacLab enabled fast, reproducible, and scalable training of blind locomotion policies across varied terrains and slip conditions. By leveraging its parallelization capabilities and high-fidelity simulation, we could evaluate multiple policy architectures under consistent conditions, making efficient use of GPU resources while maintaining the physical accuracy required for meaningful locomotion research.

3.4.2 Terrain Description

To evaluate the robot's locomotion performance, two distinct simulated terrain types are considered: a flat terrain with variable slip conditions and a rough terrain composed of heterogeneous sub-terrains. The flat terrain setup isolates locomotion under controlled geometry while introducing variability in ground friction. The rough terrain, on the other hand, provides unstructured environments with different geometric challenges, allowing for the assessment of robustness and generalization. Details of each terrain type are provided in the following subsections.

3.4.2.1 Flat Terrain

The first experimental scenario focuses on evaluating the robot's ability to adapt to flat terrain under variable friction conditions. In this setup, the robot walks on a flat, rigid surface with no geometric irregularities; however, the contact between its feet and the ground is modified through different friction coefficients, adjusted according to the test configuration. Specifically, we use the friction index (FI) to change the slip interaction between the ground

and the robot, where $FI = 0$ corresponds to an extremely low-friction condition (resulting in high slip), whereas $FI = 1$ represents near-zero slip (i.e., high traction). This controlled setup isolates the influence of friction variability, allowing us to directly analyze the policy's capability to maintain stability, balance, and effective locomotion under changing ground contact properties.

To provide a systematic evaluation, six slip configurations are considered. Four correspond to static cases, where the static and dynamic friction coefficients are fixed and equal: $(0.0, 0.0)$, $(0.1, 0.1)$, $(0.5, 0.5)$, and $(1.0, 1.0)$—these conditions will be referred to as *FI* 0.0, *FI* 0.1, *FI* 0.5, and *FI* 1.0. Here, static friction represents the resistance to the initiation of sliding motion, while dynamic friction governs resistance once sliding has begun. In addition, two randomized scenarios are included to introduce non-stationary frictional conditions. In these cases, the static and dynamic slip coefficients are sampled independently, meaning they may assume equal or different values depending on the random draw. In the Random-Reset (RR) configuration, the coefficients are reassigned at every environment reset, which may occur due to timeouts or base-contact failures. In the Random-Interval (RI) configuration, the values vary within the same episode, with changes occurring every 5–10 seconds. These randomized setups are important for testing the policy's adaptability in real-time, as they mimic environments with unpredictable surface properties.

3.4.2.2 Unstructured Terrains

The second scenario evaluates locomotion on rough terrain, designed to validate the policy's robustness and adaptability to unstructured and more challenging conditions. This environment combines four heterogeneous subterrains: (i) box-shaped obstacles, which test the agent's ability to step over discontinuities and maintain balance after abrupt height changes; (ii) randomly generated rough surfaces, introducing stochastic variations in elevation that emulate natural uneven ground; (iii) pyramid slopes, requiring the robot to ascend and descend inclined planes while maintaining stability; and (iv) inverse pyramid slopes, where the agent must negotiate concave depressions that challenge foothold selection and recovery strategies. By exposing the policy to this diverse set of geometric features, the rough terrain scenario aims to evaluate not only basic locomotion, but also the capability to generalize and adapt to unpredictable, real-world-like environments.

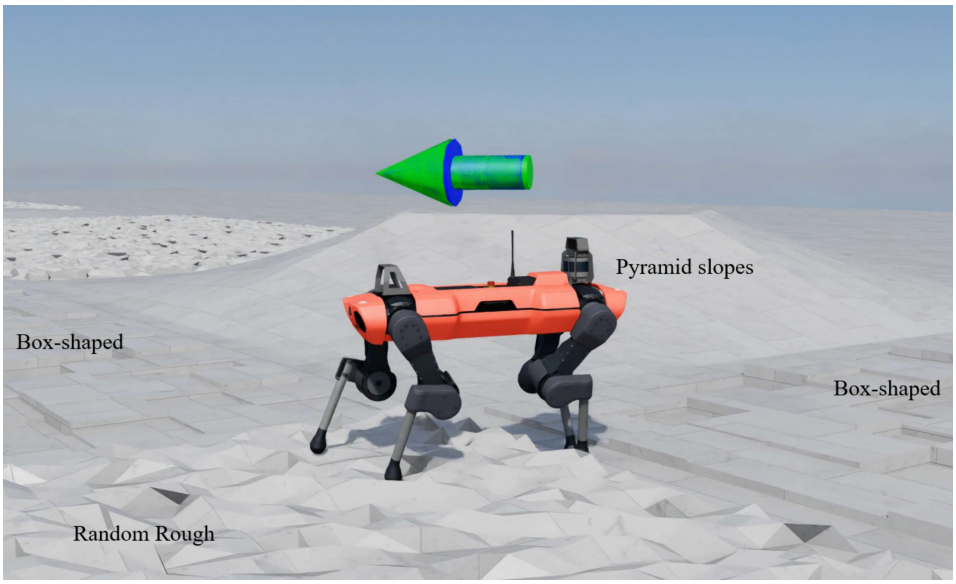

Figure 3.4: Compound terrain composed of four sub-terrain types

3.5 Training and Simulation

To achieve fast adaptation to unseen terrains, we designed two RL^2-inspired variants based on the Proximal Policy Optimization (PPO) algorithm.

In the first variant, called PPO-RNN-Hidden, the MLP receives exclusively the output of the GRU hidden state, which itself is updated solely based on the sequence of past observations. In this architecture, the policy selects actions directly from this hidden representation. Consequently, the hidden state must effectively encode both the robot's proprioceptive information and the learned patterns shared across multiple tasks to capture the environment's dynamics.

The second variant, named PPO-RNN-Concat, concatenates the GRU hidden states with the current observation vector before passing it to the subsequent MLP. In this architecture, the hidden state encodes longer-term features—such as terrain properties—while the immediate observations contribute fine-grained proprioceptive and sensory data. This combination enriches context awareness and promotes adaptive behavior across diverse environments.

Figure 3.3 illustrates the shared actor-critic architecture underlying both variants. In PPO-RNN-Hidden, the gating switch is open, allowing only the GRU hidden state to propagate forward. In contrast, PPO-RNN-Concat closes the switch, enabling the concatenation of the hidden state with the current observation. This architectural choice defines whether the policy and value

networks operate solely on temporally aggregated information or on a joint representation that also incorporates the most recent sensory inputs.

These two architectures were trained and evaluated entirely in simulation using the IsaacLab framework, running on a GPU (NVIDIA RTX 4090). The training leveraged 4096 parallel simulated environments to accelerate data collection and policy optimization. To ensure statistical reliability, each algorithm was run with 10 different seeds.

In our implementation, the recurrent module is embedded within the Actor-Critic framework, where the GRU output is passed through two fully connected layers with 256 and 128 units, respectively. These layers transform the temporal features into policy and value predictions, improving robustness to abrupt terrain changes and external disturbances by allowing the policy to infer hidden dynamics and adapt accordingly.

To ensure a fair comparison, the same hyperparameters were used for PPO and the PPO-RNN variants, except for the GRU-specific parameters: hidden size, number of layers, and sequence length. These were selected through a systematic grid search conducted on the "random per reset" flat terrain scenario—a setting aligned with the original RL^2 methodology, in which environment properties vary randomly at each reset. The search covered GRU configurations with 1–2 layers, hidden sizes from 8 to 128, and sequence lengths of 64, 128, and 256.

Each candidate configuration was evaluated over five independent random seeds, with a single seed requiring approximately 40 minutes of wall-clock time to complete. Given the size of the hyperparameter space, this procedure represented a substantial computational effort, but it was essential to identify a recurrent architecture that balanced representational capacity with training stability for the blind locomotion task.

The performance was then assessed based on training stability and average reward across runs. Using graphical analysis and a detailed evaluation of the learning curves, we identified the configurations that consistently achieved higher rewards with reduced variability. For each configuration, we extracted the maximum reward from the training curves across five seeds, computed the mean and the 95% confidence interval, and identified the four most promising setups. These are summarized in Table 3.3 for the PPO-RNN-Concat algorithm and in Table 3.4 for the PPO-RNN-Hidden algorithm. It is important to note that, under the simulated conditions, the maximum achievable average total reward per episode is approximately 25, which serves as a useful reference to contextualize the observed scores.

Table 3.3: Most promising configurations - Concat

Most promising configurations - Concat				
Concat	2c-16h-256s	1c-48h-64s	2c-8h-256s	2c-8h-128s
Mean ± CI	22.65 ± 0.10	22.68 ± 0.21	22.43 ± 0.16	22.28 ± 0.22

Table 3.4: Most promising configurations - Hidden

Most promising configurations - Hidden				
Hidden	1c-112h-128s	1c-80h-128s	1c-48h-64s	1c-128h-256s
Mean ± CI	12.43 ± 0.83	16.33 ± 1.45	20.19 ± 2.52	13.20 ± 1.03

These candidate configurations were then further validated by training an additional set of five seeds, which yielded the final results. From this process, a single GRU configuration was selected for each variant (PPO-RNN-Hidden and PPO-RNN-Concat), and these settings were subsequently applied across all experiments and terrains to ensure comparability. Given that each training seed required approximately 40 minutes of wall-clock time, the complete hyperparameter search and validation process demanded roughly 400 hours of total simulation time. The selected GRU configurations for each PPO-RNN variant are listed in Table 3.5.

Table 3.5: Selected GRU Configurations

Selected GRU Configurations			
Algorithm	**Layers**	**Hidden Size**	**Sequence Length**
PPO-RNN-Concat	2	8	128
PPO-RNN-Hidden	1	48	64

Both algorithms—baseline PPO and the two recurrent variants—were evaluated under the two terrain setups introduced in Section 3.4.2.

In the Flat Terrain setup (Section 3.4.2.1), each algorithm was independently trained under six different slip conditions and subsequently evaluated across the entire set of test scenarios. This cross-evaluation procedure enables not only the assessment of in-distribution performance—when training and testing conditions match—but also a deeper analysis of the agents' generalization capabilities under unseen slip dynamics. Such testing is particularly important for verifying whether the learned policies are robust and transferable to environments with frictional properties not directly encountered during training.

In contrast, the Unstructured Terrains setup (Section 3.4.2.2) focused exclusively on in-distribution performance. During training, the policy is ex-

posed to a compound terrain that includes all four sub-terrains simultaneously, encouraging generalization across diverse challenges. For evaluation, the policy is tested both on each sub-terrain individually and on the combined rough terrain, enabling a detailed analysis of its ability to adapt to different sources of irregularity and to generalize across varying topographies.

In all experiments—including the baseline PPO—the observation space excluded LiDAR data, relying solely on proprioceptive and sensory signals. To isolate the impact of explicit terrain perception, we conducted an additional study on the rough terrain scenario, comparing the baseline PPO (both with and without LiDAR) against PPO-RNN-Concat. The LiDAR-equipped variant will be referred to as PPO-Sensor. This setup allows us to directly contrast implicit terrain encoding via recurrence with explicit terrain perception provided by external sensing.

4
Results

To evaluate the proposed recurrent architectures, we compared PPO, PPO-RNN-Concat, and PPO-RNN-Hidden across different terrain configurations. Tables 4.1 and 4.8 summarize the Total Reward (mean) achieved by each algorithm during the test phase, along with the corresponding 95% confidence intervals (mean $\pm$ CI) computed over 10 seeds. In these tables, bold values indicate the best performance in each row (i.e., higher mean with non-overlapping confidence intervals), while values marked with a star denote results statistically equivalent to the best in that row.

Under the simulated conditions, the maximum achievable average total reward per episode is approximately 25. A video illustrating the simulation results is available at: `https://youtu.be/W0iaRkEa0sQ.`. This reference value provides context for the scores reported, helping to gauge how close each algorithm came to the performance ceiling of the task.

4.1
Flat Terrain

This section presents the results obtained on flat terrain. The analysis is organized into subsections according to the slip condition considered during evaluation, ranging from fixed values (0.0, 0.1, 0.5, and 1.0) to randomized regimes (RR and RI).

Table 4.1 summarizes the results obtained across all algorithms and training configurations, reporting mean total reward and confidence intervals for each slip condition evaluated.

Table 4.1: Simulation results on flat terrain. Each group labeled train X.X refers to the slip condition used during training

train	**FI**	**PPO**	**PPO-Concat**	**PPO-Hidden**
train 0.0	0.0	3.31 ± 0.29	3.38 ± 0.28*	1.69 ± 0.50
	0.1	4.74 ± 0.23	4.74 ± 0.42*	2.65 ± 0.65
	0.5	2.73 ± 0.79	2.37 ± 0.77*	2.76 ± 0.62*
	1.0	1.86 ± 1.46	1.86 ± 0.91*	**3.51 ± 0.65**
	RR	2.42 ± 0.88	2.16 ± 0.72*	2.68 ± 0.58*
	RI	2.35 ± 0.81	1.99 ± 0.76*	2.23 ± 0.57*
train 0.1	0.0	0.05 ± 0.22	0.65 ± 0.53	**1.50 ± 0.28**
	0.1	7.03 ± 1.06	**22.33 ± 0.22**	6.03 ± 0.82*
	0.5	3.52 ± 0.60	4.38 ± 1.73	**5.54 ± 1.04**
	1.0	2.08 ± 0.53	1.90 ± 1.44*	**5.35 ± 0.96**
	RR	2.93 ± 0.48	4.26 ± 1.43	**5.34 ± 0.96**
	RI	2.88 ± 0.50	3.71 ± 1.47*	**5.29 ± 0.96**
train 0.5	0.0	-2.30 ± 0.62	-2.36 ± 0.72*	-2.51 ± 0.90*
	0.1	2.06 ± 1.07	1.85 ± 0.77*	1.50 ± 0.59*
	0.5	24.08 ± 0.35	**24.73 ± 0.19**	23.52 ± 1.17
	RR	15.40 ± 2.03	16.49 ± 1.23*	14.47 ± 2.05
	RR	18.50 ± 0.84	**19.33 ± 0.48**	17.90 ± 0.99
	RI	17.59 ± 0.85	**18.34 ± 0.56**	17.14 ± 0.95*
train 1.0	0.0	-5.60 ± 1.15	-19.76 ± 3.03	-25.85 ± 3.73
	0.1	-8.16 ± 1.83	-21.52 ± 2.98	-26.69 ± 3.33
	0.5	2.66 ± 3.13	0.13 ± 7.40	-4.13 ± 4.11
	1.0	24.09 ± 0.22	**24.81 ± 0.13**	24.17 ± 0.55*
	RR	4.99 ± 1.68	4.01 ± 3.83*	1.34 ± 2.42
	RI	2.00 ± 1.54	2.63 ± 3.96*	0.42 ± 2.69
train RR	0.0	-0.60 ± 0.22	-0.86 ± 0.41*	-0.61 ± 0.20*
	0.1	6.27 ± 2.12	**16.10 ± 1.51**	6.93 ± 2.44*
	0.5	15.40 ± 5.36	**25.07 ± 0.21**	16.36 ± 4.44*
	1.0	14.48 ± 4.84	**24.24 ± 0.42**	13.45 ± 4.27*
	RR	14.55 ± 5.22	**24.56 ± 0.19**	14.86 ± 4.26*
	RI	14.41 ± 5.23	**24.57 ± 0.21**	14.68 ± 4.38*
train RI	0.0	-0.59 ± 0.15	-1.33 ± 0.65	-0.02 ± 0.16
	0.1	13.12 ± 1.75	**16.40 ± 1.26**	14.38 ± 1.49*
	0.5	23.78 ± 0.72	**25.04 ± 0.21**	24.15 ± 0.53*
	1.0	22.70 ± 1.11	**24.53 ± 0.29**	22.94 ± 0.91*
	RR	23.15 ± 0.87	**24.55 ± 0.23**	23.51 ± 0.59*
	RI	23.08 ± 0.84	**24.60 ± 0.23**	23.47 ± 0.59*

4.1.1 Training with FI = 0.0: No Friction

Table 4.2: Simulation results on flat terrain - training FI = 0.0

FI	**PPO**	**PPO-Concat**	vs. PPO	**PPO-Hidden**	vs. PPO
0.0	3.31 ± 0.29	3.38 ± 0.28*	+2.1%	1.69 ± 0.50	-49.0%
0.1	4.74 ± 0.23	4.74 ± 0.42*	0.0%	2.65 ± 0.65	-44.1%
0.5	2.73 ± 0.79	2.37 ± 0.77*	-13.2%	2.76 ± 0.62*	+1.1%
1.0	1.86 ± 1.46	1.86 ± 0.91*	+0.1%	**3.51 ± 0.65**	+89.0%
RR	2.42 ± 0.88	2.16 ± 0.72*	-10.8%	2.68 ± 0.58*	10.8%
RI	2.35 ± 0.81	1.99 ± 0.76*	-15.4%	2.23 ± 0.57*	-5.4%

Starting with flat terrain results 4.2, we observe that under total slip, both PPO-RNN-Concat and PPO-RNN-Hidden performed similarly to the baseline PPO. Across all algorithms, the results show only minor gains or losses, and overall performance remains limited.

In this condition, the robot is physically unable to generate sufficient traction to move efficiently, regardless of the control policy. As a result, none of the algorithms were able to achieve meaningful performance. This experiment, however, is important because it illustrates what happens when training occurs under conditions in which locomotion is fundamentally impossible: the policy learned in such settings not only performs poorly during training but also fails to transfer to other terrain configurations.

Even with recurrent architectures such as PPO-RNN-Concat, which provide access to both the current observations and the history of past interactions, the embedded memory is unable to compensate for the lack of useful training data. Temporal information alone cannot replace the need for a policy grounded in effective experience. In other words, rapid adaptation through memory mechanisms acts only as a fine-tuning process of an already viable policy; it cannot transform an ineffective one into an efficient solution.

Consequently, when applying a policy learned in the no-friction setting to other terrains (e.g., FI = 1.0, where locomotion is easier), the robot still fails to achieve satisfactory performance. The reason is straightforward: since the agent was unable to learn to walk during training, it cannot succeed when tested under different conditions.

Figure 4.1 shows the mean learning curves across 10 random seeds for each algorithm. Because the plot represents the average trajectory of training, it highlights that none of the algorithms were able to achieve consistent improvements: the agent remained stuck at very low reward levels throughout

training. In other words, the lack of friction not only prevented the emergence of effective locomotion in isolated cases, but systematically limited all training runs, demonstrating that the environment provides no useful signal to support learning. Thus, the curve confirms that the algorithms, regardless of their architecture, were incapable of learning under these conditions.

From a practical perspective, each complete training run for a single seed demanded approximately 40 minutes of wall-clock time on our hardware setup. This duration was consistent across both the baseline PPO and the PPO-RNN-Concat variant, indicating that the recurrent overhead was negligible.

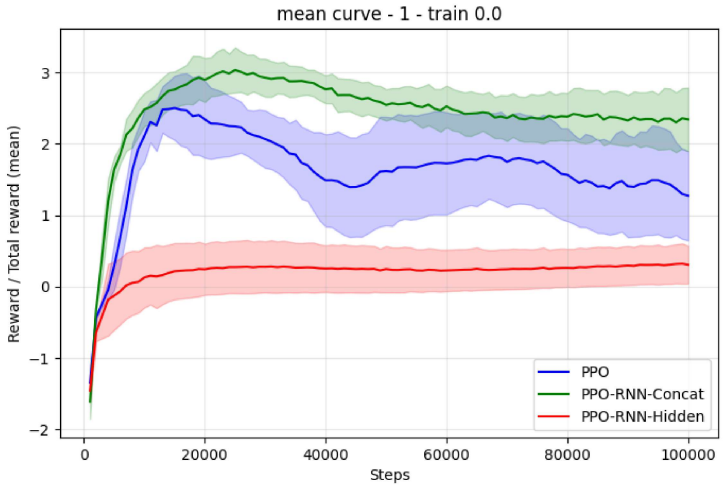

Figure 4.1: Mean training curves across 10 random seeds for each algorithm under FI = 0.0. The shaded regions represent 95% confidence intervals across seeds

4.1.2 Training with FI = 0.1: Low Friction / Fast Adaptation

Table 4.3: Simulation results on flat terrain - training FI = 0.1

FI	PPO	PPO-Concat	vs. PPO	PPO-Hidden	vs. PPO
0.0	0.05 ± 0.22	0.65 ± 0.53	+1220%	**1.50 ± 0.28**	+2944%
0.1	7.03 ± 1.06	**22.33 ± 0.22**	+217.8%	6.03 ± 0.82*	-14.2%
0.5	3.52 ± 0.60	4.38 ± 1.73	+24.5%	**5.54 ± 1.04**	+57.5
1.0	2.08 ± 0.53	1.90 ± 1.44*	-8.3%	**5.35 ± 0.96**	+157.8
RR	2.93 ± 0.48	4.26 ± 1.43	+45.1%	**5.34 ± 0.96**	+82.2%
RI	2.88 ± 0.50	3.71 ± 1.47*	28.8%	**5.29 ± 0.96**	+83.5%

Under very low friction conditions 4.3, PPO-RNN-Concat achieved significant improvements over PPO, often reaching much higher average returns

and narrower confidence intervals, particularly when tested on terrains with FI = 0.1. PPO-RNN-Hidden also outperformed PPO in several scenarios, though the improvements were generally smaller, less consistent, and statistically insignificant.

This configuration is especially relevant because it demands fast adaptation from the robot. With such low friction, the agent has only a few seconds to adjust its behavior before losing balance or falling. For conventional algorithms like PPO, which lack an explicit mechanism for rapid adjustment, this represents a major challenge, as the policy cannot adapt online to these sudden conditions. Figure 4.2 illustrates this effect for PPO algorithm under a friction index of $FI = 0.1$.

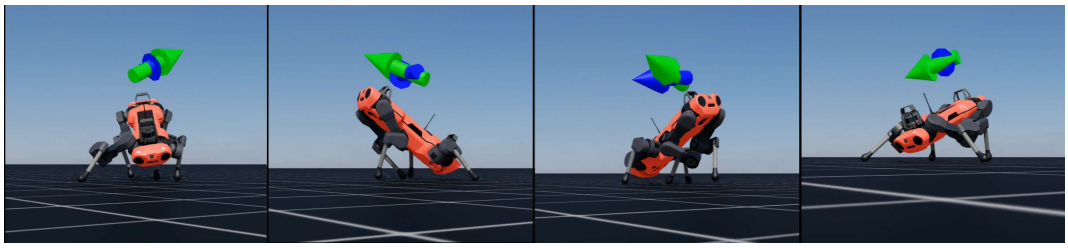

Figure 4.2: Robot behavior under low-friction conditions ($FI = 0.1$). Using the PPO algorithm, the sequence shows representative failure modes such as lateral slipping, imbalance during stance transitions, excessive body roll, and foreleg collapse, emphasizing the need for rapid online adaptation.

The RL^2 algorithms, on the other hand, incorporate mechanisms for rapid online adaptation, in this case implemented using recurrent neural networks. In architectures like PPO-RNN-Concat, the agent use leverage information from past interactions to modify its actions in real time. This allows the robot to detect subtle changes in its environment, such as slight slips or loss of traction, and adjust its behavior accordingly, improving stability and overall performance.

The FI = 0.1 scenario clearly illustrates the advantage of memory-based policies. While a standard PPO agent struggles to cope with the sudden loss of traction, the recurrent Meta-RL agent can rapidly detect changes in the environment by recognizing patterns learned from past experiences. Using this temporal information, it can adjust its actions in real time, effectively stabilizing locomotion. Figure 4.3 illustrates this effect for PPO-RNN-Concat algorithm under a friction index of $FI = 0.1$.

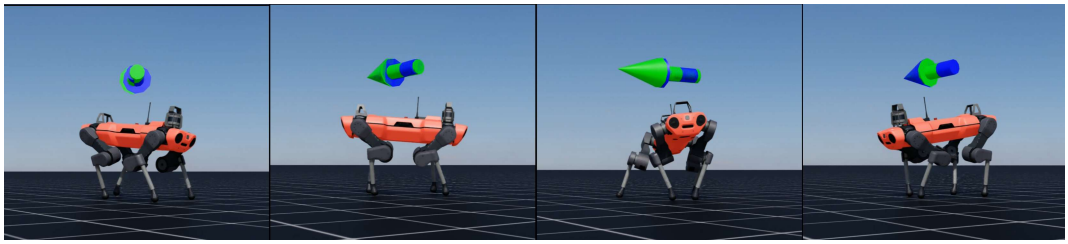

Figure 4.3: Robot behavior under low-friction conditions ($FI = 0.1$). While the standard PPO policy exhibits instability—manifested as slipping, body roll, and loss of posture—the PPO-RNN-Concat architecture rapidly identifies the decrease in traction and adapts its actions online, maintaining stability despite the sudden change in ground friction.

When the policy is trained under the FI = 0.1 condition and tested in the same setting, PPO-RNN-Concat achieved a 217.8% improvement in performance over conventional PPO, highlighting the benefits of recurrent architectures. This demonstrates the crucial role of memory-based policies in scenarios where stability depends on split-second behavioral adaptations.

Note, however, that the trained policy did not generalize to the other test scenarios, since they were not in the training distribution as is required for Meta-RL. As such, the increased performance should be attributed to using a recurrent network, instead of to using a Meta-RL algorithm.

Figure 4.4 shows the mean learning curves across 10 seeds for each algorithm. The plot makes it clear that PPO-RNN-Concat not only reached much higher reward levels but also maintained consistent growth throughout training, while PPO quickly plateaued and PPO-RNN-Hidden exhibited only modest improvements. The narrow shaded region around the green curve also confirms the stability of PPO-RNN-Concat across seeds, reinforcing the statistical results shown in Table 4.3. In this case, training required a longer horizon of 250k timesteps—compared to 100k in the other scenarios—due to the increased difficulty of the environment. Since the agent needed to gather more information to adapt to the challenging slip condition, learning progressed at a slower pace and convergence took considerably longer.

From a practical perspective, each complete training run for a single seed demanded approximately 120 minutes of wall-clock time on our hardware setup. This duration was consistent across both the baseline PPO and the PPO-RNN-Concat variant, indicating that the recurrent overhead was negligible.

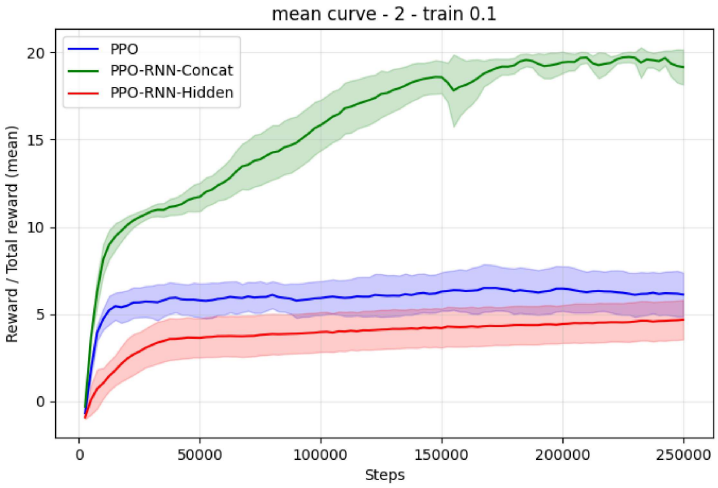

Figure 4.4: Mean training curves across 10 random seeds for each algorithm under FI = 0.1. The shaded regions represent 95% confidence intervals across seeds

4.1.3 Training with FI = 0.5 and 1.0: Moderate and High Friction

Table 4.4: Simulation results on flat terrain - training FI = 0.5

FI	PPO	PPO-Concat	vs. PPO	PPO-Hidden	vs. PPO
0.0	-2.30 ± 0.62	-2.36 ± 0.72*	-2.7%	-2.51 ± 0.90*	-9.1%
0.1	2.06 ± 1.07	1.85 ± 0.77*	-9.9%	1.50 ± 0.59*	-26.9%
0.5	24.08 ± 0.35	**24.73 ± 0.19**	+2.7%	23.52 ± 1.17	-2.3%
1.0	15.40 ± 2.03	16.49 ± 1.23*	+7.0%	14.47 ± 2.05	-6.1%
RR	18.50 ± 0.84	**19.33 ± 0.48**	+4.5%	17.90 ± 0.99	-3.2%
RI	17.59 ± 0.85	**18.34 ± 0.56**	+4.3%	17.14 ± 0.95*	-2.6%

Table 4.5: Simulation results on flat terrain - training FI = 1.0

FI	PPO	PPO-Concat	vs. PPO	PPO-Hidden	vs. PPO
0.0	-5.60 ± 1.15	-19.76 ± 3.03	-253.0%	-25.85 ± 3.73	-362.0%
0.1	-8.16 ± 1.83	-21.52 ± 2.98	-163.6%	-26.69 ± 3.33	-227.1%
0.5	2.66 ± 3.13	0.13 ± 7.40	-95.2%	-4.13 ± 4.11	-255.6%
1.0	24.09 ± 0.22	**24.81 ± 0.13**	+3.0%	24.17 ± 0.55*	+0.3%
RR	4.99 ± 1.68	4.01 ± 3.83*	-19.7%	1.34 ± 2.42	-73.2%
RI	2.00 ± 1.54	2.63 ± 3.96*	+31.8%	0.42 ± 2.69	-79.2%

At higher friction levels, FI = 0.5 (Table 4.4) and FI = 1.0 (Table 4.5), the performance of PPO-RNN-Concat became more variable when tested on

different slip conditions: in some test cases, it maintained modest gains in confidence intervals and reward returns over PPO, while in others, it suffered noticeable drops. PPO-RNN-Hidden, meanwhile, tended to remain close to the baseline PPO, without clear improvement or degradation. When training on single tasks, no method consistently outperformed the others, as expected.

In these conditions, all algorithms are capable of achieving reasonable performance. This is because locomotion under moderate and high friction is easier to learn: the changes in terrain dynamics due to slip are less abrupt, and thus a standard PPO agent is already sufficient to develop stable locomotion strategies. The ability to rely on a relatively consistent interaction with the ground reduces the need for fast adaptation mechanisms.

Nevertheless, the recurrent architectures still provide some advantages. Even though PPO can perform well under its training condition, PPO-RNN-Concat often achieves narrower confidence intervals, suggesting that recurrent adaptation contributes to greater stability and consistency during training. This effect is more noticeable when training at FI = 0.5, where PPO-RNN-Concat not only improves stability in its own training condition but also transfers moderately well to nearby settings such as FI = 1.0, Random Reset (RR), and Random Interval (RI). However, this transfer does not extend to harsher scenarios with lower slip values (e.g., FI = 0.1), where the absence of traction demands rapid adaptation, requiring specific training to achieve such responsiveness.

The case of FI = 1.0, on the other hand, reveals a different trend. While PPO-RNN-Concat maintains reasonable performance in the training condition, its ability to generalize is weaker. Policies trained at high friction tend to overfit to the easier locomotion dynamics, resulting in poor transfer to other terrains. This highlights that while high friction enables efficient training, it does not necessarily promote robustness across diverse slip conditions.

In summary, the FI = 0.5 condition emerges as an effective middle ground for training: it provides enough friction for the agent to learn stable policies while still exposing the network to variability that encourages moderate transfer to nearby medium- and high-friction conditions. However, this advantage does not extend to low-friction scenarios such as FI = 0.1 or 0.0, where performance remain negligible. FI = 1.0, by contrast, yields strong performance in isolation but fails to generalize, while PPO-RNN-Hidden remains mostly aligned with PPO, showing limited adaptation capability.

Figure 4.7 shows the mean learning curves across 10 random seeds for each algorithm when training with FI = 0.5 and FI = 1.0. Unlike the no-friction case, these plots clearly illustrate that all algorithms were able

to improve over time, converging to high reward levels. This confirms that under favorable conditions of traction, the agent can learn effective locomotion strategies. Recurrent architectures, particularly PPO-RNN-Concat, tend to reduce variance across seeds, providing more stable convergence patterns.

From a practical perspective, each complete training run for a single seed demanded approximately 40 minutes of wall-clock time on our hardware setup. This duration was consistent across both the baseline PPO and the PPO-RNN-Concat variant, indicating that the recurrent overhead was negligible.

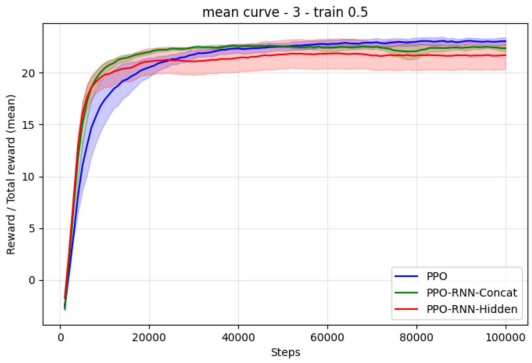

Figure 4.5: FI = 0.5

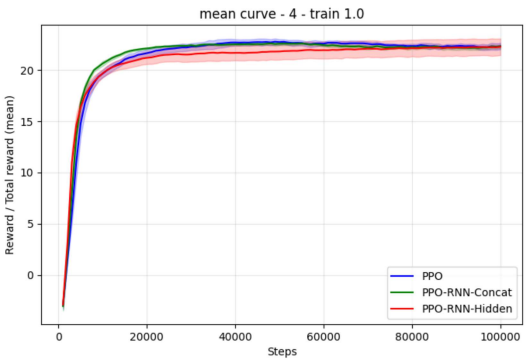

Figure 4.6: FI = 1.0

Figure 4.7: Mean training curves across 10 random seeds for each algorithm under different friction conditions. The shaded regions represent 95% confidence intervals across seeds

4.1.4 Training with Randomized and Variable Slip (RR / RI)

Table 4.6: Simulation results on flat terrain - training FI = RR

FI	PPO	PPO-Concat	vs.PPO	PPO-Hidden	vs.PPO
0.0	-0.60 ± 0.22	-0.86 ± 0.41*	-43.8%	-0.61 ± 0.20*	-1.3%
0.1	6.27 ± 2.12	**16.10 ± 1.51**	+156.8%	6.93 ± 2.44*	+10.5%
0.5	15.40 ± 5.36	**25.07 ± 0.21**	+62.8%	16.36 ± 4.44*	+6.3%
1.0	14.48 ± 4.84	**24.24 ± 0.42**	+67.4%	13.45 ± 4.27*	-7.1%
RR	14.55 ± 5.22	**24.56 ± 0.19**	+68.8%	14.86 ± 4.26*	+2.1%
RI	14.41 ± 5.23	**24.57 ± 0.21**	+70.5%	14.68 ± 4.38*	+1.9%

Table 4.7: Simulation results on flat terrain - training FI = RI

FI	**PPO**	**PPO-Concat**	vs.PPO	**PPO-Hidden**	vs.PPO
0.0	-0.59 ± 0.15	-1.33 ± 0.65	-123.2%	-0.02 ± 0.16	+97.3%
0.1	13.12 ± 1.75	**16.40 ± 1.26**	+25.0%	14.38 ± 1.49*	+9.6%
0.5	23.78 ± 0.72	**25.04 ± 0.21**	+5.3%	24.15 ± 0.53*	+1.5%
1.0	22.70 ± 1.11	**24.53 ± 0.29**	+8.0%	22.94 ± 0.91*	+1.0%
RR	23.15 ± 0.87	**24.55 ± 0.23**	+6.1%	23.51 ± 0.59*	+1.6%
RI	23.08 ± 0.84	**24.60 ± 0.23**	+6.6%	23.47 ± 0.59*	+1.7%

When trained under randomized friction conditions—specifically, random resets (RR) 4.6, which are the proper conditions required for RL^2, or random intervals (RI) 4.7, PPO-RNN-Concat achieved consistent, generalizable performance with substantial gains over standard PPO, with improvements in average total reward ranging from approximately 62% to 70%. PPO-RNN-Hidden, by comparison, generally performed similarly to PPO under these randomized scenarios.

A key distinction between RR and RI helps explain the relative performance of the algorithms. In RR, friction changes occur exclusively between episodes and thus are not part of the distribution experienced within a single episode during training. Consequently, standard PPO cannot learn this dynamic: Since the task changes every time the environment resets, the PPO baseline struggles to learn a single policy capable of solving all tasks. The absence of memory prevents PPO from transferring knowledge across episodes, which explains its lower performance under RR. By contrast, in RI the friction condition changes within the same episode. This means the distribution of friction transitions is directly present during training, enabling the agent to observe and react to these variations as part of its ongoing interaction. Figure 4.8 illustrates this effect for PPO algorithm under a friction index of $FI = RR$.

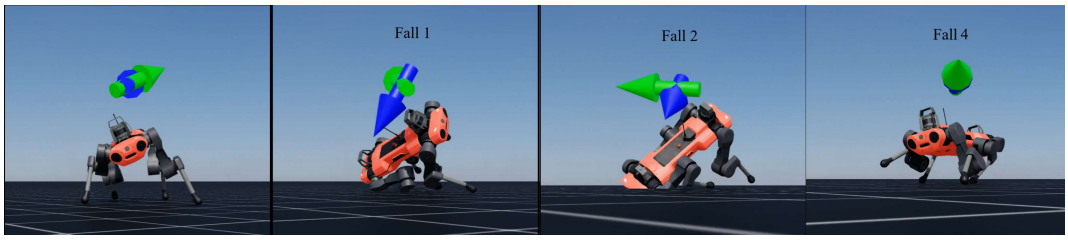

Figure 4.8: Robot behavior under Random-Reset (FI = RR) friction conditions. Using the PPO algorithm, the robot fails to adapt to the abrupt friction changes occurring between episodes, resulting in characteristic failure modes such as lateral slipping, imbalance during stance transitions, excessive body roll, and foreleg collapse. These behaviors highlight the difficulty faced by memoryless policies when the task changes only at episode resets.

This distinction clarifies why recurrent variants, and especially PPO-RNN-Concat, perform better in both scenarios. The recurrent architecture can encode the history of interactions, retaining information about previously encountered slip values. This embedded memory allows the policy to adapt more effectively to variable friction, whether the changes occur across episodes (RR) or within them (RI). PPO-RNN-Hidden, meanwhile, seems less capable of fully exploiting this sequential context, which explains its lower and less consistent performance compared to the Concat variant.

Beyond achieving higher average returns, PPO-RNN-Concat consistently exhibited narrower confidence intervals, suggesting greater stability and reduced variance across random seeds. This robustness is especially evident when training and evaluation occur under the same friction regime, reinforcing that the recurrent design contributes not only to better performance but also to more predictable and reliable behavior. Figure 4.9 illustrates this effect for PPO-RNN-Concat algorithm under a friction index of $FI = RR$.

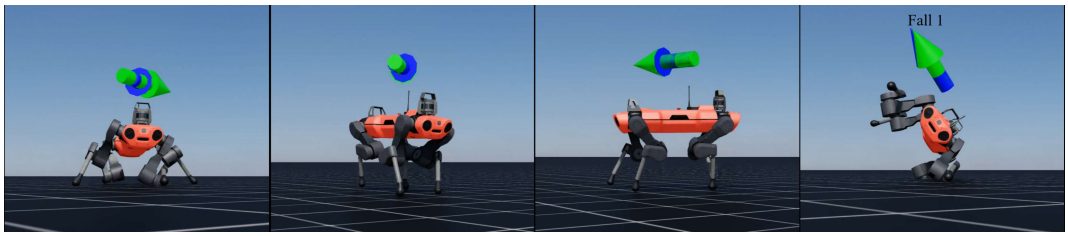

Figure 4.9: Robot behavior under Random-Reset (FI = RR) friction conditions. Using the PPO-RNN-Concat algorithm, the robot adapts to friction changes occurring between episodes by leveraging its recurrent memory, maintaining stable locomotion and avoiding failures such as slipping, imbalance, body roll, and foreleg collapse.

Figure 4.12 shows the mean learning curves across 10 random seeds for each algorithm under the randomized friction conditions (RR and RI). In

the RR setting, PPO-RNN-Concat clearly outperforms both PPO and PPO-RNN-Hidden, achieving substantially higher rewards and converging faster, while also maintaining narrow confidence intervals. PPO-RNN-Hidden shows moderate improvements over PPO in the initial stages but saturates at lower performance levels. In contrast, under the RI condition, all algorithms reach similar asymptotic performance, but PPO-RNN-Concat exhibits faster convergence in the early stages of training, highlighting the benefits of recurrence when dealing with within-episode variations. Taken together, these results indicate that recurrence is especially advantageous in environments with high non-stationarity, such as RR, while in milder randomized conditions like RI the benefits are less pronounced but still provide faster adaptation.

From a practical perspective, each complete training run for a single seed demanded approximately 40 minutes of wall-clock time on our hardware setup. This duration was consistent across both the baseline PPO and the PPO-RNN-Concat variant, indicating that the recurrent overhead was negligible.

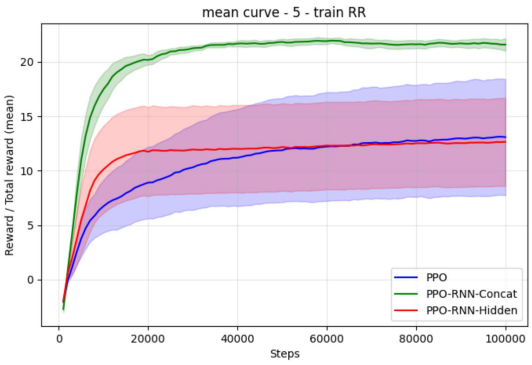

Figure 4.10: FI = RR

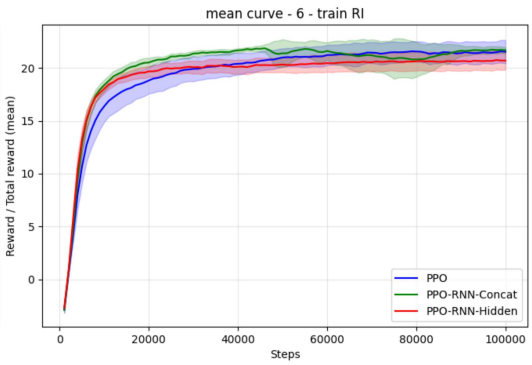

Figure 4.11: FI = RI

Figure 4.12: Mean training curves across 10 random seeds for each algorithm under different friction conditions. The shaded regions represent 95% confidence intervals across seeds

4.2 Training on Rough Terrain

Table 4.8: Overview of the simulation outcomes for rough terrain

Terrain	PPO	PPO-RNN-Concat	PPO-Sensor
Mixed Terrain	16.14 ± 1.60	18.61 ± 0.90	**20.09 ± 2.96**
Boxes	5.73 ± 0.63	7.35 ± 0.17	**8.96 ± 1.62**
Pyramid slope	18.54 ± 1.30	21.51 ± 0.55	**22.03 ± 2.54**
Pyramid slope inv.	18.31 ± 1.25	21.23 ± 0.59	**21.97 ± 2.67**
Random rough	12.23 ± 1.96	14.85 ± 1.05	**16.74 ± 3.24**

Table 4.9: Relative improvements (%) over PPO for rough terrain

Terrain	**PPO-Concat vs. PPO**	**PPO-Sensor vs. PPO**
Mixed Terrain	15.3%	24.5%
Boxes	28.2%	56.2%
Pyramid slope	16.1%	18.8%
Pyramid slope inv.	15.9%	20.0%
Random rough	21.4%	36.9%

Table 4.8 reports the evaluation results on distinct, unseen rough terrains, each introducing heterogeneous challenges such as irregular obstacles, abrupt slopes, and varying elevations. Complementarily, while Table 4.9 highlights the relative improvements (%) achieved under these conditions. Together, these scenarios not only assess the locomotion capability of the policies but also reveal their adaptability to unpredictable dynamics beyond the training distribution.

We compare base PPO to PPO-RNN-Concat, and PPO-Sensor. Across all terrains, both enhanced variants consistently outperformed standard PPO. PPO-RNN-Concat achieved improvements ranging from approximately 16% to 28%, confirming that recurrent layers provide a clear advantage in environments where temporal information can be exploited. By leveraging past interactions, the recurrent policy implicitly infers terrain characteristics and adjusts behavior accordingly, resulting in more adaptive and robust locomotion.

PPO-Sensor, in turn, reached even higher gains—ranging from 13% up to 56%—since it benefits from privileged access to exteroceptive terrain information. This additional sensing allows the agent to make more informed decisions without the need to rely on memory, effectively serving as an upper-bound baseline. Although such privileged information is not typically available in real-world deployments, these results highlight the performance ceiling achievable when terrain features are explicitly provided.

However, despite PPO-Sensor achieving the best overall performance, its results could have been even stronger. One limiting factor was the variability across different random seeds: in some runs, the algorithm failed to converge to a sufficiently good policy. A possible explanation for this instability lies in the high dimensionality of the observation space—while the recurrent variants operated with a reduced set of 48 features, the addition of exteroceptive sensing in PPO-Sensor expanded this number to 235. The increased complexity of the input may have made training more challenging, leading some seeds to struggle in discovering effective policies.

This outcome shows that, even when provided with privileged environment information, there is no guarantee that the agent will always learn an optimal—or even adequate—policy. Another notable limitation emerged in the boxes terrain, where PPO-Sensor underperformed relative to expectations. This terrain presents particularly challenging obstacles that demand specialized training to master, which likely explains the reduced scores.

In contrast, the PPO baseline without exteroceptive sensing exhibited a much more generic behavior. Since the robot has no information about upcoming topography, the policy was forced to learn a broad, terrain-agnostic strategy capable of handling most scenarios reasonably well, but without specializing in any particular type of obstacle. This generic behavior explains its lower overall performance, yet it also highlights that even without privileged perception or memory, the agent can still acquire a moderately robust locomotion pattern that attempts to generalize across a wide range of unseen terrains. Figure 4.13 illustrates this effect for PPO without sensor under a unstructured terrain.

Figure 4.13: Robot behavior under unstructured mixed terrain using the PPO baseline without exteroceptive sensing. Lacking information about upcoming terrain features, the robot executes a generic, terrain-agnostic policy, which leads to characteristic failure modes such as stumbling, loss of balance, and inadequate foot placement. These behaviors illustrate the limitations of a non-recurrent, sensorless policy when facing unpredictable topography.

PPO-RNN-Concat, while generally performing slightly below PPO-Sensor, demonstrated far greater stability across seeds. This consistency reflects the robustness of the recurrent architecture: by relying on temporal abstraction rather than privileged perception, it was able to consistently acquire competent locomotion strategies across terrains. Figure 4.14 illustrates this effect for PPO-RNN-Concat under a unstructured terrain.

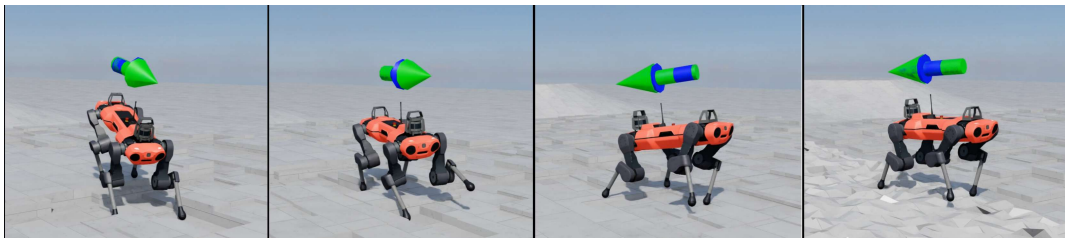

Figure 4.14: Robot behavior under unstructured rough terrain using the PPO-RNN-Concat algorithm. By leveraging its recurrent memory, the agent adapts online to irregular topography, maintaining stable locomotion even without access to exteroceptive terrain information. The sequence highlights the robot's ability to handle unpredictable surface variations while avoiding common failure modes such as stumbling, loss of balance, or improper foot placement.

Taken together, the comparison between PPO-RNN-Concat and PPO-Sensor illustrates two complementary pathways for enhancing locomotion policies: one based on memory and temporal abstraction, and another based on privileged perception. While PPO-Sensor unsurprisingly sets the highest benchmark, the stable and reliable improvements of PPO-RNN-Concat highlight that recurrence alone enables substantial gains in generalization and robustness, even under the absence of explicit terrain sensing.

Figure 4.15 shows the mean learning curves across 10 seeds for each algorithm in the mixed rough terrain setting. The plot confirms the trends observed in the quantitative results: PPO-RNN-Concat clearly outperformed standard PPO, reaching higher reward levels with steadier progress and narrower variance across seeds. PPO-Sensor, benefiting from privileged exteroceptive information, achieved the highest overall performance and faster convergence, but also displayed larger variability across runs, consistent with the instability reported in the quantitative analysis. In contrast, PPO-RNN-Concat showed slower but more consistent growth, reflecting its ability to rely on temporal abstraction rather than privileged input.

It is also important to note that training in this mixed terrain scenario required a significantly larger number of timesteps compared to the flat terrain experiments. While the most challenging flat case (FI = 0.1) demanded 250k timesteps to converge, the mixed terrain setup required up to 1M timesteps. This fourfold increase in sample complexity translated to a substantial computational cost, with each seed requiring approximately 16 hours of wall-clock time on our hardware. This slower convergence reflects the higher complexity of the environment: the agent needed more time—and significantly more computation—to gather sufficient information to effectively adapt to the diverse and unpredictable terrain dynamics.

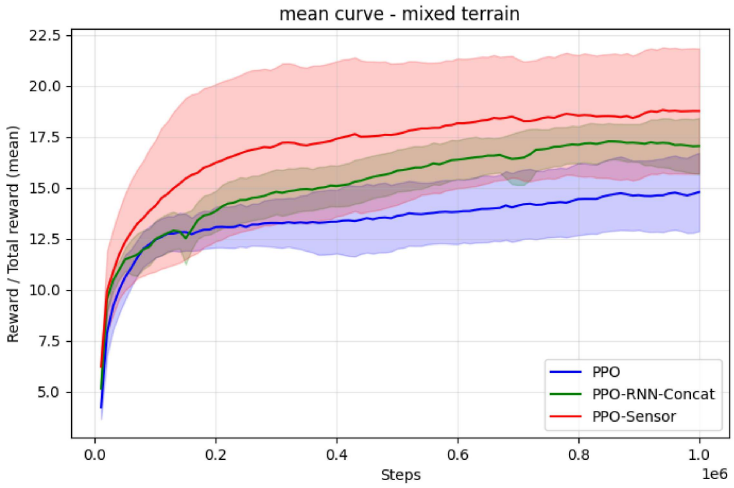

Figure 4.15: Mean training curves across 10 random seeds for each algorithm for mixed terrain. The shaded regions represent 95% confidence intervals across seeds

4.3 Summary

Overall, the experiments demonstrate that PPO-RNN-Concat consistently outperforms the other approaches, providing meaningful gains in both performance and stability across different terrains and friction conditions. In the extreme no-friction case (FI = 0.0), none of the algorithms were able to learn meaningful locomotion, since the robot could not generate sufficient traction to move effectively. This highlighted a fundamental limitation: recurrent memory cannot compensate for the absence of viable interaction signals.

At very low friction (FI = 0.1), however, PPO-RNN-Concat achieved substantial gains over PPO, with improvements exceeding 200% when trained and tested under the same condition. This confirmed that recurrence provides a crucial advantage in scenarios requiring fast online adaptation, where small changes in traction must be detected and compensated within seconds to prevent instability. PPO-RNN-Hidden, while occasionally beneficial, showed less consistent improvements in this regime.

In contrast, under moderate and high friction (FI = 0.5 and 1.0), all algorithms—including standard PPO—achieved effective locomotion. PPO-RNN-Concat offered some benefits in terms of narrower confidence intervals and improved transfer to nearby friction levels (e.g., from FI = 0.5 to 1.0), but its advantage diminished when locomotion was already easy to learn. Training at high friction (FI = 1.0) even led to poor generalization, since policies tended to overfit to the favorable conditions and failed when tested

under more challenging slip dynamics.

When trained under the Meta-RL scenario of randomized friction conditions (RR and RI), PPO-RNN-Concat again stood out, achieving improvements of up to 70% compared to PPO. In these non-stationary environments, where friction changes either between episodes (RR) or within them (RI), the recurrent memory proved that they can encode the history of interactions, retaining information about previously encountered slip values. This embedded memory allows the policy to learn and use this informations to track variations and adapting online. PPO-RNN-Hidden, on the other hand, rarely showed significant differences from PPO, suggesting that the concatenation of past interactions with current observations was a more effective way of exploiting recurrence.

Finally, evaluations on unseen rough terrains confirmed that recurrent policies generalize better to unstructured environments. PPO-RNN-Concat consistently outperformed PPO, with gains of up to 28%, while PPO-Sensor—which used privileged exteroceptive information—achieved even higher returns, effectively representing an upper-bound reference.

5
Conclusion and Future work

5.1
Limitations

Although the proposed approach demonstrates promising results for blind locomotion and fast adaptation, several limitations must be acknowledged regarding the methodology, model calibration, and experimental design.

A major limitation of this work is the strong dependence of the robot's performance on the calibration of the GRU parameters. Variations in hidden size, number of layers, or sequence length can lead to substantially different levels of stability and adaptability across terrains. This sensitivity indicates that the recurrent architecture does not generalize uniformly across configurations, making the learning process highly dependent on proper hyperparameter tuning.

Closely related to this issue, the calibration of the recurrent network was performed manually. Despite following a systematic exploration of configurations, both the evaluation of results and the selection of the final model relied on manual interpretation. This process is labor-intensive and inherently suboptimal, offering no guarantee that the chosen configuration is near-optimal.

Another limitation lies in the architectural choices. The experiments were conducted using only a single MLP backbone, which restricts the generality of the conclusions. More expressive or specialized feature extractors—such as deeper MLPs, CNN-based encoders for spatial structures, or attention-based architectures—were not explored, and their impact on locomotion performance remains unknown.

Additionally, several robot- and environment-level parameters were kept fixed throughout the experiments. Factors such as entropy regularization, actuator gains, domain randomization ranges, and reward shaping coefficients were not systematically varied. Because these elements can affect training stability and robustness, the conclusions drawn here are limited to the specific parameterization adopted.

The study also relied exclusively on the ANYmal C robotic model and operated solely in simulation. Although simulation provides a controlled environment to evaluate recurrent policies, the absence of real-world experiments means that issues such as unmodeled dynamics, sensor noise, hardware wear, and terrain compliance were not addressed. As a result, the applicability of

the findings to physical systems remains uncertain.

Finally, the analysis focused primarily on performance metrics, with minimal investigation into the internal representations learned by the recurrent network. Understanding what information the GRU retains, how memory evolves over time, and how it reacts to changes in terrain could provide deeper insight into failure cases and generalization properties, but such analysis was not included in this work.

5.2 Conclusion

This work investigated the application of Meta-Reinforcement Learning to enhance the adaptability and robustness of quadrupedal robots in blind locomotion scenarios, where exteroceptive sensing is unavailable and terrain properties, such as slip, cannot be directly observed. Building upon the RL^2 framework, we proposed two recurrent policy architectures—PPO-RNN-Concat and PPO-RNN-Hidden—built upon the PPO algorithm.

Simulation results demonstrated that PPO-RNN-Concat, in particular, outperformed standard PPO across multiple friction conditions and unseen rough terrains. Its recurrent structure enabled the policy to capture temporal dependencies and implicitly infer task-specific dynamics from proprioceptive feedback alone. This translated into improvements in average total rewards and narrower confidence intervals, reflecting increased policy stability and robustness under variable ground conditions.

Unlike PPO-RNN-Hidden, which relies solely on hidden-state memory, PPO-RNN-Concat leverages the explicit concatenation of past interactions with current observations, allowing it to dedicate the hidden state to detecting non-observable state information. The recurrent memory also demonstrated the ability to encode the history of interactions, retaining information about previously encountered slip values. This embedded memory enabled the policy to track variations and adapt online, reacting in real time to subtle changes in traction. However, the recurrent memory cannot compensate for the absence of viable interaction signals, meaning that it is unable to improve policies in situations where locomotion itself is physically infeasible. Taken together, these properties explain why PPO-RNN-Concat provided more stable, adaptive, and resilient policies than standard PPO.

Moreover, evaluations on unstructured terrains demonstrated that incorporating memory of past interactions allows the agent to adapt effectively to unexpected changes and obstacles. While standard PPO tends to learn generic movement strategies that suffice for most terrains, it lacks the capacity to

recall task-specific information across episodes. By contrast, recurrent policies—especially PPO-RNN-Concat—benefit from temporal memory, enabling the robot to recognize and react to terrain patterns encountered before.

Regarding PPO-RNN-Hidden in both cases, its behavior closely resembles that of the baseline PPO. The improvements are minor and inconsistent, often failing to show statistically significant differences. This indicates that simply adding recurrence to the hidden state without concatenating explicit observations may not be sufficient to capture the temporal dependencies needed for more effective adaptation.

Taken together, these results indicate that recurrence is not universally beneficial but plays a decisive role in conditions that demand rapid adaptation (e.g., low friction or randomized terrains). When locomotion is inherently easy (high friction), standard PPO already suffices, and recurrence mainly contributes to stability rather than raw performance.

Among the evaluated methods, PPO-RNN-Concat emerged as the most promising approach in our experiments, combining higher returns with improved generalization across different friction levels and heterogeneous terrains. These results suggest that incorporating recurrence provides a significant advantage for learning adaptive locomotion policies in unpredictable and non-stationary environments. Nonetheless, the approach remains limited by its strong sensitivity to GRU hyperparameter calibration and by the manual tuning procedure adopted in this work, which may affect reproducibility and scalability. Despite these constraints, our findings highlight the potential of Meta-RL and recurrent architectures to advance blind quadrupedal locomotion, reducing reliance on external sensors and enabling more robust performance in real-world scenarios characterized by uncertainty and dynamic terrain conditions.

5.3 Future Works

Future research should investigate hybrid approaches that combine recurrent memory with limited exteroceptive sensing, aiming to balance robustness with improved terrain awareness. One promising direction is to provide exteroceptive data—such as LiDAR-based height profiles—exclusively to the critic during training. Since the critic is not required during deployment, this strategy could enhance learning without increasing the sensing requirements of the final policy executed on the robot.

Transferring the proposed approach to real-world hardware represents another essential step. Evaluating the recurrent policies on a physical quadruped

would help determine whether the benefits observed in simulation translate to environments with unmodeled dynamics, sensor noise, contact uncertainties, and complex terrain interactions.

Another important direction for future investigation concerns the calibration of recurrent architectures. As observed in this work, the performance of memory-based policies is highly sensitive to the choice of GRU hyperparameters, such as hidden size, number of layers, and sequence length. Small variations in these values can lead to markedly different levels of stability, adaptability, and convergence across terrains. However, the calibration process carried out in this study relied on manual and systematic exploration, which is both time-consuming and suboptimal. Future research should therefore incorporate automated hyperparameter optimization techniques—such as Bayesian optimization, evolutionary strategies, or population-based training—to jointly tune the recurrent network and learning parameters. Such methods could yield more reliable configurations, reduce sensitivity to initialization, and improve the overall consistency of recurrent policies across diverse environments.

Moreover, alternative memory-based architectures could be explored to further improve adaptability and long-term reasoning. Promising candidates include state concatenation strategies, attention mechanisms, and hierarchical recurrent models, each offering different capabilities for capturing temporal dependencies and latent terrain characteristics.

Finally, the results obtained in this work highlight the broader potential of recurrent neural networks in legged locomotion. Memory-based policies offer multiple benefits for adapting to unstructured environments, suggesting a wide range of future opportunities for research in learning-based locomotion control.

6
Bibliography

[1] B. van Marum, M. Sabatelli, and H. Kasaei, "Learning perceptive bipedal locomotion over irregular terrain," *arXiv preprint arXiv:2304.07236*, 2023.

[2] B. Zhang, G. Li, Q. Zheng, X. Bai, Y. Ding, and A. Khan, "Path planning for wheeled mobile robot in partially known uneven terrain," *Sensors*, vol. 22, no. 14, p. 5217, 2022.

[3] M. Liu, D. Qu, F. Xu, F. Zou, J. Song, C. Tang, Z. Ma, and L. Jiang, "Dynamic modeling of quadrupedal robot based on the screw theory," in *2019 Chinese Automation Congress (CAC)*, pp. 5540–5544, IEEE, 2019.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5761–5768, IEEE, 2018.

[6] T. P. de Carvalho, V. S. Medeiros, and M. A. Meggiolaro, "Analysis and comparison of slip detection methods for quadruped robots," in *2024 Latin American Robotics Symposium (LARS)*, pp. 1–6, IEEE, 2024.

[7] M. Aractingi, P.-A. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères, "Controlling the solo12 quadruped robot with deep reinforcement learning," *scientific Reports*, vol. 13, no. 1, p. 11945, 2023.

[8] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 2908–2927, 2022.

[9] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, S. Whiteson, *et al.*, "A tutorial on meta-reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 18, no. 2-3, pp. 224–384, 2025.

[10] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," 2017.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] A. Csillag, L. Dávid, L. Márton, E. Györfi, and Z. Köllő, "Intelligent kinematic control of a quadrupedal mobile robot," in *2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 000067–000072, IEEE, 2024.

[13] A. Hidayat, A. N. Jati, and R. E. Saputra, "Autonomous quadruped robot locomotion control using inverse kinematics and sine pattern methods," in *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pp. 96–100, IEEE, 2016.

[14] K. S. Holkar and L. M. Waghmare, "An overview of model predictive control," *International Journal of control and automation*, vol. 3, no. 4, pp. 47–63, 2010.

[15] D. Kang, F. De Vincenti, and S. Coros, "Nonlinear model predictive control for quadrupedal locomotion using second-order sensitivity analysis," *arXiv preprint arXiv:2207.10465*, 2022.

[16] O. Villarreal, V. Barasuol, P. M. Wensing, D. G. Caldwell, and C. Semini, "Mpc-based controller with terrain insight for dynamic legged locomotion," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2436–2442, 2020.

[17] G. Lu, T. Chen, X. Rong, G. Zhang, J. Bi, J. Cao, H. Jiang, and Y. Li, "Whole-body motion planning and control of a quadruped robot for challenging terrain," *Journal of Field Robotics*, vol. 40, no. 6, pp. 1657–1677, 2023.

[18] C. Dario Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3359–3365, 2017.

[19] L. Amanzadeh, T. Chunawala, R. T. Fawcett, A. Leonessa, and K. A. Hamed, "Predictive control with indirect adaptive laws for payload transportation by quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 9, no. 11, pp. 10359–10366, 2024.

[20] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3402–3421, 2023.

[21] M. Gurram, P. K. Uttam, and S. S. Ohol, "Reinforcement learning for quadrupedal locomotion: Current advancements and future perspectives," in

2025 9th International Conference on Mechanical Engineering and Robotics Research (ICMERR), pp. 28–38, IEEE, 2025.

[22] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath, "Learning torque control for quadrupedal locomotion," in *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, pp. 1–8, IEEE, 2023.

[23] J. Lee, L. Schroth, V. Klemm, M. Bjelonic, A. Reske, and M. Hutter, "Exploring constrained reinforcement learning algorithms for quadrupedal locomotion," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11132–11138, IEEE, 2024.

[24] X. Han and M. Zhao, "Learning quadrupedal high-speed running on uneven terrain," *Biomimetics*, vol. 9, no. 1, p. 37, 2024.

[25] F. Di Giuro, F. Zargarbashi, J. Cheng, D. Kang, B. Sukhija, and S. Coros, "Meta-reinforcement learning for universal quadrupedal locomotion control," *arXiv e-prints*, pp. arXiv–2407, 2024.

[26] Y. Zhao, T. Wu, Y. Zhu, X. Lu, J. Wang, H. Bou-Ammar, X. Zhang, and P. Du, "Zsl-rppo: Zero-shot learning for quadrupedal locomotion in challenging terrains using recurrent proximal policy optimization," *arXiv preprint arXiv:2403.01928*, 2024.

[27] Á. Belmonte-Baeza, J. Lee, G. Valsecchi, and M. Hutter, "Meta reinforcement learning for optimal design of legged robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12134–12141, 2022.

[28] A. Kuzhamuratov, D. Sorokin, A. Ulanov, and A. Lvovsky, "Adaptation of quadruped robot locomotion with meta-learning," *arXiv preprint arXiv:2107.03741*, 2021.

[29] I. Nahrendra, B. Yu, M. Oh, D. Lee, S. Lee, H. Lee, H. Lim, and H. Myung, "Obstacle-aware quadrupedal locomotion with resilient multi-modal reinforcement learning," *arXiv preprint arXiv:2409.19709*, 2024.

[30] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[31] S. D. Marcus, A. Shapiro, and C. Giladi, "Enhancing quadruped robot walking on unstructured terrains: A combination of stable blind gait and deep reinforcement learning," *Electronics*, vol. 14, no. 7, p. 1431, 2025.

[32] Y. Nisticò, S. Fahmi, L. Pallottino, C. Semini, and G. Fink, "On slip detection for quadruped robots," *Sensors*, vol. 22, no. 8, p. 2967, 2022.

[33] S. Teng, M. W. Mueller, and K. Sreenath, "Legged robot state estimation in slippery environments using invariant extended kalman filter with velocity update," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3104–3110, IEEE, 2021.

[34] T. H. F. d. Oliveira, *Algoritmos de Aprendizagem por Reforço para Problemas de Otimização Multiobjetivo*. PhD thesis, UFRN, Rio Grande do Sul, 2021.

[35] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[37] N. Mohammadpour, M. Fozi, M. M. Ebadzadeh, A. Azimi, and A. Kamali, "Proximal policy optimization with adaptive generalized advantage estimate,"

[38] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 2002.

[39] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling. arxiv 2014," *arXiv preprint arXiv:1412.3555*, vol. 1412, 2014.

[40] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600, IEEE, 2017.

[41] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[42] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, pp. 91–100, PMLR, 2022.

[43] A. Serrano-Muñoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba, "skrl: Modular and flexible library for reinforcement learning," *Journal of Machine Learning Research*, vol. 24, no. 254, pp. 1–9, 2023.

[44] H. Taud and J.-F. Mas, "Multilayer perceptron (mlp)," in *Geomatic approaches for modeling land change scenarios*, pp. 451–455, Springer, 2017.

[45] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, *et al.*, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.