ENHANCING ROBOT SOCIAL NAVIGATION WITH REINFORCEMENT LEARNING AND ADVANCED PREDICTIVE MODELS: COSINE-GATED-LSTM AND ADAPTIVE PREDICTIVE HORIZONS

DIRICHUKWU GOODLUCK OGUZIE

Doctor of Philosophy

ASTON UNIVERSITY

December 2024

Abstract

This thesis presents a comprehensive exploration of Social Robot Navigation (SocNav) in human-centric environments, a field of growing importance as robots become integral to sectors such as healthcare, hospitality, and public service. The research focuses on the integration of Reinforcement Learning (RL) with advanced predictive models to improve the navigation and interaction capabilities of robots in social environments.

A significant contribution of this work is the development and integration of our novel predictive world models into RL frameworks. These models improve the agent's ability to predict future states, thereby improving decision-making efficiency and adaptability in a dynamic social environment. However, the initial implementation of fixed prediction horizons, such as always predicting two steps ahead in the *2StepAhead* model, revealed limitations in flexibility and computational efficiency. Addressing this, we introduced an entropy-driven adaptive prediction horizon mechanism that dynamically adjusts the prediction horizon based on real-time policy entropy, balancing computational resources with the need for long-term future state prediction.

An important method in this thesis is the introduction of the Cosine-Gated Long Short-Term Memory (CGLSTM) model. By integrating a cosine similarity-based gating mechanism with vanilla LSTM (Long Short-Term Memory) networks, CGLSTM significantly advances sequence prediction capabilities. The model consistently outperformed vanilla LSTM, GRU (Gated Recurrent Units), and RAU (Recurrent Attention Unit) models, achieving up to a 30% reduction in Mean Absolute Error (MAE) in environments such as FallingBallEnv and SocNavGym. Furthermore, integrating CGLSTM into DreamerV a state-of-the-art model-based reinforcement learning framework that learns a latent world

model and plans actions through imagination resulted in an approximately 5% increase in cumulative reward, demonstrating that stronger predictive sequence models can directly enhance RL performance.

The thesis also addresses the computational challenges associated with predictive models in varying environmental complexities. The entropy adaptive prediction horizon mechanism effectively mitigates the computational challenges by adjusting the prediction horizon in response to environmental uncertainty, leading to a 15% improvement in success rates in high-entropy scenarios while maintaining computational efficiency with only a 2% increase in inference time in low-entropy situations.

Overall, this thesis significantly contributes to the advancement of SocNav and predictive modeling within RL, laying the groundwork for future research aimed at integrating robots more intuitively into our society. The developed models improve robots' ability to navigate complex environments with improved predictive models and computational efficiency, paving the way for seamless integration into various sectors.

Keywords: Social Robot Navigation, Reinforcement Learning, Predictive World Models, Cosine-Gated LSTM, Adaptive Horizon, DreamerV3, Sequence Prediction, Computational Efficiency, Human-Robot Interaction.

To my beloved only sister, Sandra Oguzie, who passed away on May 17, 2024. Your memory inspires me every day. Although my PhD journey extended from three to five years, preventing me from seeing you again, your spirit remains a guiding light in my endeavors.

Acknowledgements

As I reflect on my journey to complete this thesis, I am filled with gratitude for the special people who supported me through a period rich in new experiences and marked by significant challenges, not least due to the COVID-19 pandemic. The assistance I received from friends and family was indispensable.

Firstly, I extend my thanks to Dr. Faria Diego, my initial supervisor who guided the commencement of my research, but subsequently left the school. His place was taken by Dr. Luis J. Manso, whose profound knowledge in robotics, computer science, and AI was instrumental in my work.

I also express my sincere appreciation to my associate supervisors, Prof. Aniko Ekart and Dr. George Vogiatzis. Their insightful contributions greatly enhanced my thesis. Dr. Vogiatzis departed from the university however, their influence remains a valuable part of my work.

My gratitude also extends to my friends in the Computer Science department. Dr. Renato Arantes, Dr. Daniel Rodriguez Criado, Cliona Kelly, and Vincent Zakka were more than just colleagues; they were wonderful companions who significantly brightened this journey.

This thesis would not have been possible without the collective input of these individuals. Their support, guidance, and friendship have been invaluable learning experiences for me.

Heartfelt thanks to all my friends in Birmingham who augmented this transformative journey. A special mention to Ola Taiwo for organizing weekly football sessions to ease the

stress of my Ph.D. studies. Your presence has unquestionably enriched my experience in this city.

Lastly, I offer my deepest gratitude to my family - my parents, Hon. Ifeanyi and Lolo Benedette, and my siblings. Your unwavering belief in me and the strength you provided during the most challenging moments have been truly uplifting. To Nneka, your steadfast support has been an integral part of this journey, especially during the thesis's last and most demanding stages. Your continuous care and encouragement were a pillar of strength that made this arduous journey considerably smoother. For this, I am eternally grateful.

List of Publications

Publications Arising from this Thesis:

- [90] Oguzie, Goodluck, Ekárt, Anikó (2023). Predictive World Models for Social Navigation. In: Advances in Computational Intelligence Systems, Contributions Presented at the 22nd UK Workshop on Computational Intelligence. Advances in Intelligent Systems and Computing (AISC). GBR: Springer. (In Press)

- [89] Oguzie, Goodluck. "Cosine-Gated LSTM." In 2024 IEEE 5th International Conference on Pattern Recognition and Machine Learning (PRML), pp. 8-15. IEEE, 2024.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a busy hospital hallway, a robot moves carefully between patients and staff, stopping briefly as a nurse hurries past. What once seemed like science fiction is now becoming real as robots start to work in places where people are present. From healthcare to hotels, robots that can move around social spaces are changing industries. However, making sure they can interact smoothly with people is still a big challenge.

Social Robot Navigation (SocNav) is becoming increasingly important as robots are used in healthcare, hospitality, and public service. In these areas, robots must move around efficiently and follow social norms. This requires more than just planning their paths; robots need to understand and interact with people in human-centered environments [77]. As technology advances, robots are becoming more capable and can better understand human behavior and social rules. This progress promises a future where robots integrate smoothly into our daily lives and positively influence social interactions [63, 85].

SocNav is a multidisciplinary field that combines robotics, psychology, sociology, and human-robot interaction [54]. Early robot navigation methods focused heavily on geometric path planning and obstacle avoidance in structured environments, often treating humans as moving obstacles [58]. Over time, researchers recognized the limitations of purely collision-avoidance-based strategies and began integrating social norms, human behavior, and intention prediction into navigation models. Now, research in SocNav includes adapting

to social norms, improving interactions with humans, and understanding human intentions. This growth is driven by rapid technological advances and a better understanding of the technical and social aspects of the coexistence of humans and robots [77].

Reinforcement learning (RL) plays an important role in this progress, providing a framework for robots to learn and adapt their navigation strategies dynamically. Unlike rule-based programming [45], which relies on pre-defined instructions and often struggles to handle the variability of dynamic environments, RL enables robots to make decisions that maximize long-term rewards, aligning their actions with desired outcomes [124, 44]. This paradigm shift has led to the emergence of more flexible and context-sensitive strategies in complex social scenarios.

However, applying RL to social navigation poses several challenges. One core issue lies in modeling human behaviours and social norms. In real-world contexts, high-quality data for training RL models can be limited, and ensuring safety is paramount. Simulated environments, while beneficial for training and validation, often fail to capture the full complexity of human interactions. This discrepancy raises ongoing research questions about the development of more accurate predictive models for dynamic environments.

To address these challenges, our research uses advanced machine learning techniques. Specifically, we combine RL with sequence modeling methods to improve robot navigation. We focus on integrating Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) units, along with predictive world models, into RL frameworks. These improvements help manage long-term dependencies, handle outliers, and accurately predict future states. As a result, robots can make better decisions in complex and uncertain environments.

Moreover, while RL has excelled in most discrete action space environments, real-world tasks often naturally involve continuous action spaces that demand fine-grained control and adaptability. This transition from discrete to continuous actions introduces additional complexity in exploration, sample efficiency, and long-term planning. Traditional RL frameworks with predictive models typically rely on fixed prediction horizons, which limit the agent's ability to adapt dynamically to varying levels of environmental complexity. For in-

stance, a predictive model might always plan two steps ahead (as in our 2StepAhead model), irrespective of the environmental conditions. While effective in certain scenarios, this fixed-horizon strategy can be computationally expensive or suboptimal when the environment fluctuates between complex, crowded settings and simpler, less challenging settings.

Recognizing this gap, we proposed an adaptive prediction horizon mechanism driven by entropy. By using policy entropy as a measure of uncertainty, the agent dynamically adjusts its prediction horizon: longer horizons are selected in uncertain, complex situations, while shorter horizons conserve computational resources in more predictable contexts. This adaptive approach not only improves long-term efficiency, but also improves agent performance, bridging the gap between theoretical models and practical, real-world implementations of RL in social navigation tasks.

1.1 Research Questions

This thesis focuses on two primary research questions that guide our exploration:

1. **How do predictive world models improve decision-making in Social Robot Navigation (SocNav)?** This question examines how advanced predictive models improve a robot's ability to predict future states and make better decisions in complex, human-centered environments. These models are considered advanced compared to traditional rule-based systems, which often lack the foresight and adaptability required for dynamic social contexts.

2. **What challenges arise when transitioning Reinforcement Learning applications from discrete to continuous action spaces in Social Robot Navigation, and how can we address the challenges?** This inquiry explores the difficulties when moving RL applications from limited, discrete actions space to a wide range of continuous actions space in SocNav. It evaluates how this shift affects decision-making accuracy, learning efficiency, and adaptability.

1.2 Contributions

Our research contributes to both RL and SocNav in several ways by integrating predictive models, improving sequence modeling, and introducing adaptive prediction horizons.

1.2.1 Predictive World Models for social navigation

- **Development of Predictive Models**: We developed and integrated predictive world models such as *2StepAhead*, *MASPM*, and *2StepAhead-MASPM* into RL frameworks. These models enhance the agent's ability to anticipate future states, thereby improving decision-making in dynamic environments (see Chapter 5, *Predictive World Models for Social Navigation*, and Chapter 6, *Cosine-Gated LSTM*).

- **Enhanced Decision-Making**: The integration of these predictive models demonstrated significant improvements in the agent's performance, particularly in complex scenarios like SocNavGym. By predicting future states, the RL agent can make more informed and strategic decisions, leading to higher success rates and more efficient navigation paths.

1.2.2 Advanced Sequence Modeling with Cosine-Gated Long Short-Term Memory (CGLSTM)

- **Contribution of CGLSTM and Superior Performance**: We introduced the Cosine-Gated LSTM (CGLSTM) model, which integrates a cosine similarity-based gating mechanism with vanilla LSTM networks. This method addresses the limitations of vanilla recurrent architectures in handling long-term dependencies and outliers in sequence (see Chapter 6,*Cosine-Gated LSTM*). As a result, CGLSTM consistently outperformed LSTM, GRU, and RAU models in sequence prediction tasks, achieving up to a 50% reduction in MAE in environments such as FallingBallEnv and SocNavGym. These performance improvements highlight the effectiveness of CGLSTM in improving the predictive capabilities of RL agents.

- **Integration into DreamerV3**: We successfully integrated CGLSTM into the state-of-the-art DreamerV3 model, replacing the vanilla GRU. This integration improved the cumulative rewards, demonstrating how CGLSTM can enhance existing RL frame-

works' predictive and decision-making processes.

1.2.3 Entropy-Driven Adaptive Prediction Horizon Mechanism for RL

- **Entropy-Driven Adaptive Prediction Horizon**: To address the limitations of fixed-horizon predictions, we introduced an entropy-driven adaptive prediction horizon mechanism while focusing continuous action spaces. This method dynamically adjusts the planning horizon based on real-time policy entropy, balancing computational efficiency with the need for long-term planning in uncertain, dynamically changing environments (see Chapter 7).

- **Enhanced Efficiency and Adaptability**: The adaptive prediction horizon mechanism allowed RL agents to modify their prediction horizon in response to environmental uncertainty. This adaptability resulted in a 15% improvement in success rates in high-entropy scenarios within SocNavGym. Meanwhile, in simpler, low-entropy situations, the agent reduced planning depth, conserving computational resources and maintaining efficiency.

- **Integration with SAC**: By integrating the CGLSTM and the adaptive prediction horizon mechanism into the Soft Actor-Critic (SAC) framework, we created a hybrid model leveraging the strengths of both predictive modeling and adaptive prediction horizon. This integration led to significant improvements in cumulative rewards, policy stability, and overall adaptability, further bridging the gap between theoretical RL models and practical real-world applications.

1.3 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 explores the domain of Social Robot Navigation. The integration of RL and predictive modeling techniques is highlighted, demonstrating the complexities and requirements of human-centric navigation tasks.

Chapter 3 introduces the basic concepts and structures of deep neural networks, focusing on foundational knowledge for subsequent chapters. Readers familiar with deep

learning fundamentals may choose to skip this chapter.

Chapter 4 introduces the fundamentals of Reinforcement Learning, examining key algorithms, their features, and limitations.

Chapter 5 introduces our predictive world models (2StepAhead, MASPM, 2StepAhead-MASPM) and evaluates their performance in RL-based social navigation scenarios. The chapter provides insights into their strengths and identifies the limitations of fixed horizon predictions.

Chapter 6 presents the Cosine-Gated LSTM (CGLSTM) model and discusses its integration into DreamerV3. This chapter demonstates how CGLSTM significantly improves sequence prediction and cumulative reward acquisition during RL training.

Chapter 7 focuses on the entropy-driven adaptive prediction horizon mechanism, detailing its implementation, evaluation, and integration with SAC. We highlight how this proposed method addresses computational overhead and improves adaptability and efficiency in dynamic and complex environments.

Chapter 8 concludes the thesis by summarizing key findings, discussing the broader impacts and potential real-world applications, and outlining directions for future research and refinement.

Chapter 2

Social Robot Navigation

2.1 Introduction to Social Robot Navigation

Social Robot Navigation (SocNav) is an interdisciplinary field that combines robotics, human-robot interaction (HRI), psychology, and sociology. The goal is to develop systems that allow mobile robots to effectively move around in environments populated by humans. In this chapter, we first outline the historical development of SocNav and then review major approaches to social navigation, including classical rule-based methods and modern machine learning techniques. Beyond ensuring safety, SocNav aims to adapt to human social norms and improve human–robot interactions. In this context social norms refers to the unwritten rules that govern human behaviour for instance, interpersonal distance (proxemics), queuing etiquette, or customary gestures. Respecting these norms enables robots to navigate in ways that people find natural and comfortable. Over the past three decades, the field has significantly grown, driven by technological advancements and a better understanding of the social aspects of human–robot coexistence [77].

2.2 Historical Overview of Social Robot Navigation

In the early stages of robot navigation research, most methods focused on geometric path planning and obstacle avoidance in structured environments, often treating humans merely as moving objects [58]. Early algorithms such as the Dynamic Window Approach, Prob-

abilistic Roadmaps, and Rapidly-Exploring Random Trees aimed to compute collision-free paths by minimising crash rates [70]. As a result, robot behaviours often appeared unnatural or socially inappropriate from a human perspective, as observed in empirical studies evaluating human-robot interaction in shared spaces [65].

This lack of social awareness stemmed from the fact that traditional planners did not incorporate models of human comfort, social interaction cues, which are essential when navigating shared spaces [77].

During the 1980s and 1990s, the focus remained on preventing collisions in static or partially known environments. Human agents were either abstracted as static obstacles or afforded minimal dynamic modeling. This collision-centric mindset, while foundational for path-planning techniques used in industry, did not integrate notions of personal space, user comfort, or cultural conventions. Over time, researchers recognised that mere collision avoidance was insufficient in settings where robots and humans must coexist, prompting a shift toward more socially aware frameworks [25]. This recognition coincided with a growing interest in Human-Robot Interaction (HRI) as a distinct field, where user experience, acceptability, and trust in robots became an area of interest [10].

By the late 1990s and early 2000s, it became clear that robots needed to respect human comfort zones to avoid provoking anxiety or defensive behaviors [74, 101]. Initial methods introduced extended safety margins around humans and began exploring how social norms (e.g. body language, proxemics) could be integrated into navigational logic. Although these approaches were still simplistic, they laid the foundation for more advanced techniques in social robot navigation. These early adaptations often relied on empirically defined personal space boundaries [42], marking one of the first efforts to codify human spatial behaviour into robotic control systems.

As the field progressed, researchers became more focused on human-centered design principles. The introduction of concepts like proxemics [42] and the social force model [47] marked a turn toward modeling interpersonal distances and forces for path planning. Unlike purely geometric methods, these approaches explicitly integrated human comfort and perception into navigation algorithms, representing an important transition from obstacle-

based planning to socially aware navigation.

Additionally, these developments marked a move toward ethical and socially-intelligent robot behaviour, where actions are judged not only on efficiency but also on perceived politeness and appropriateness [23].

Between the early and mid 2000s, researchers began to use simulation platforms such as CARLA [22] and CrowdSim [128] to investigate social navigation at scale. Simulations allowed for repeated controlled experiments involving multiple human-robot interactions without risking hardware or human safety. Although early simulators struggled to replicate realistic human behaviour, they evolved rapidly, becoming foundational tools for training, testing, and validating social navigation methods in increasingly realistic scenarios [22].

The rise of these simulation environments also enabled reproducibility and benchmarking, critical for comparing navigation strategies under controlled yet varied social scenarios [110].

Recognising the limitations of early, purely geometric approaches, researchers began to explore data-driven methods that integrate social norms and human behavioral data. Reinforcement learning emerged as a promising method, enabling robots to learn from trial-and-error interactions in simulated or semi-structured environments [77, 58]. Simulation platforms such as SocNavGym [56] and SocialGym [51] further drive this transition, providing controlled contexts for rapid experimentation [58, 51].

These tools made it possible to encode human-centric metrics directly into the learning reward structure, allowing agents to optimise for both task success and social acceptability [151].

2.3 Approaches to Social Robot Navigation

Multiple strategies have been proposed to enable robots to navigate among humans in a socially acceptable manner. These approaches range from classical, rule-based frameworks enhanced to include social constraints to modern, data-driven techniques using machine learning and advanced methods.

2.3.1 Classical Approaches

Rule-Based and Heuristic Methods

Early social navigation systems often hard-coded heuristics such as "yield to pedestrians on the right" or "avoid crossing between two people facing each other" [55]. Though effective in structured settings (like office corridors), purely rule-based methods struggle in unstructured or culturally varied environments. As the diversity and dynamic nature of pedestrian behavior increase, maintaining and updating such rules quickly becomes impractical.

Social Force Model and Enhanced Potential Fields

One of the early and notable frameworks for modeling human movement in crowds is the Social Force Model (SFM), proposed by Helbing and Molnar [47]. In this model, each pedestrian is treated as if they experience "forces" from goals, obstacles, and other pedestrians. Formally, a simplified version of the Social Force Model can be expressed as:

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \frac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \sum_j \mathbf{F}_{ij} + \sum_B \mathbf{F}_{iB}, \tag{2.1}$$

where:

- m_i is the mass of pedestrian i.
- $\mathbf{v}_i$ is the pedestrian's velocity.
- $\mathbf{v}_i^0$ is the pedestrian's desired velocity.
- τ_i is a relaxation time constant.
- $\mathbf{F}_{ij}$ represents interaction forces between pedestrian i and other pedestrians j.
- $\mathbf{F}_{iB}$ denotes interaction forces between pedestrian i and boundaries or obstacles.

In robotics, researchers have adapted the Social Force Model to integrate more nuanced social norms, such as group cohesiveness, enabling robots to generate more human-like trajectories [71]. Building upon classical formulations like the SFM, enhanced potential

fields have been developed to better balance goal attraction with repulsive fields, thereby respecting personal space around humans. These "social potential fields" are conceptually intuitive and relatively straightforward to implement. However, they rely on carefully tuned environment-specific parameters and may struggle to capture higher-level social norms.

To address these limitations, various SFM variants have been proposed. These adaptations integrate additional factors such as dynamic changes in pedestrian intent, environmental context, and more complex interaction mechanisms. For instance, some variants integrate group behavior dynamics, allowing for the simulation of cohesive group movements within crowds. Others integrate real-time sensory data to dynamically adjust forces, improving the responsiveness of the model to changing environments [71].

Overall, while the original Social Force Model provides a solid foundation for understanding and simulating pedestrian dynamics, its improved variants offer improved realism and adaptability, making them more suitable for complex and dynamic environments encountered in real-world applications.

2.3.2 Machine Learning-Based Approaches

Data-driven methods have gained prominence in Social Robot Navigation (SocNav) due to their ability to learn complex social norms from examples. The field has evolved from early rule-based and simple models, which had limited adaptability to new and changing situations, to more advanced techniques such as imitation learning, reinforcement learning (RL), and graph neural networks. Among these, RL stands out for enabling robots to learn from their experiences, continuously improving their behavior through rewards and punishments [51].

Imitation Learning

Imitation learning uses recorded human trajectories or expert demonstrations to train navigation policies. Large-scale datasets that capture various pedestrian-robot interactions help the model learn cost functions that favor socially compatible paths [70]. One challenge is the possible mismatch between training and deployment environments. If real-world conditions differ significantly from the training data, performance may degrade.

Graph Neural Networks (GNNs) for Social Awareness

Graph Neural Networks have shown great promise in enhancing social awareness in robot navigation. By modeling the relationships and interactions between agents, GNNs allow robots to better understand and predict human behaviors, leading to more socially acceptable navigation strategies [101]. The primary strength of GNNs is their ability to handle complex interactions in crowded environments. However, they are computationally intensive and may struggle to scale effectively in real-time applications where quick decision-making is crucial [74].

Reinforcement Learning (RL)

Reinforcement Learning has become an important area of modern SocNav systems, enabling robots to learn optimal navigation strategies through interactions with their environment. Techniques such as Deep Q-Learning and Proximal Policy Optimization (PPO) have been beneficial in developing socially-aware navigation policies. The key strength of RL lies in its ability to adapt over time, improving its performance based on feedback received during navigation tasks [83, 109]. However, RL limitations include the high computational cost and the potential for unsafe behaviors during the learning phase, which requires careful simulation before real-world deployment [150]. Further details and elaborations on RL and its application in our research will be provided in chapter 4.

Importance of Simulated Environments in RL Research

Simulated environments are very important for advancing RL research within SocNav. They offer safe and controlled settings for testing algorithms, which significantly reduces the time and resources needed for development and iteration. Simulations allow for creating diverse scenarios that might be difficult or impossible to replicate in the real world, providing flexibility for developing versatile and robust navigation algorithms. These environments help in handling a wide range of social interactions and environmental conditions, which is important for the advancement of SocNav strategies [57].

2.4 SocNavGym

Simulated environments like SocNavGym [57], SocialGym [51], and CARLA [22] have been important in advancing SocNav research. These platforms provide safe and controlled environments for testing and refining navigation algorithms, reducing the risk of deploying untested strategies in real-world settings. While simulations offer flexibility and safety, they also present challenges, such as the "reality gap," where simulated behaviors do not perfectly translate to real-world applications [111].

While SocNavGym is our primary simulation environemnt, there are alternative platforms such as CrowdSim [128] and CARLA [22] for their distinct capabilities in simulating social environments. However, for our specific focus on social navigation, SocNavGym's features and ability to integrate complex human behavior models made it the preferred choice. As we continue to explore and refine these strategies, the role of SocNavGym and other simulated environments will remain essential, pushing the boundaries of what is possible in social navigation research. Further details and elaborations on SocNavGym and its application in our research will be provided in Chapters 5 and 6.

SocNavGym Environment

SocNavGym offers an environment where researchers can implement and evaluate their navigation algorithms. It simulates a variety of social contexts, including crowded public spaces, integrating models of human behavior to create somewhat realistic interaction scenarios [57, 56]. For our initial experiments, SocNavGym was configured with a discrete action space of four actions (stop, move forward, rotate left, and rotate right), three moving humans, and a social navigation reward function [4]. The agent's goal in SocNavGym is to navigate towards the target while avoiding collisions and minimizing discomfort to humans. A detailed discussion about this environment will be provided in Chapter 5

Benefit to Our Research

SocNavGym's simulation capabilities allow us to test different models across various social contexts, mirroring the complexities of real-world social navigation. It enables the optimization of our models for efficiency and safety, important for improved performance in actual human-robot interaction scenarios. A screenshot of the configured SocNavGym for

our experiments is shown in Fig. 2.1, showing humans, their goals, and robot agents in the simulated environment [58].

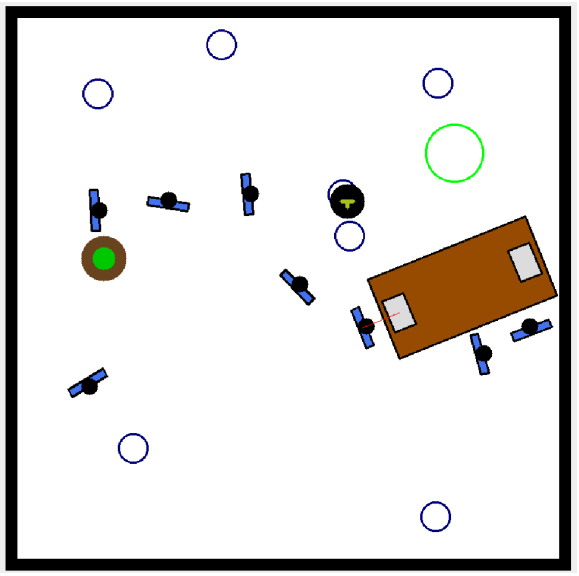


Figure 2.1: Screenshot of SocNavGym, the environment used for the experiments [58]. Blue squares represent humans, blue circles indicate humans' goals (which are non-observable by the robot), green circles represent the robot's goals, the large brown rectangle with white rectangles on top, located near the bottom right corner of the simulation area, represents a table with laptops on it and black-green circles represent robot agents.

The field of Social Robot Navigation (SocNav) has made significant strides in developing systems that can navigate human environments safely and effectively. However, challenges such as scalability, real-time adaptability, and generalisation to diverse environments remain.

In this thesis, we used the SocNavGym environment to develop and evaluate advanced models aimed at improving predictive accuracy and sequence prediction in social navigation contexts. Specifically, We used these models to address key challenges in social navigation, such as predictive world modeling and sequence prediction.

Preliminary Experiments in SocNavGym Environment

This section describes our initial experiments in the SocNavGym environment to test all our parameters we will be using for our research. We set out with three main goals: to verify whether SocNavGym reliably simulates social interactions, to compare different coordinate system setups (world vs. robot), and to evaluate how specific model architectures (e.g., a

Variational Autoencoder (VAE) and a Recurrent Neural Network (RNN)) integrate with SocNavGym's reward and observation structures.

In these experiments, we examined the Variational Autoencoder (VAE) for dimensionality reduction of high-dimensional inputs and a Recurrent Neural Network (specifically, an LSTM) for capturing temporal dependencies. Additionally, we tested our baseline discrete action reinforcement learning agent to identify any limitations in the environment setup, such as pitfalls of the reward function. Our primary contribution during this phase was the design and implementation of a novel reward function based on the Socially-aware Navigation Graph Neural Network (SNGNN) *SocNavGym* paper, which aimed to incorporate social norms and human comfort into the navigation task.

During the initial phase of our research, we worked with an early prototype version of the SocNavGym environment (`SocNavGym-v0`) while the main version (`SocNavGym-v1`) was still under development. This allowed us to focus on refining our reward function independently of the environment's maturity. However, as we progressed with our experiments, we encountered several issues with the stability and reliability of `SocNavGym-v0`, which hindered our ability to thoroughly evaluate our reward function. These issues included inconsistent collision detection and state updates, which made it challenging to draw reliable conclusions from our experiments.

Fortunately, by the time these limitations became apparent, the development team had released `SocNavGym-v1`, which addressed the stability concerns and provided a more robust platform for social navigation research (see the SocNavGym GitHub repository). We transitioned to `SocNavGym-v1` and continued our testing, ultimately validating the effectiveness of our reward function in a more stable environment. This transition not only allowed us to proceed with our research but also highlighted the importance of environment stability in reinforcement learning pipelines.

We further focused on optimising our hyperparameters, including window slides and timesteps, to improve predictive accuracy without incurring excessive computational overhead. By systematically analysing different configurations, we arrived at an optimal setup that aligned better with the complex social scenarios of SocNavGym. These findings guided

our subsequent research decisions, especially in designing more advanced predictive models and adaptive mechanisms, as described in later chapters.

2.5 Setting Up the Experimental Environment

We used `SocNavGym-v0` a prototype-stage environment designed for social navigation. The setup consisted of two mobile humans (treated as obstacles (red)) and a robot agent with its goal. We deliberately limited the scenario to two humans to create a simplified, controlled environment that enables isolated evaluation of the reward function's sensitivity to human-robot spatial interactions. Figure 2.2 shows the initial configuration, highlighting two different perspectives: the world coordinate system and the robot coordinate system.

Figure 2.2: Comparative illustration of the initial environment setup in `SocNavGym-v0`, showing the world coordinate system (left) and the robot coordinate system (right). The world coordinate system provides a global reference frame, whereas the robot coordinate system is agent-centric, simplifying state representation and decision-making.

In Figure 2.2, the world coordinate system (left) offers a consistent global view but may not align naturally with the agent's perspective, whereas the robot coordinate system (right) is centered on the robot, potentially easing motion planning in RL applications.

Evaluating the Reward Function

The reward function is critical for guiding the agent towards safe and socially compliant navigation, we prioritised its design and validation. Below, we detail its evolution from a simple structure to a complex model integrating social norms.

Initial Simple Reward Function

In our intial experiments, we defined and used a basic reward function common in navigation tasks:

$$R = \begin{cases} +1 & \text{if goal is reached} \\ -1 & \text{if collision with human or obstacle occurs} \\ 0 & \text{otherwise.} \end{cases}$$

This design incentivised reaching the goal and avoiding collisions but treated humans as mere obstacles, ignoring social norms such as maintaining personal space (proxemics) or respecting human comfort.

2.5.1 Initial Experiments with SAC

To investigate the impact of the reward funtion and the coordinate systems on agent performance, we trained a Soft Actor–Critic (SAC) agent using the world coordinate system for 100 000 episodes. The agent achieved an average reward of approximately -0.65 with a standard deviation of 0.15, and a success rate (goal reached) of less than 10%, indicating frequent collisions and limited learning progress.

Hypothesising that the robot-coordinate system might simplify observations, we trained the SAC agent for an extended 3.5 million episodes. The average reward was approximately -0.60 with a standard deviation of 0.12, and the success rate improved marginally to 15%. Despite the prolonged training, no significant progression towards higher returns was observed, suggesting challenges beyond the coordinate system, such as environment stability or reward-integration issues.

Consequently, the agent exhibited unnatural behaviours, such as passing too close to humans, which could cause discomfort in real-world settings. This limitation highlighted the need for a reward function that better captures social-navigation requirements.

The simple reward function's failure to model social norms led to poor performance in socially complex scenarios. Social navigation requires balancing task efficiency with human comfort, necessitating a reward structure that penalises **behaviours** causing discomfort, such as abrupt movements or proximity violations. To address this, we adopted the Socially-

aware Navigation Graph Neural Network (SNGNN) [100], which generates data-driven cost maps to quantify human discomfort based on the robot's position, velocity, and orientation relative to humans. This approach aligns with the thesis's focus on social navigation and responds to feedback calling for detailed social-norm **modelling**.

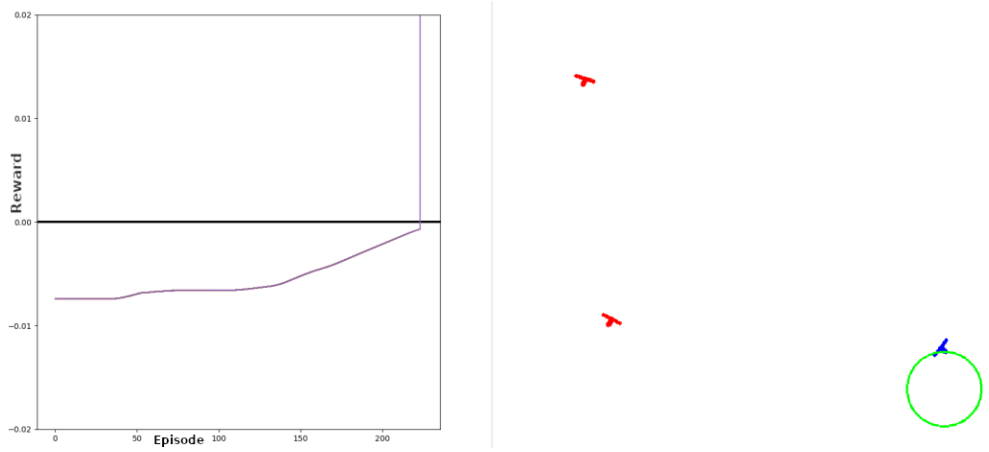


Figure 2.3: A manual joystick test revealed that the reward increases as the agent approaches its goal, suggesting our reward design is not strictly negative.

These tests imply the reward function itself, though computationally complex (due to the SNGNN's data-driven cost maps), was not obviously at fault. However, the agent's learning remained stagnant, which led us to suspect potential issues internal to `SocNavGym-v0` for example, unreliable collision detection or incorrectly updated environment states. Though we could not isolate a definitive software bug (lack of reproducible logs or code tracebacks), the persistent learning failures despite a seemingly valid reward scheme prompted us to conclude that the `SocNavGym-v0` challenge is beyond our research scope.

Our initial experiments in `SocNavGym-v0` highlighted challenges in training RL agents for social navigation. The updated SNGNN-based reward function effectively incorporated social norms, as validated by manual tests, but the agent's persistent poor performance pointed to limitations in the prototype environment, such as unreliable collision detection. These findings motivated a transition to `SocNavGym-v1`, which offers improved stability for further experimentation, as discussed in subsequent sections.

2.6 Transitioning from `SocNavGym-v0` to `SocNavGym-v1`

After our initial experiments in SocNavGym-v0 yielded inconsistent results, we hypothesised that instability within this prototype environment contributed to the agent's failure to learn robustly, even after extensive training. Our experiments (see Section 2.4) revealed unreliable collision detection, inconsistent state updates and other issues inherent to `v0`, which was still under active development at the time. These shortcomings severely hindered a fair assessment of our SNGNN-based reward function [100], whose purpose is to integrate social norms such as proxemics and human-comfort preservation. Whilst manual joystick tests confirmed that the reward function behaved as intended, the agent still struggled with collision-avoidance and goal-reaching, pointing to environment-level deficiencies rather than reward-design flaws.

We therefore adopted SocNavGym-v1, designed with improved stability and more consistent feedback loops. We also expanded our experiments to use additional model components, specifically: A Variational Autoencoder (VAE), to compress high-dimensional environment observations into manageable latent representations. An LSTM network, to capture sequential dependencies such as human trajectory patterns.

By combining these elements, we hoped to form a more reliable pipeline that could learn stable navigation policies without the inconsistencies observed in `SocNavGym-v0`. To train these models, we gathered a dataset of 1,000 episodes in `SocNavGym-v1`, which provided sufficient variety in human-robot interactions for initial evaluations. Details of how the VAE and LSTM were integrated with our RL agent are discussed in Chapter 5.

VAE Testing in SocNavGym-v1

We first tested the **VAE**'s ability to reconstruct environment observations, verifying it could encode and decode the robot's perspective. Figure 2.4 compares real vs. reconstructed images. Ideally, the reconstructed output should be gray if the VAE accurately reconstructed the observation; any other color indicates reconstruction errors. Overall, the VAE performed well in reducing the observation dimensionality.

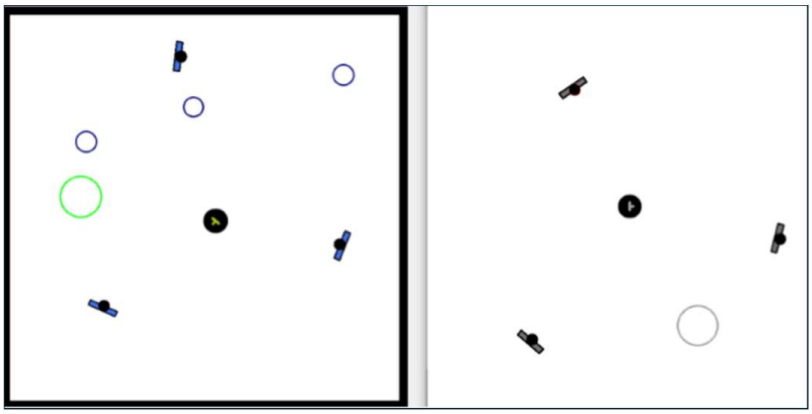


Figure 2.4: Observation from the environment (left) vs. VAE reconstruction (right). A black image implies successful encoding of scene details without artifacts.

Optimising Timestep and Window Size in SocNavGym

Finally, we investigated how different window slides (the number of prior states used) and timesteps (the granularity of each prediction step) influence performance.

Timestep Selection

Most movement predictions use simplified curves such as straight lines or simple parabolas, and the timestep used is determined by the amount of error one is willing to tolerate. Selecting the best timestep is a trade-off: using a smaller timestep allows more precise tracking of the environment's dynamics, but it consumes more computational resources and increases round-off error. Larger timesteps reduce computational overhead but compromise temporal resolution.

Window Size Selection

The LSTM model's ability to make contextual and accurate predictions depends on its access to prior temporal context. The window size parameter determines how many past states the model considers for each prediction. For example, a window size of 16 means that the model uses the previous 15 states along with the current one to generate the next prediction.

To determine the optimal combination of timestep and window size, we conducted experiments across a matrix of values and recorded average loss over the final training phase. The lowest average prediction loss was consistently observed when using a timestep of 0.25,

regardless of window size. Specifically:

- With a timestep of 0.25, average losses ranged from 0.00010 across all window sizes (1, 4, 8, 16).
- Timestep 0.5 yielded slightly higher losses around 0.00013–0.00014.
- Timestep 1.0 produced average losses around 0.00020–0.00021.
- Timestep 2.0 showed the poorest performance, with losses between 0.00029 and 0.00030.

Interestingly, the impact of window size was comparatively minor when using small timesteps. The differences in performance were more pronounced at coarser timesteps, suggesting that smaller timesteps can mitigate limitations in context length. However, reducing the timestep also increases computational overhead, which must be balanced against the modest accuracy gains.

Our analysis indicates that while lower timesteps, particularly 0.25, consistently yield better predictive performance (average loss ≈ 0.00010), the choice of window size has diminishing returns beyond size 8. For our implementation, a timestep of 1 combined with a window size of 16 offers a practical compromise between computational efficiency and accuracy, and this configuration was used in our final world model implementation.

Within `SocNavGym-v1`, we also evaluated our LSTM model visually by comparing the predicted next state to the actual future state, as seen in Figure 2.5.

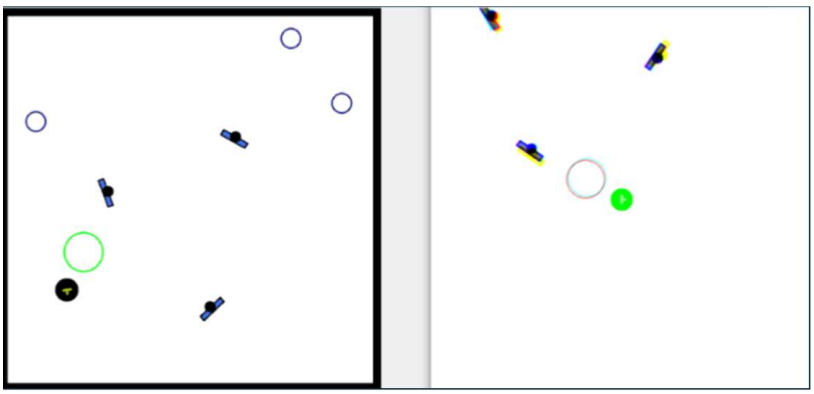


Figure 2.5: Left: actual next state; right: LSTM-predicted next state. Yellow denotes the current state, blue the actual future state, and red any mismatches.

We found that window size significantly affects the LSTM's performance: a larger window captures more context, improving predictive accuracy. However, larger windows also increase the computational load. In `SocNavGym-v1`, we observed analogous trends: expanding the temporal context enabled more robust predictions but required greater memory and training time.

Our transition from `SocNavGym-v0` to `SocNavGym-v1`—along with integrating a VAE for dimensionality reduction and an LSTM for sequence modeling—proved essential for establishing a more stable social navigation framework. Although manual joystick tests indicated that the reward function was not inherently flawed, persistent suboptimal results in `SocNavGym-v0` pointed to environment-level inconsistencies. By contrast, `SocNavGym-v1` provided more reliable feedback loops and allowed us to adjust key hyperparameters (timestep and window slide) for improved predictive accuracy.

Lower timesteps (around 0.25) generally yielded more precise short-horizon predictions, though at a higher computational cost, while selecting an effective window slide (such as 16) balanced contextual information with efficiency. The VAE condensed high-dimensional observations for faster policy training, and the LSTM improved temporal prediction in human-robot interactions. Overall, these findings validate our approach to data-driven modeling for social navigation and emphasize the importance of environment stability in reinforcement learning pipelines.

Having laid this groundwork, we now turn to Chapter 5, which delves into Predictive World Models for Social Navigation. There, we introduce and evaluate advanced models such as 2StepAhead, MASPM, and 2StepAhead-MASPM, building on our SocNavGym-v1 experiences to achieve more robust and human-centric navigation.

Chapter 3

Basics of Deep Neural Networks

3.1 Introduction

Reflecting the primary objective outlined in Section 1.2, this chapter establishes the foundations of deep neural networks (DNNs) that form the basis for the methods used in our thesis. We begin with the essential concepts of feedforward neural networks, activation functions, and the fundamentals of optimization. We then move on to recurrent neural networks (RNNs), particularly LSTMs, which are important for sequence prediction tasks, followed by the Transformer architecture used as a baseline for comparisons in certain tasks.

Throughout this chapter, we emphasize only those neural network techniques and principles that are directly relevant to our research.

3.2 Multi-Layer Perceptrons

At the core of many deep neural networks (DNNs) lies the Multi-Layer Perceptron (MLP), which evolves from Frank Rosenblatt's original perceptron model [102]. An MLP typically consists of one or more hidden layers, each comprising multiple perceptron-like units with learnable weights and biases. By integrating nonlinear activation functions, MLPs can approximate a wide range of functions [28].

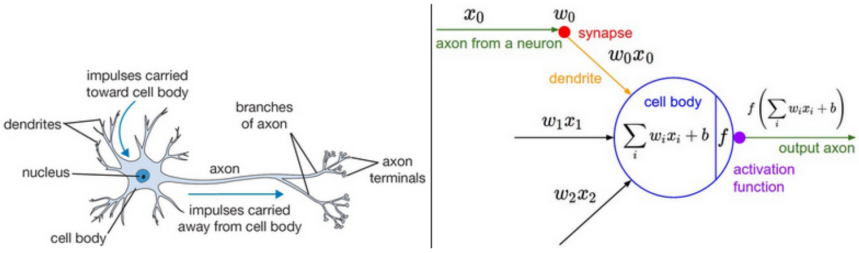


Figure 3.1: Comparison of a biological neuron (left) and the basic unit of an MLP, the Perceptron (right). Image source: Stanford University's CS231n Course Notes on Neural Networks. Available at: `https://cs231n.github.io/neural-networks-1/`

3.2.1 Activation Functions

Activation functions introduce nonlinearity, enabling the network to model complex relationships. Common activation functions include:

Sigmoid [91]:

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}, \tag{3.1}$$

The sigmoid function maps any real-valued input z to a value between 0 and 1. This property makes it especially useful for binary classification tasks where the output can be interpreted as a probability. Additionally, sigmoid functions are often employed in gating mechanisms within neural networks. However, a notable drawback is that sigmoid functions can cause vanishing gradients during backpropagation in deep networks, which hampers the learning process.

Tanh [67]:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{3.2}$$

The hyperbolic tangent function outputs values in the range $[-1, 1]$. Unlike the sigmoid function, tanh centers the data around zero, which can lead to faster convergence during training. It is commonly used in hidden layers of neural networks to introduce nonlinearity while maintaining symmetry around the origin. Similar to the sigmoid function, tanh can also suffer from vanishing gradients, though to a lesser extent.

ReLU (Rectified Linear Unit) [27]:

$$\text{ReLU}(z) = \max(0, z). \tag{3.3}$$

The ReLU activation function outputs the input directly if it is positive; otherwise, it outputs zero. This simple nonlinearity has become the default activation function for many modern DNNs due to its computational efficiency and ability to mitigate the vanishing gradient problem. By allowing only positive values to pass through, ReLU helps in maintaining sparse activations, which can lead to more efficient learning and better generalization.

3.2.2 Backpropagation and Adam

MLP training is generally approached through gradient-based optimizers:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}, \tag{3.4}$$

where α is the learning rate, and $\mathcal{L}$ is a loss function. This equation represents the basic gradient descent update rule, where the weights $\mathbf{w}$ are adjusted in the direction that minimally decreases the loss. The term $\nabla_{\mathbf{w}} \mathcal{L}$ denotes the gradient of the loss function with respect to the weights, calculated using the backpropagation algorithm [103]. Backpropagation efficiently computes these gradients by propagating the error backward through the network layers, allowing each weight to be updated appropriately to minimize the overall loss.

Backpropagation [103] computes gradients with respect to each parameter.

Adam [61], a widely used optimizer, adaptively scales learning rates for each parameter:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) \nabla_{\mathbf{w}} \mathcal{L}, \tag{3.5}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) \left(\nabla_{\mathbf{w}} \mathcal{L}\right)^2, \tag{3.6}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{v_t / \left(1 - \beta_1^t\right)}{\sqrt{s_t / (1 - \beta_2^t)} + \epsilon}. \tag{3.7}$$

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) \nabla_{\mathbf{w}} \mathcal{L}, \tag{3.8}$$

This equation 3.2.2 updates the first moment estimate v_t, which represents the exponentially decaying average of past gradients. The parameter β_1 controls the decay rate of these past gradients, allowing the optimizer to maintain a running average that smooths out the

gradient updates over time.

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) \left(\nabla_{\mathbf{w}} \mathcal{L}\right)^2, \tag{3.9}$$

Here, s_t is the second moment estimate, representing the exponentially decaying average of past squared gradients. The parameter β_2 determines the decay rate, helping to account for the variability in the gradients and ensuring stable updates by adapting the learning rate for each parameter individually.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{v_t / \left(1 - \beta_1^t\right)}{\sqrt{s_t / \left(1 - \beta_2^t\right)} + \epsilon}. \tag{3.10}$$

This is the parameter update rule for the Adam optimizer. It adjusts the weights $\mathbf{w}$ by subtracting a term that scales the first moment estimate v_t by the square root of the bias-corrected second moment estimate s_t. The term ϵ is a small constant added for numerical stability to prevent division by zero. This adaptive scaling allows Adam to perform efficient and reliable updates, especially in scenarios with sparse gradients or noisy data.

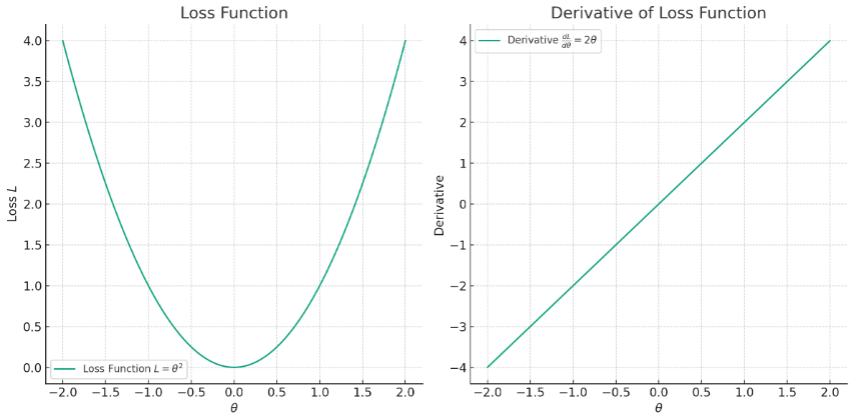


Figure 3.2: Visualization of a one-dimensional loss function and its derivative in the context of gradient descent.

In our experiments, we rely primarily on Adam for all MLP-based modules.

3.3 Recurrent Neural Networks (RNNs)

For tasks such as time-series prediction or language modeling—both important in our predictive world models—Recurrent Neural Networks (RNNs) handle variable-length se-

quences [92]. At each timestep t, an RNN maintains a hidden state h_t influenced by the previous hidden state h_{t-1} and the current input x_t. This enables the network to capture temporal dependencies in data.

Vanilla RNNs often face vanishing or exploding gradients when processing long sequences [94], which affect the model's ability to learn long-term dependencies. This is a key reason for adopting Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRUs).

3.3.1 Long Short-Term Memory (LSTM)

The Long Short-Term Memory (LSTM) network [48] mitigates vanishing gradients, introducing a cell state C_t that can propagate information across many timesteps. It comprises three gating mechanisms:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{3.11}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{3.12}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{3.13}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{3.14}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{3.15}$$

$$h_t = o_t * \tanh(C_t). \tag{3.16}$$

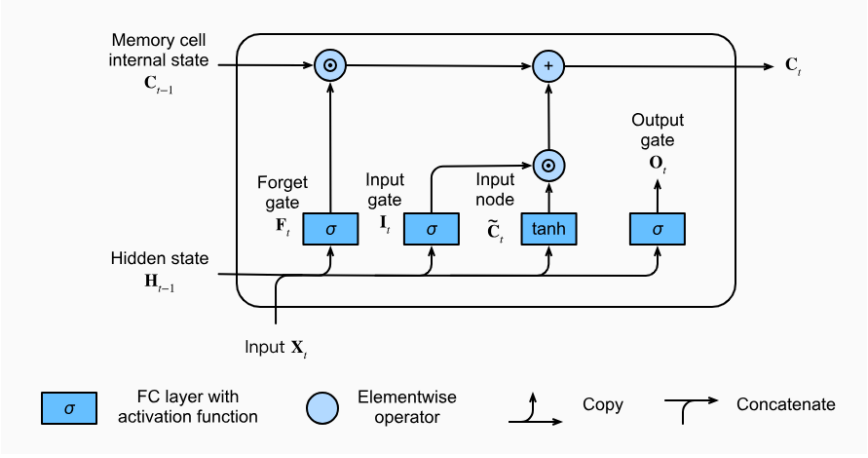


Figure 3.3: Architecture of an LSTM unit, showing forget/input/output gates and the cell state. Image adapted from Zhang et al. [148].

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{3.17}$$

This equation 3.3.1 defines the forget gate f_t at timestep t. The sigmoid activation σ ensures that the gate values are between 0 and 1. The forget gate determines how much of the previous cell state C_{t-1} should be retained. By multiplying f_t with C_{t-1}, the network can effectively forget irrelevant information from the past.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{3.18}$$

The input gate i_t controls the extent to which new information from the current input x_t and the previous hidden state h_{t-1} should be incorporated into the cell state. The sigmoid activation ensures the gate values are between 0 and 1, allowing the network to regulate the flow of new information.

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{3.19}$$

This equation 3.3.1 computes the candidate cell state $\tilde{C}_t$ using the hyperbolic tangent activation function. It generates new information that could be added to the cell state, ensuring that the values are bounded between -1 and 1. This candidate is then modulated by the input gate to update the cell state.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{3.20}$$

The cell state C_t is updated by combining the retained information from the previous cell state C_{t-1}, modulated by the forget gate f_t, with the new candidate information $\tilde{C}_t$, scaled by the input gate i_t. This mechanism allows the LSTM to maintain and update its memory effectively over long sequences.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{3.21}$$

The output gate o_t determines how much of the cell state should be exposed to the hidden state h_t. By applying the sigmoid activation, the gate controls the proportion of the cell state that contributes to the output, allowing the network to regulate the flow of information from the cell to the hidden state.

$$h_t = o_t * \tanh(C_t). \tag{3.22}$$

The hidden state h_t is computed by applying the hyperbolic tangent activation to the cell state C_t, and then scaling it by the output gate o_t. This ensures that the hidden state captures the relevant information from the cell state while being regulated by the output gate to prevent the propagation of irrelevant or excessive information.

As illustrated in Figure 3.3, the LSTM unit maintains a cell state that is modulated by the forget, input, and output gates. This structure allows the LSTM to retain or discard information over long sequences, effectively capturing long-term dependencies.

LSTMs form the basis for the Cosine-Gated LSTM (CGLSTM) introduced in Chapter 6, where we improve the gating mechanisms using cosine similarity to better handle outliers and long-term context.

3.3.2 Gated Recurrent Units (GRUs)

An alternative to LSTMs is the **GRU** [18], which merges the forget and input gates into a single update gate z_t:

$$z_t = \sigma(W_z[h_{t-1}, x_t]), \tag{3.23}$$

$$r_t = \sigma(W_r[h_{t-1}, x_t]), \tag{3.24}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]), \tag{3.25}$$

$$h_t = (1 - z_t)\, h_{t-1} + z_t\, \tilde{h}_t. \tag{3.26}$$

$$z_t = \sigma(W_z[h_{t-1}, x_t]), \tag{3.27}$$

The update gate z_t controls the extent to which the hidden state h_t should be updated with new information. By applying the sigmoid activation function, z_t takes values between 0 and 1, allowing the GRU to balance between retaining the previous hidden state and incorporating the new candidate hidden state.

$$r_t = \sigma(W_r[h_{t-1}, x_t]), \tag{3.28}$$

The reset gate r_t determines how much of the previous hidden state h_{t-1} should be forgotten when computing the candidate hidden state $\tilde{h}_t$. This gate allows the GRU to selectively ignore irrelevant past information, enabling the network to capture relevant dependencies more effectively.

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]), \tag{3.29}$$

The candidate hidden state $\tilde{h}_t$ is computed by applying the hyperbolic tangent activation to a linear combination of the reset-modulated previous hidden state $r_t * h_{t-1}$ and the current input x_t. This candidate state represents the new information that could be added to the hidden state.

$$h_t = (1 - z_t)\, h_{t-1} + z_t\, \tilde{h}_t. \tag{3.30}$$

The new hidden state h_t is a linear interpolation between the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$, controlled by the update gate z_t. This allows the GRU to retain long-term dependencies by keeping a portion of the previous hidden state while also integrating new information, providing a balance between memory and adaptability.

As shown in Figure 3.4, the GRU combines the functions of the forget and input gates into a single update gate, simplifying the architecture compared to LSTM while still effectively managing long-term dependencies. This can be simpler than LSTM and is also used in some baselines. However, our thesis primarily focuses on LSTMs and further extends them to CGLSTM.

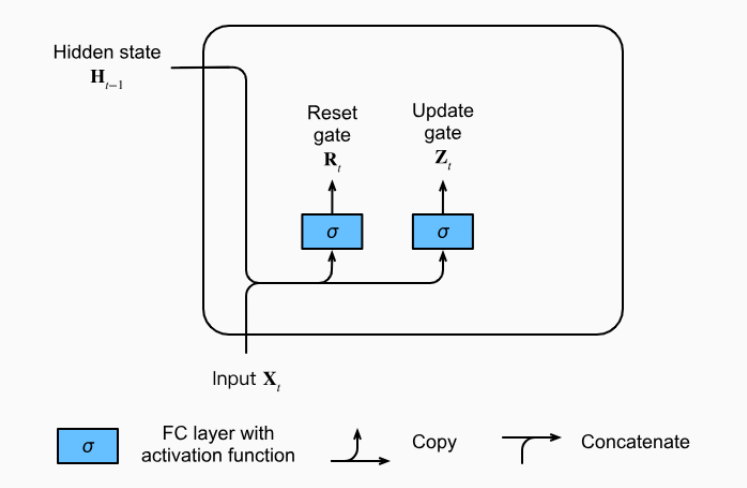


Figure 3.4: Overview of a GRU cell illustrating the reset gate r_t and update gate z_t.

3.4 Transformers

Transformers [134] have revolutionised sequence modelling by replacing recurrence with self-attention mechanisms. Key concepts include the encoder-decoder structure shown in Figure 3.5, which illustrates the use of multi-head attention and positionwise feed-forward layers repeated across both encoder and decoder components.

3.4.1 Self-Attention and Multi-Head Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^\top}{\sqrt{d_k}}\right) V, \tag{3.31}$$

This equation 3.4.1 defines the scaled dot-product attention mechanism used in Transformers. Here, Q, K, and V are the query, key, and value matrices, respectively, derived from the input embeddings. The dot product $Q\,K^\top$ computes the similarity between queries and keys, which is then scaled by $\sqrt{d_k}$ to stabilize gradients. The softmax function converts these scores into attention weights that sum to one, allowing the model to weigh the values V accordingly. This mechanism enables the model to focus on different parts of the input sequence when generating each output element.

Multi-head attention splits these projections into multiple heads, capturing diverse relationships in parallel.

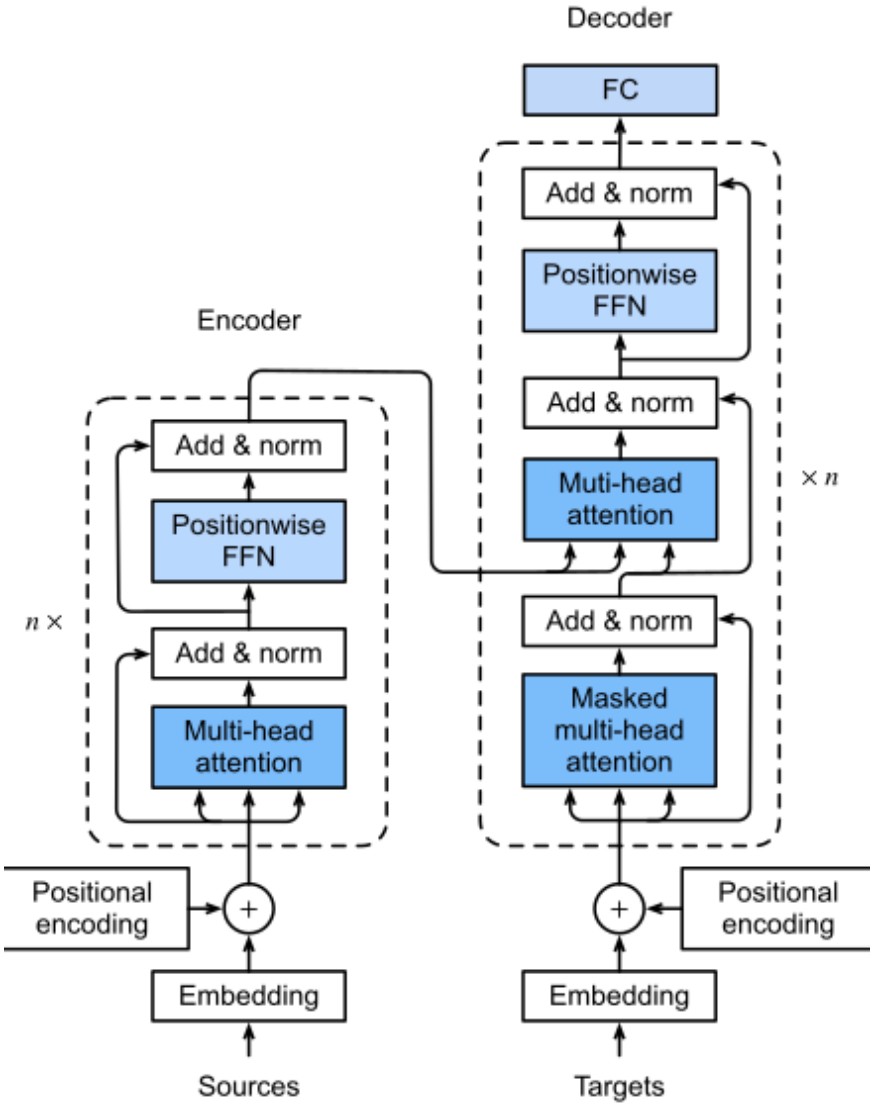


Figure 3.5: Schematic of the Transformer encoder. Multi-head self-attention and feed-forward layers are repeated, with positional encodings added to the inputs. Image adapted from Vaswani et al. [134].

3.4.2 Positional Encoding

Since Transformers process all elements in parallel, *positional encodings* are added to the input embeddings to inject order information. This allows the model to capture the sequence's order, which is inherently important for tasks involving sequential data.

In Chapter 6, we compare some of our tasks (like row-wise MNIST or next-state observation in SocNavGym) against an Encoder Transformer baseline. This helps illustrate how advanced sequence models can outperform or differ from recurrent architectures such as LSTMs. Transformers serve primarily as a benchmark for evaluating our sequence modeling improvements.

In Chapter 6, we introduce the Cosine-Gated LSTM (CGLSTM) to address the limitations of standard LSTMs in capturing long temporal dependencies and managing outliers. The concepts of gating and hidden-state updates described in this chapter serve as the foundation for that extension.

In this chapter, we focus on the concepts of neural networks that are relevant to our thesis. We introduced Multi-Layer Perceptrons (MLPs) for feedforward tasks, recurrent neural networks (RNNs), especially LSTMs and GRUs, to handle sequential data, and provided an overview of the transformer architecture, which later serves as a baseline for sequence modeling tasks.

Chapter 4

Fundamentals of Reinforcement Learning

This chapter provides an overview of Reinforcement Learning (RL), focusing on its foundational concepts, key algorithms, and their relevance to this thesis. We introduce the roles of agents and environments, outline the Markov Decision Process (MDP), and explore the RL algorithms used in our thesis, highlighting their features, limitations, and applications.

4.1 History of Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning that combines insights from psychology, neuroscience, and computer science, RL has become a fundamental area of artificial intelligence (AI). The idea of trial-and-error learning can be traced to Edward Thorndike's Law of Effect (1911), which proposed that behaviors leading to satisfying outcomes are more likely to be repeated [129]. This principle paved the way for key components in RL, where an agent's actions are reinforced by reward signals. In the 1920s, Ivan Pavlov introduced the concept of conditioned reflexes, using the term "reinforcement" to describe how stimuli could shape responses [95]. Together, Thorndike's trial-and-error framework and Pavlov's work on reinforcement in animal learning would later inform the reward-based learning principles fundamental to RL.

The mid-20th century saw groundbreaking ideas on how machines could learn from experience. In 1948, Alan Turing suggested a "pleasure-pain system" that resembled reward and punishment for machine learning [130]. Later, in his 1950 landmark paper, Turing again emphasized the possibility of machines learning through reinforcement signals [131]. Around the same time, John von Neumann and Stanislaw Ulam developed the Monte Carlo Method, showing the power of simulation and sampling for computational experiments [136]. Another notable contribution came from Claude Shannon, whose mechanical mouse (1952) demonstrated maze-solving through trial-and-error [113].

In the 1950s, mathematical foundations for optimal decision-making were laid by Richard Bellman, who introduced the concept of optimal control and the Bellman equation [7, 5]. Bellman's work led to Dynamic Programming (DP) methods, which iteratively compute optimal policies by breaking down problems into subproblems [8]. In parallel, Ronald Howard's (1960) policy iteration method offered an alternative DP-based solution for what would become known as Markov Decision Processes (MDPs) [52]. These developments -based on value functions, states, and returns - remain core to modern RL theory [97].

The 1960s brought further attention to machine learning through neural networks and trial-and-error methods. Early researchers such as Marvin Minsky [81], Waltz and Fu [137], and Mendel [78] explored how machines could learn adaptive behaviors. Around the same time, work by Michie and Chambers (1968) introduced GLEE and the BOXES controller, demonstrating trial-and-error learning in tasks like tic-tac-toe and pole balancing [79]. Harry Klopf's investigations in the 1970s into trial-and-error research shed light on the differences between reinforcement and supervised approaches [62]. Building on these ideas, Sutton (1978) extended animal learning theories to computational frameworks [121], laying early groundwork for what would soon be called temporal-difference (TD) learning.

Despite DP's solid theoretical basis, its computational demands and the need for accurate environment models limited its applicability. To address these issues, Werbos (1977) proposed approximate DP, known as heuristic dynamic programming [144]. The RL breakthrough occurred in 1989, when Christopher Watkins introduced Q-learning, an off-policy method for learning value functions directly from trial-and-error interactions with the environment, without a model of the system's dynamics [143]. Later, Watkins and Dayan

(1992) provided a formal convergence analysis for Q-learning [142], paving the way for new possibilities for model-free RL. Around the same period, extensive work on neurodynamic programming by Bertsekas and Tsitsiklis (1996) bridged the gap between DP, function approximation, and neural networks [9].

Modern RL research was furthered by the development of temporal-difference (TD) learning, which updates value estimates based on the difference between successive predictions. Arthur Samuel (1959) had provided early illustrations of TD by using self-play for checkers [105], but TD methods were formalized as a central RL concept by Sutton and Barto in the 1980s, laying the path toward TD-Gammon [127].

In the 2010s, the convergence of deep neural networks with RL often termed Deep Reinforcement Learning (DRL) produced breakthroughs. Mnih et al. demonstrated the Deep Q-Network (DQN), which surpassed human-level performance on multiple Atari 2600 games by learning from raw pixels [82, 83]. Building on DQN, Google DeepMind further advanced DRL with AlphaGo, combining deep RL and Monte Carlo Tree Search to defeat a world champion in the game of Go [116]. Subsequent systems like AlphaZero and MuZero extended these techniques across multiple domains [117, 107], Demonstrating the remarkable generality of DRL.

For a more comprehensive historical perspective, Sutton and Barto's "Reinforcement Learning: An Introduction" [124] is recommended. This text traces RL's evolution, from foundational concepts to contemporary advancements that blend deep learning with classical RL theories.

4.2 The Reinforcement Learning Problem

In Reinforcement Learning (RL), an agent interacts with an environment by mapping states to actions in a way that maximises cumulative rewards. The agent receives feedback in the form of rewards and adjusts its behaviour over time to optimise performance. This interaction process is formally described by a Markov Decision Process (MDP), which consists of a set of states ($\mathcal{S}$), a set of actions ($\mathcal{A}$), transition probabilities ($\mathcal{P}$) that define the likelihood of moving between states, and a reward function ($\mathcal{R}$) that assigns values to state-action

transitions. The typical interaction loop between the agent and the environment is illustrated in Figure 4.1, where the agent observes the current state S_t, selects an action A_t, and receives a reward R_t, after which the environment transitions to a new state S_{t+1}.

A policy π governs the agent's decision-making process, defining the action the agent will take in each state. The goal is to find an optimal policy π^* that maximizes the expected return, which is the sum of discounted future rewards. The MDP framework helps in making decisions that balance immediate and future rewards by leveraging the Markov property (where future states depend only on the current state and action).

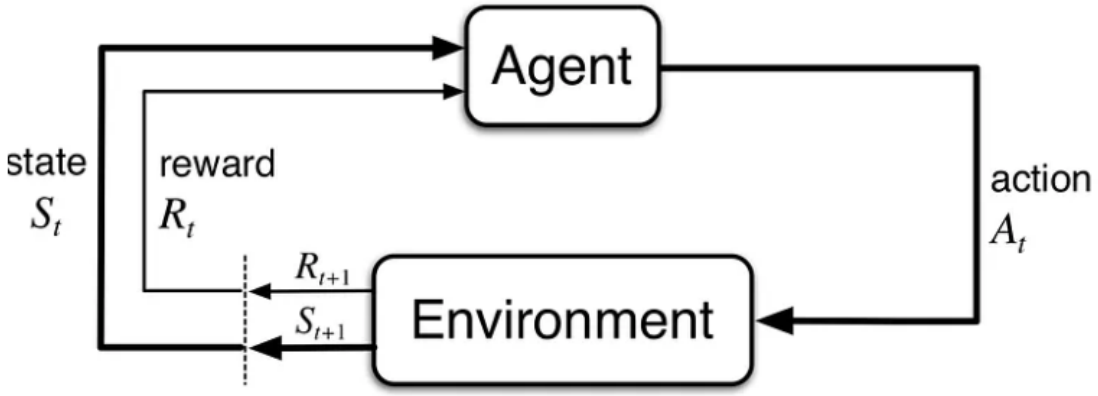


Figure 4.1: Interaction loop between the agent and the environment in an RL setting. The agent observes the state S_t, chooses an action A_t, and receives a reward R_t. The environment transitions to the next state S_{t+1}, influenced by the action taken. This figure is adapted from "Reinforcement Learning: An Introduction" by Sutton and Barto [124].

Environment and Markov Decision Processes

An RL environment can be either real or virtual, comprising states, actions, transitions between these states, and rewards associated with these transitions. These rewards guide the learning process of an RL agent. At the core of RL is the Markov Decision Process (MDP), defined by its state space ($\mathcal{S}$), action space ($\mathcal{A}$), transition probabilities ($\mathcal{P}$), reward structure ($\mathcal{R}$), and initial state distribution ($\mathcal{P}_0$). In MDPs, the future state depends on the current state and the action taken, following the Markov property, which ensures that predictions and decisions are made based only on the current observable state [97].

Reward Optimization and Value Functions

In RL, agents learn through a system of rewards and penalties. Negative rewards denote undesirable behaviors, while positive rewards encourage favorable actions. Agents aim to optimize their cumulative reward R_t, which is the total sum of rewards obtained over time, adjusted by a discount factor γ to balance immediate and future rewards:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{4.1}$$

In Equation 4.1, γ is the discount factor, ranging from 0 to 1. This factor devalues future rewards compared to immediate ones, reflecting the uncertainty and time value of future rewards. In stochastic environments, discounting helps stabilize the learning process, emphasizing short-term gains while considering long-term outcomes [124].

The goal in RL is to find an optimal policy π^* that maximizes the expected return from each state. The Bellman optimality equations calculate the state-value function $V^*(s)$ under an optimal policy and the optimal action-value function $Q^*(s, a)$:

$$V^*(s) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s] \tag{4.2}$$

$$Q^*(s, a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a] \tag{4.3}$$

The state-value function $V^*(s)$ represents the expected return starting from state s and following the optimal policy, while the action-value function $Q^*(s, a)$ represents the expected return after taking action a in state s under the optimal policy. Both functions are essential for RL algorithms like Q-Learning [142], which seek to estimate these optimal values. Through iterative updates, these algorithms adjust their estimates of $V^*(s)$ and $Q^*(s, a)$ based on observed rewards and future states, thereby improving the policy towards optimality [141].

4.3 Model-Free vs. Model-Based Reinforcement Learning

Reinforcement Learning strategies are broadly categorized into Model-Free and Model-Based types.

Model-Free Reinforcement Learning

Model-Free Reinforcement Learning interacts directly with the environment to learn the value of actions and states through trial and error, without assuming any knowledge of the environment's dynamics [122]. Model-Free RL often requires numerous interactions for effective learning.

Temporal-Difference Learning

Temporal-Difference Learning (TD), within the Model-Free aproach, updates value estimates based on differences between subsequent estimates [122]. Merging ideas from Monte Carlo methods and dynamic programming [6], it facilitates efficient learning directly without a model of the environment's dynamics. TD Learning is fundamental in algorithms like Q-Learning [142] and state-action-reward-state-action (SARSA) [149] for estimating state-value functions and refining policies towards optimality [123].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{4.4}$$

Equation 4.4 shows the Q-learning update rule in a TD Learning approach, where $Q(s, a)$ represents the action value in state s, α is the learning rate, and γ is the discount factor [141].

Model-Based Reinforcement Learning

Model-Based Reinforcement Learning involves using a model of the environment to predict future states and rewards [122]. This approach improves planning and predicting, allowing agents to anticipate the outcomes of their actions before executing them. This can lead to a more efficient learning process, reducing the number of interactions required with the environment, a concept known as sample efficiency (the ability to learn effectively with fewer interactions) [33]. Although complex, Model-Based RL is advantageous in situations where

understanding the environment's dynamics or achieving sample efficiency is important [33].

By relying on an internal model, Model-Based RL enables strategic decision-making, potentially reducing the number of interactions needed for learning [108].

$$s_{t+1}, r_t = f(s_t, a_t) \tag{4.5}$$

Equation 4.5 represents the predictive model of Model-Based RL, where s_t is the current state, a_t the action, s_{t+1} the subsequent state, and r_t the reward received [31]. Figure 4.2 illustrates a model-based RL agent's interaction with the environment, using a predictive model to anticipate future states. This process facilitates strategic planning, reducing unnecessary exploration and exemplifying the core principles of Model-Based RL.

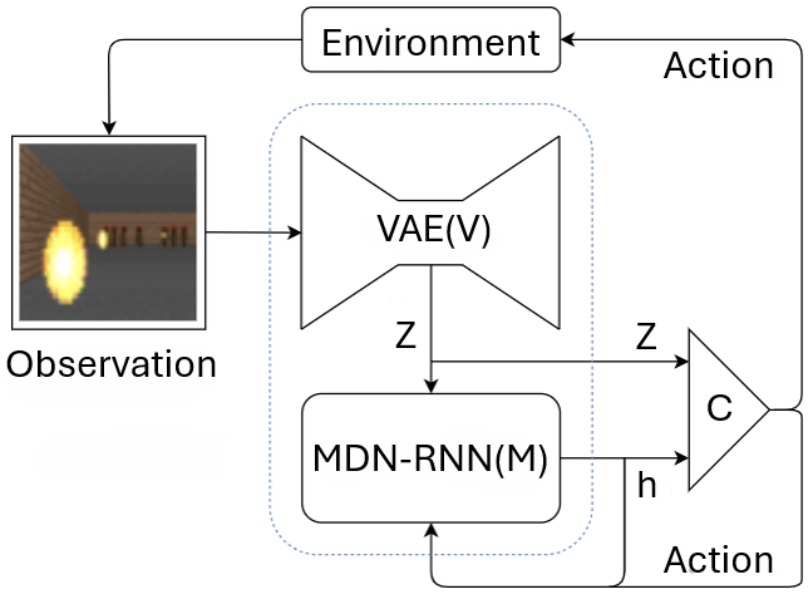


Figure 4.2: An example of a model-based RL agent interacting with the environment, using a predictive model to anticipate future states. The agent uses a variational autoencoder (VAE) to encode observations into a latent space z, which is then used by a mixture density network recurrent neural network (MDN-RNN) to predict the next state h and the corresponding action values c. (Adapted from "World Models" by Ha and Schmidhuber [33]).

While Model-Free RL is more straightforward and widely applicable than Model-Based RL, it often requires extensive interactions to learn effectively. Model-Based RL, on the other hand, offers the advantage of efficiency and predictions but at the cost of additional complexity and the need for accurate modeling. Choosing between the two approaches

depends on the specific requirements and constraints of the task at hand, including the availability of data, the need for sample efficiency (learning with fewer environment interactions), and the complexity of the environment.

As AI and Reinforcement Learning evolve, the distinction between Model-Free and Model-Based approaches becomes less rigid, with hybrid models and advanced algorithms leveraging the strengths of both approaches [14]. For instance, AlphaGo combines model-free deep reinforcement learning with model-based planning to achieve superior performance. In practice, these hybrid approaches may learn a partial model of the environment for planning while simultaneously employing model-free updates, as in MuZero, thus blurring the traditional separation between Model-Free and Model-Based RL.

For a deeper understanding of Model-Free and Model-Based methods, including TD Learning, Q-Learning, and predictive models, reader is refer to foundational texts like "Reinforcement Learning: An Introduction" by Sutton and Barto [123].

4.4 Policy Learning in Reinforcement Learning

Reinforcement learning (RL) strategies are often characterized based on their approach to policy learning, with two primary approaches being On-Policy Learning and Off-Policy Learning [125]. Each approach has distinct methods and implications for the learning process and the type of problems they are best suited to solve. On-Policy and Off-Policy differ in whether the behavior policy used for exploration matches the policy being learned. This choice can impact both the safety (minimizing exploratory errors that could lead to undesirable actions) and the sample efficiency of the learning process.

On-Policy Learning

On-Policy Learning focuses on evaluating and improving the policy that is directly used to make decisions within the environment. A key method within this approach is the State-Action-Reward-State-Action (SARSA) algorithm [149], which updates the policy based on the actions taken and the resulting rewards. These approaches are noted for their safe and consistent learning processes, especially in environments where errors are costly [149, 124]. For instance, SARSA is commonly applied in robotic navigation tasks where reducing

dangerous exploratory actions is critical [149].

Off-Policy Learning

Off-Policy Learning, on the other hand, involves learning a policy that is separate from the policy used for exploration. This allows agents to gather exploratory experiences using one policy while optimizing a different, typically more deterministic, target policy. Deep Q-Networks (DQN) [53] is an example of an Off-Policy Learning algorithm that learns an optimal policy from experience replay while using an exploratory behavior policy. This makes Off-Policy methods highly effective in large state spaces where extensive exploration is required. DQN, for example, has been successfully applied in mastering Atari games.

Choosing between On-Policy and Off-Policy approaches depends on the demands of the environment and the learning objectives. While On-Policy methods like PPO (Proximal Policy Optimization) offer safety and stability in robotic control, Off-Policy methods provide better sample efficiency and performance in domains requiring aggressive exploration, such as autonomous driving and video games.

4.5 RL Algorithms Used in this Thesis

This section provides an overview of the Reinforcement Learning (RL) algorithms used in this thesis, highlighting their features, limitations, and applications.

4.5.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) is a value-based RL algorithm that integrates Q-Learning with deep neural networks [99]. First introduced by Mnih et al. [83], DQN leverages a replay buffer and a target network to stabilize training when learning from high-dimensional inputs. DQN approximates the Q-value function, predicting the expected return from taking an action in a specific state:

$$Q(s, a; \theta) = \mathbb{E}\big[r + \gamma \max_{a'} Q(s', a'; \theta) \;\big|\; s, a\big]. \tag{4.6}$$

Equation 4.6 represents the Q-value function in DQN. It computes the expected return of taking action a in state s and then following the optimal policy thereafter. $Q(s, a; \theta)$ is the expected sum of rewards r discounted by γ at each future step, starting from the current

state-action pair. However, DQN can overestimate Q-values when the environment requires long-horizon predictions, which may lead to suboptimal policies [99]. A widely adopted solution for this issue is Double DQN [133], which decouples the action selection from the action evaluation to mitigate overestimation.

DQN also faces challenges in high-dimensional or continuous action spaces, because it typically relies on discretizing actions to function effectively. In spite of these limitations, DQN has achieved remarkable success across several domains. For instance, DQN famously reached human-level performance on multiple Atari 2600 games by learning directly from raw pixels [83], demonstrating its capacity to handle complex visual inputs. It has also been applied in discrete robotic control tasks, such as navigation or manipulation, where possible actions can be discretized [64]. Similarly, in autonomous driving, simplified maneuvers like braking or steering can be discretized, enabling DQN-based policies to learn safe driving strategies [112].

DQN excels in environments where the action space is inherently discrete, making it a suitable candidate for certain robotics, gaming, and navigational tasks. It also benefits from stable learning mechanisms such as experience replay and target networks [83], which reduce sample correlation and mitigate catastrophic forgetting. Moreover, DQN serves as a strong baseline in reinforcement learning, allowing straightforward comparisons with subsequent extensions (e.g., Double DQN or Dueling DQN) and providing a solid reference point for performance metrics.

The Dueling Deep Q-Network (Dueling DQN) [140] architecture is an improvement of the standard DQN that addresses some of its limitations by decomposing the Q-value function into two streams, one for estimating the state value function $V(s)$, and another for computing the action advantage function $A(s, a)$.

$$Q(s, a; \theta, \alpha, \beta) = V\big(s; \theta, \beta\big) + \Big(A\big(s, a; \theta, \alpha\big) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\big(s, a'; \theta, \alpha\big)\Big). \tag{4.7}$$

Equation 4.7 effectively decomposes the Q-value into two components: the value function

$V(s;\theta,\beta)$ and the advantage function $A(s,a;\theta,\alpha)$. Here, θ denotes the shared network parameters, while β and α indicate the distinct parameters for the value and advantage streams, respectively. By subtracting the average advantage of all actions a' from the advantage of action a, Dueling DQN stabilizes the training process and leads to more accurate estimations of state values, particularly in scenarios with many similar-valued actions. This architecture addresses some of the overestimation and inefficiency challenges associated with the vanilla DQN.

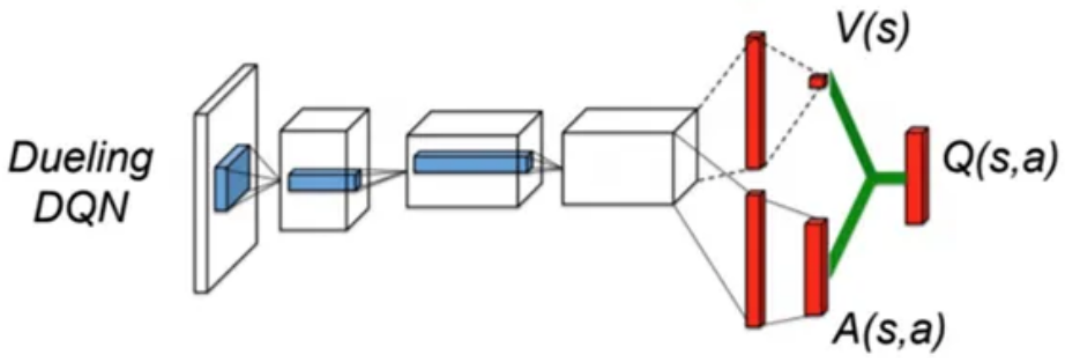


Figure 4.3: The Dueling DQN architecture, illustrating the value and advantage estimation streams. Image adapted from the Dueling Network Architectures for Deep Reinforcement Learning by Wang et al. [140].

Dueling DQN has shown significant performance improvements, particularly in gaming environments such as Atari 2600 games [140], where refined strategies are required. Its application also extends to robotics and autonomous systems, providing a more sophisticated approach for discrete action decision-making [140]. For an in-depth understanding of dueling DQN, the reader is referred to the work by Wang et al. [140].

4.5.2 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy actor-critic algorithm that leverages deep function approximators for environments with high-dimensional, continuous action spaces [68]. Like DQN, DDPG also uses experience replay and target networks but is specifically tailored to handle the continuous action domain. It extends the principles of Deep Q-Learning to these continuous domains, demonstrating significant success in over 20 simulated physics tasks, thereby demonstrating its effectiveness in handling complex environmental interactions [68].

DDPG's architecture is unique in how it efficiently handles continuous action spaces through its actor-critic mechanism. This dual-structure is important for balancing action selection (actor) and evaluating these actions (critic). The actor network in DDPG, $\mu(s|\theta^{\mu})$, is a deterministic policy network mapping states directly to actions [68]. Its update is based on the policy's gradient, calculated to maximize the expected return. This direct mapping to actions is what enables DDPG to operate in continuous action spaces, differing from methods requiring discretization. The critic network, $Q(s, a|\theta^{Q})$, estimates the action-value function [68]. Its update mechanism uses the Bellman equation, a fundamental component in temporal difference learning. Here, the critic network evaluates the actor's proposed actions, guiding the actor's policy update towards optimal actions.

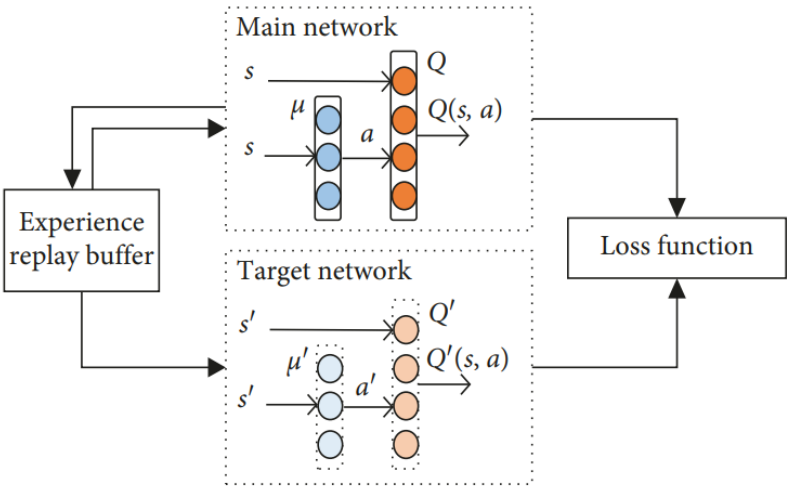


Figure 4.4: An overview of the Deep Deterministic Policy Gradient (DDPG) framework. The main network comprises a **blue actor** network, which outputs action μ from state s, and an **orange critic** network, which estimates $Q(s, a)$. The target network mirrors this structure, with a **blue target actor** μ' and an **orange target critic** Q'. Transitions (s, a, r, s') are sampled from the experience replay buffer and used to update both actor and critic through the loss function. Adapted from [13].

Deep Deterministic Policy Gradient (DDPG) benefits from the replay buffer by storing and reusing past experiences, Reducing the correlation between updates and improving sample efficiency [68]. Target networks, updated more slowly than the main networks, help stabilize training by providing consistent targets for the learning updates. Compared to naive policy gradient algorithms that can suffer from high variance in continuous domains, DDPG's off-policy design and use of target networks offer more stable learning and better sample efficiency. This approach collectively addresses the instability often encountered with large, non-linear function approximators, improving robustness and performance.

The Bellman equation for DDPG's action-value function, essential to its learning process, is expressed as:

$$Q^{\pi}(s,a) = \mathbb{E}_{r_t, s_{t+1} \sim E}\Big[r(s,a) + \gamma\, \mathbb{E}_{a_{t+1} \sim \pi}\big[Q^{\pi}(s_{t+1}, a_{t+1})\big]\Big]. \tag{4.8}$$

Equation 4.8 computes the expected return of choosing action a in state s under policy π. It sums the immediate reward $r(s,a)$ and the discounted future rewards, thus forming the recursive foundation for optimally updating the policy in DDPG.

DDPG's exploration in continuous spaces often involves adding noise to the actor's policy. This strategy is important for effective exploration in continuous action spaces, differentiating it from methods suitable for discrete spaces.

DDPG has been successfully applied to a variety of robotic control tasks, such as manipulation in MuJoCo simulations and multi-joint locomotion [68, 31], demonstrating its capacity to learn stable policies directly in high-dimensional, continuous settings. It has also been employed in continuous navigation for autonomous driving, where discretizing actions can lead to loss of precision [93], and in other domains requiring fine-grained control.

4.5.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) represents a significant advancement in reinforcement learning, developed to address the stability and complexity challenges inherent in policy gradient methods [109]. It is particularly effective in environments with continuous action spaces, such as robotic locomotion (e.g., MuJoCo HalfCheetah or Humanoid) and simulated control benchmarks where fine-grained actions are crucial [109].

The primary motivation for PPO, as introduced by Schulman et al. [109], is to create a more stable and efficient algorithm by implementing controlled, incremental policy updates. Traditional policy gradient methods often suffer from instability because updates can lead to drastic policy changes, especially in high-dimensional or continuous action spaces. For instance, an overly large update in such settings might cause the agent to adopt an entirely different behaviour that performs poorly or diverges entirely. PPO addresses this by using a clipped surrogate objective that constrains the policy change within a predefined

range. This ensures that learning progresses steadily without overfitting to noisy updates or losing valuable previously learned behaviours. For example, in a continuous control task like robotic locomotion (e.g., MuJoCo's HalfCheetah), such large shifts can destabilise the gait cycle learned by the agent, leading to erratic or non-functional movement. PPO's incremental updates prevent this by ensuring smoother transitions between policy iterations, which results in more consistent and sample-efficient learning.

Trust Region Policy Optimization (TRPO) [108] is one of the earlier attempts to address these stability issues by enforcing a hard constraint on the size of policy updates using a trust region method. While TRPO is effective at maintaining stability, it requires second-order gradient computations and complex optimisation, making it computationally expensive and harder to implement. Compared to TRPO, PPO's clipped objective offers a simpler yet robust alternative. By replacing the hard constraint with a clipped probability ratio, PPO achieves similar benefits in policy stability while significantly reducing computational overhead. This balance of performance and efficiency has made PPO a popular and practical choice in modern reinforcement learning applications.

PPO's architecture maintains a balance between exploration and exploitation, especially in continuous action spaces. Unlike algorithms that may aggressively update policies, leading to potential instability, PPO incrementally adjusts the policy, mitigating risks associated with significant policy deviations. This leads to more consistent convergence by avoiding large shifts in consecutive updates.

A key innovation of PPO is its clipping mechanism in the objective function, which moderates the extent of policy updates. Unlike methods such as Deep Deterministic Policy Gradient (DDPG), which often apply large, unconstrained updates to the policy parameters, PPO limits the update magnitude through a clipped surrogate objective. By constraining the policy update ratio to a narrow range determined by the hyperparameter ϵ, PPO ensures that the new policy does not deviate excessively from the old policy. This mechanism helps reduce the risk of destabilising the training process a common issue in more aggressive update strategies and makes PPO uniquely stable and reliable for continuous decision-making tasks [109].

$$L(\theta) = \min\Big(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} A^{\pi_{\theta_{old}}}(s,a),\ \text{clip}\Big(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)}, 1-\epsilon, 1+\epsilon\Big) A^{\pi_{\theta_{old}}}(s,a)\Big) \tag{4.9}$$

In practice, $L(\theta)$ is used to compute the gradient for updating the policy parameters θ. By maximizing $L(\theta)$, PPO encourages higher returns through $A^{\pi_{\theta_{old}}}$ while keeping the new policy close to $\pi_{\theta_{old}}$, thus preventing overly large policy steps.

PPO uses an actor-critic framework where the Actor proposes actions and the Critic evaluates them using the value function. The Temporal Difference (TD) error, integral to the Critic, measures the discrepancy between predicted and actual rewards [124], guiding value function updates and refining the policy through actor-critic interaction. This continuous feedback loop facilitates a more responsive and adaptive policy update mechanism, aligning with the dynamic nature of continuous environments.

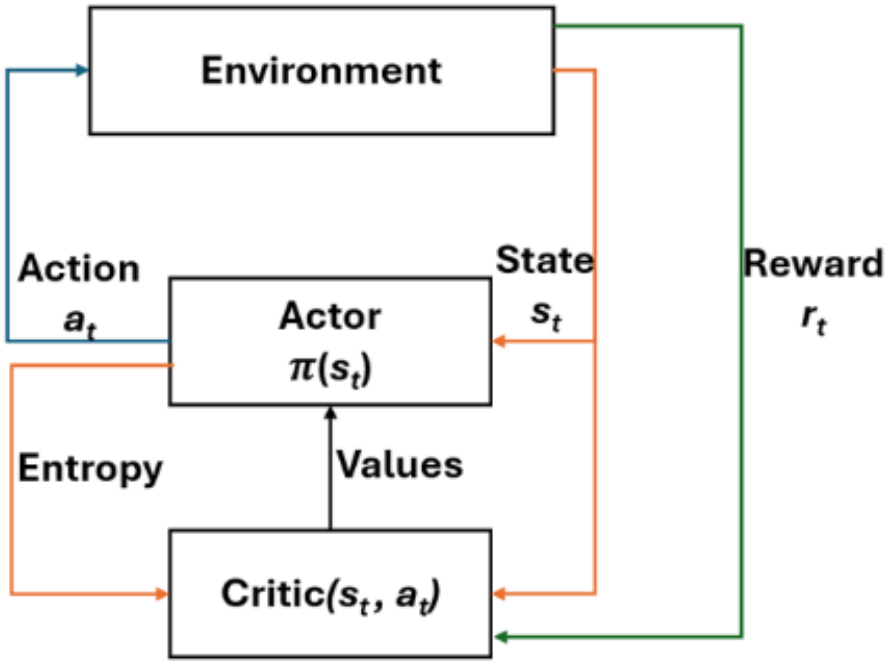


Figure 4.5: An illustration of PPO's actor-critic structure, highlighting its controlled policy update mechanism. The Critic estimates the value function (using TD error) to guide the Actor's policy updates according to the clipped objective in Equation 4.9. An entropy term is often added to the objective to encourage exploration by promoting stochasticity in the policy, which is reflected in the diagram as part of the Actor's update. (Adapted from "On Improving Cross-dataset Generalization of Deepfake Detectors".)

PPO's widespread adoption and success in various robotics, gaming, and continuous control domains demonstrate its ability to balance efficiency and stability. Its clipped objective offers a middle ground between the conservative updates of TRPO and the riskier unconstrained policy gradient methods. This robust yet relatively simple approach makes PPO a preferred choice for many complex tasks. For a deeper understanding of PPO's methodology and applications, readers are referred to Schulman et al.'s paper [109].

4.5.4 Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) is a reinforcement learning algorithm that combines the strengths of value-based and policy-based methods to optimize learning in environments with high-dimensional inputs, such as raw pixel data [84]. In traditional policy gradient methods, learning can be unstable due to high variance in gradient estimates, especially in complex or continuous environments. These instabilities can slow down learning or cause convergence failures. The primary motivation behind A2C is to create a more stable and efficient learning process by integrating the direct policy learning of the Actor network with the evaluative feedback from the critic network, which estimates the value function.

A2C uses an actor-critic framework consisting of two main components: the Actor and the Critic, as illustrated in Figure 4.6. The Actor is the policy network responsible for selecting actions based on the current state. Its parameters are updated to maximize expected returns, effectively learning the policy by moving it towards more rewarding actions [84]. Concurrently, the critic is the value network that approximates the value function, providing a baseline to evaluate the quality of actions taken by the Actor. The critic's parameters are updated to minimize the difference between predicted values and actual returns, thereby refining the policy indirectly through performance feedback [84]. Both networks often share lower-level parameters to facilitate feature learning, commonly utilizing convolutional or linear layers with separate outputs for the policy (using softmax activation [114]) and the value function (using linear activation) [84].

The advantage function $A(s_t, a_t; \theta, \theta_v)$ plays an important role in scaling the policy gradient and is computed as follows:

$$A(s_t, a_t; \theta, \theta_v) = \left(\sum_{i=0}^{k-1} \gamma^i r_{t+i} \right) + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{4.10}$$

In Equation 4.10, the discount factor γ and the horizon k are used to measure the relative advantage of actions under the current policy, allowing for more targeted and efficient updates of the policy parameters [84].

To encourage exploration and prevent premature convergence to suboptimal deterministic policies, A2C incorporates entropy regularization [29] by adding an entropy term to the objective function:

$$\nabla_\theta \log \pi(a_t|s_t; \theta)(R_t - V(s_t; \theta_v)) + \beta \nabla_\theta E(\pi(s_t; \theta)) \tag{4.11}$$

In Equation 4.11, E represents the entropy of the policy, and β is the coefficient that scales the entropy regularization term. This encourages the policy to explore a diverse set of actions, enhancing its adaptability and robustness in varied tasks and environments [84].

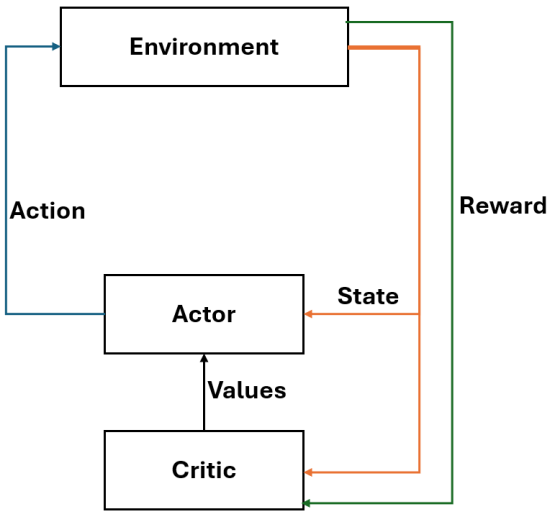


Figure 4.6: Schematic representation of the Actor-Critic architecture in the A2C algorithm, illustrating the interaction between the Actor, which selects actions, and the Critic, which evaluates the chosen actions and updates the value function based on the received reward and the computed TD error. Image adapted from "Asynchronous Methods for Deep Reinforcement Learning" by Mnih et al. [84].

A2C has proven effective in tasks with high-dimensional inputs, such as video games

and robotic control [84]. Its balanced approach, characterized by the dual Actor-Critic architecture and the strategic use of n-step returns, makes it a robust framework for complex reinforcement learning challenges. By effectively balancing exploration and exploitation, A2C facilitates informed decision-making in dynamic and challenging environments [84].

For a comprehensive understanding of A2C's methodology and applications, readers refer to Mnih et al. work [84].

4.5.5 Soft Actor-Critic (SAC)

SAC is an off-policy maximum entropy deep reinforcement learning algorithm renowned for its exceptional performance in complex, high-dimensional continuous control tasks [34]. SAC distinguishes itself by simultaneously maximising expected rewards and policy entropy. This dual objective promotes more robust learning by encouraging exploration while seeking to maximise returns [34].

Unlike on-policy methods such as Advantage Actor-Critic (A2C), which may include entropy as a secondary regulariser to aid exploration, SAC incorporates entropy maximisation as a central objective. This makes exploration an explicit part of the optimisation process rather than a mere auxiliary loss term. Furthermore, SAC is off-policy and employs stochastic policies, allowing for more sample-efficient learning and reuse of experience, in contrast to A2C's on-policy nature, which requires fresh data for every update. SAC integrates this maximum entropy framework directly into its policy objectives. Many existing algorithms, such as Deterministic Policy Gradient methods or on-policy techniques like Proximal Policy Optimisation (PPO), primarily focus on reward maximisation and either implicitly encourage exploration through hyperparameter tuning or do not address it explicitly. In contrast, SAC explicitly balances the trade-off between reward maximisation and policy entropy, thereby mitigating the risk of premature convergence to suboptimal deterministic policies. Additionally, SAC employs twin Q-networks to reduce overestimation bias in the learned action-value function, a feature that sets it apart from some earlier actor-critic approaches [126].

SAC uses an actor-critic architecture comprising two primary components: the Actor and the Critic, as illustrated in Figure 4.7. The Actor network proposes a policy $\pi(a_t|s_t;\theta)$,

which determines the actions to be taken based on the current state. Concurrently, the Critic network assesses the quality of these actions through a Q-function $Q(s_t, a_t; \theta_q)$. The Critic's evaluation is important in refining the Actor's policy towards more valuable actions, thereby improving the overall learning process.

The soft Q-value in SAC is iteratively updated using a Bellman operation that incorporates entropy to balance exploration and exploitation:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p, a_{t+1} \sim \pi} \big[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1}|s_{t+1}) \big] \tag{4.12}$$

In Equation 4.12, $r(s_t, a_t)$ denotes the reward received after executing action a_t in state s_t, and γ is the discount factor. The expectation is taken over the next state s_{t+1} and action a_{t+1} sampled from the current policy π. The term $-\log \pi(a_{t+1}|s_{t+1})$ represents the entropy gain, encouraging the policy to maintain a degree of randomness and thus promoting exploration.

Policy improvement in SAC is achieved by adjusting the policy parameters to minimize the divergence from a Boltzmann distribution of the Q-function:

$$\pi_{\text{new}}(a_t|s_t) \propto \exp\left(\frac{Q(s_t, a_t)}{\alpha} \right) \tag{4.13}$$

In Equation 4.13, the temperature parameter α controls the trade-off between reward and entropy. A higher α places greater emphasis on entropy, leading to more exploratory policies, while a lower α prioritizes reward maximization, resulting in more deterministic policies.

To operationalize the maximum entropy framework, SAC introduces an entropy-regularized objective function:

$$J(\pi) = \mathbb{E}_{s_t \sim p, a_t \sim \pi} \big[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \big] \tag{4.14}$$

This objective function in equation 4.13 ensures that the policy not only seeks to increase

the expected return but also to act as randomly as possible within the bounds set by α. This balance enhances the policy's adaptability and robustness across varied tasks and environments [34].

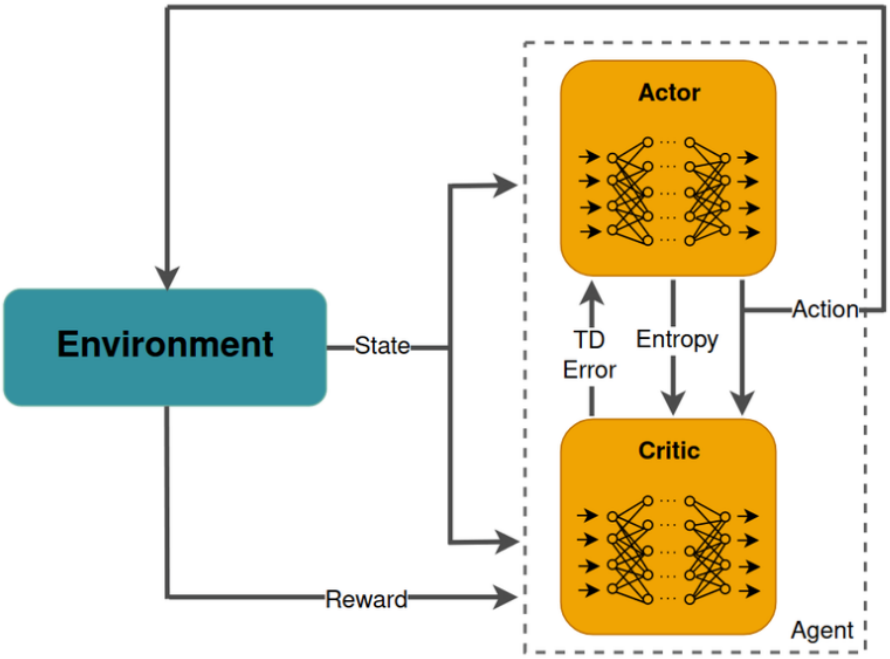


Figure 4.7: The Soft Actor-Critic (SAC) architecture, illustrating the interaction between the environment, the Actor, and the Critic within the agent. The environment provides the state to the Actor, which then decides on an action to take. This action leads to a new state and a reward from the environment. The Critic assesses the taken action by calculating the Temporal Difference (TD) Error and accounting for the entropy of the policy, which encourages exploration. The Actor uses the Critic's assessment to update its policy, aiming to maximize both the expected return and the entropy of the policy, thus fostering both effective exploitation and exploration. Image adapted from "A Deep Reinforcement Learning-Based Resource Scheduler for Massive MIMO Networks" [1].

Over recent years, SAC has demonstrated state-of-the-art performance across a variety of continuous control domains, including robotic manipulation [32], humanoid locomotion [34], and complex simulated tasks in MuJoCo and OpenAI Gym environments [126]. Its ability to balance exploration and exploitation through entropy maximization makes SAC particularly effective in high-dimensional and highly stochastic tasks, where thorough exploration is essential for avoiding local optima and discovering optimal policies.

The SAC algorithm's architecture and entropy maximization principle have significantly advanced the capabilities of deep reinforcement learning, especially in environments requiring nuanced and adaptive decision-making. For a comprehensive understanding of SAC's

mechanics and benefits, refer to the foundational work by Haarnoja et al. [34].

4.5.6 DreamerV3

DreamerV3 is a model-based reinforcement learning (RL) algorithm that significantly advances learning efficiency and adaptability across diverse environments by leveraging a learned world model for predictive planning [40]. By anticipating future states, DreamerV3 can more effectively learn from its latent representations, achieving state-of-the-art performance in both continuous and discrete action tasks.

DreamerV3 is an evolution of the earlier Dreamer and DreamerV2 algorithms [36, 38]. While previous versions already demonstrated impressive sample efficiency by learning a latent world model, DreamerV3 refines this approach through improved stability mechanisms, more robust optimization strategies, and the ability to handle a wider range of input scales. These improvements make DreamerV3 more versatile across tasks like high-dimensional robotics, Atari games, and simulated control environments [40].

DreamerV3 uses a Recurrent State-Space Model (RSSM) that captures both the deterministic and stochastic aspects of environment dynamics [40]. The RSSM projects the agent's observations into a compact latent space, enabling the agent to predict future states from imagined rollouts without directly interacting with the environment. The RSSM equation is as follows.

$$h_{t+1} = f_{\text{det}}(h_t, z_t, a_t; \theta_h), \tag{4.15}$$

$$z_{t+1} \sim p_{\text{stoch}}(z_{t+1} \mid h_{t+1}; \theta_z), \tag{4.16}$$

where h_t is the deterministic hidden state, z_t is the stochastic latent variable, and a_t is the action at time t. Function f_{det} updates the hidden state deterministically, while p_{stoch} models uncertainty through latent variables, capturing non-deterministic environment factors. By learning to predict future observations (or rewards) from this latent model, DreamerV3 can plan and optimize its policy through "imagined" trajectories.

Within the latent space defined by the RSSM, DreamerV3 uses a critic network to estimate the expected return of taking action a_t in a latent state s_t. This latent-state Q-function $Q(s_t, a_t)$ guides the agent toward beneficial long-term rewards. The critic parameters are updated using the Bellman equation:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}\big[V(s_{t+1})\big], \tag{4.17}$$

where r is the reward, γ is the discount factor, and $V(s_{t+1})$ is the value function approximated within the latent space. By training on imagined transitions sampled from the RSSM, DreamerV3 drastically reduces interactions with the real environment, improving efficiency.

The actor network is responsible for selecting actions that maximize cumulative return while balancing exploration via entropy. DreamerV3 extends standard policy gradients by integrating an entropy term:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_t \gamma^t \Big(Q(s_t, \pi(s_t)) + \alpha \, \mathcal{H}\big(\pi(\cdot \mid s_t)\big)\Big)\right], \tag{4.18}$$

where $\mathcal{H}(\pi(\cdot \mid s_t))$ denotes the policy entropy, and α is a temperature parameter that controls the trade-off between exploration and exploitation [34]. DreamerV3 learns the RSSM, actor, and critic simultaneously using stochastic gradient descent (SGD). A replay buffer may also be used to store and reuse past experience, further improving data efficiency.

Two notable innovations in DreamerV3 are symlog prediction and divided KL divergence loss [40]. Symlog prediction is a scaling approach designed to handle unbounded or extremely large values, helping to stabilise learning across a wide range of observation magnitudes. By applying a symmetric logarithmic transformation to predicted targets (e.g., rewards or latent representations), it mitigates numerical instability and promotes smoother gradient updates. While Symlog is not inherently limited to DreamerV3, it is particularly beneficial in models like Dreamer that operate over raw pixel observations or use latent-variable predictions, where unbounded values and large fluctuations are more

likely. Other architectures could theoretically incorporate Symlog, but its impact is most noticeable in environments with high dynamic range or stochastic latent transitions. As such, its widespread adoption depends on the specific needs and design assumptions of each model. Divided KL divergence loss addresses the challenge of reconciling multiple data distributions—for instance, reconstructions of past observations versus predictions of future states in the RSSM. Splitting the KL term into separate divergences for short-horizon and long-horizon predictions allows the model to balance immediate accuracy with robust long-term planning.

DreamerV3 uses several stability mechanisms to ensure consistent learning in both the latent model and the policy optimization. It maintains periodic copies of the critic and world model parameters, known as target networks, which smooth out training targets and mitigate instability from rapidly changing Q-values. Return normalization is used to rescale returns, handling varying reward magnitudes and preventing large updates that could destabilize training. Furthermore, imagination rollouts simulate multistep trajectories in latent space, allowing DreamerV3 to update both the critic and actor with 'imagined' experience, thus lowering sample complexity compared to purely model-free RL.

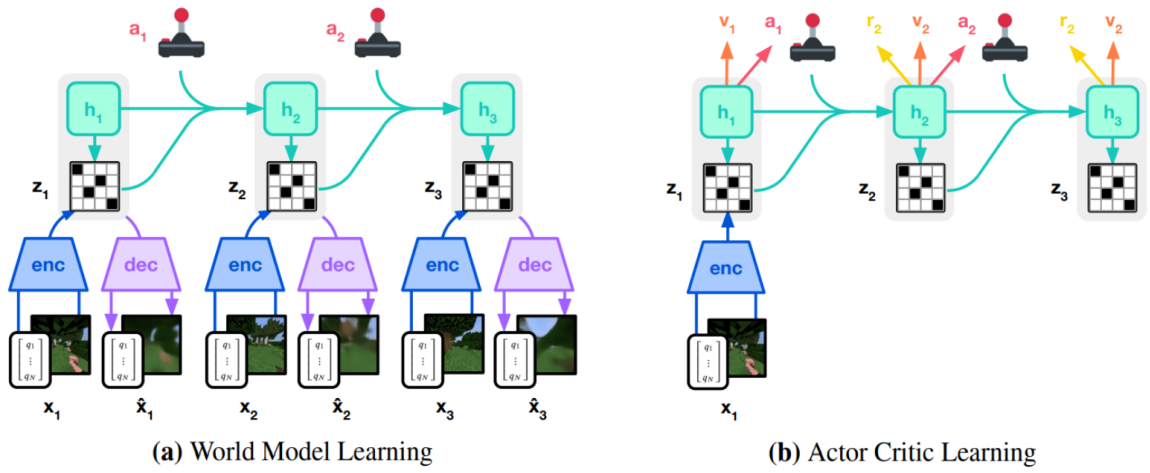


Figure 4.8: Overview of DreamerV3's pipeline, adapted from [40]. The RSSM learns a compact latent representation of observations, predicting both deterministic and stochastic elements of the environment. The Critic network evaluates the agent's decisions within this latent space, while the Actor network selects actions that maximize expected return and policy entropy. The entire system is updated via stochastic gradient descent, leveraging imagined rollouts and stable training techniques such as target networks and return normalization.

DreamerV3 has demonstrated notable success in various continuous and discrete action

tasks, including Atari, MuJoCo, and the DeepMind Control Suite, showing strong sample efficiency and state-of-the-art performance compared to other model-free or model-based algorithms [40]. The ability to handle visual and low-dimensional inputs alike, along with symlog scaling and KL balancing, makes DreamerV3 versatile. Its approach to learning from imagination in latent space, combined with robust optimization methods, places DreamerV3 at the forefront of model-based RL research.

In summary, DreamerV3 refines and extends the capabilities of previous Dreamer algorithms by integrating symlog prediction, divided KL divergence loss, and RSSM for learning complex dynamics. These methods, coupled with stable training mechanisms and a focus on long-horizon predictive planning, enable DreamerV3 to master a wide spectrum of challenging RL tasks efficiently. For a deeper exploration of DreamerV3's design and empirical results, readers are referred to Hafner et al. [40].

4.6 Comparative Analysis of RL Algorithms Used in this Thesis

Reinforcement Learning (RL) encompasses a wide variety of algorithms broadly categorized into model-free and model-based methods, each offering distinct approaches to policy optimization, exploration, and computational overhead. Within model-free RL, algorithms can be further divided into on-policy methods (e.g., Proximal Policy Optimization, Advantage Actor-Critic) and off-policy methods (e.g., Deep Q-Network, Soft Actor-Critic, Deep Deterministic Policy Gradient). In contrast, model-based RL, like DreamerV3, leverages an internal model of the environment to facilitate predictive planning and reduce real-world interactions.

Table 4.1 presents a comparison of these approaches, focusing on two key criteria: **Model Type** and **Policy Type**.

Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) are both model-free, on-policy algorithms renowned for their stable and reliable policy updates. PPO achieves stability by constraining policy changes within each update step, thereby minimizing the risk of erratic behaviors during training. This characteristic is particularly

Algorithm	Model Type	Policy Type
Proximal Policy Optimization (PPO)	Model-Free	On-Policy
Advantage Actor-Critic (A2C)	Model-Free	On-Policy
Deep Q-Network (DQN)	Model-Free	Off-Policy
Soft Actor-Critic (SAC)	Model-Free	Off-Policy
Deep Deterministic Policy Gradient (DDPG)	Model-Free	Off-Policy
DreamerV3	Model-Based	Off-Policy

Table 4.1: Comparison of RL Algorithms Used in This Thesis

advantageous in environments requiring precise and cautious movements, such as navigating through crowded spaces. However, their on-policy nature necessitates more interactions with the environment, potentially slowing the learning process in dynamic settings.

Advantage Actor-Critic (A2C) enhances traditional actor-critic methods by maintaining separate actor and critic networks. The actor directly updates the policy, while the critic evaluates actions by estimating the value function. This structure allows A2C to learn efficiently in environments with high-dimensional inputs. Despite these strengths, A2C shares the limitation of PPO regarding the need for extensive environment interactions in rapidly changing scenarios.

On the other hand, model-free, off-policy algorithms like Deep Q-Network (DQN), Soft Actor-Critic (SAC), and Deep Deterministic Policy Gradient (DDPG) offer distinct advantages. Deep Q-Network (DQN) uses experience replay and target networks to stabilize training by decoupling data collection from the learning process. This approach enhances learning efficiency, especially in environments with discrete action spaces, such as simple navigation tasks with clearly defined actions like "move forward" or "turn left." However, DQN's performance may decline in environments requiring high-dimensional or continuous actions.

Soft Actor-Critic (SAC) extends the actor-critic framework by integrating an entropy maximization objective, which promotes exploration by maintaining policy randomness. SAC is well-suited for continuous action spaces, enabling robots to perform nuanced movements necessary for smooth navigation among humans. The entropy term helps prevent premature policy convergence, fostering robust learning in stochastic environments. Nonetheless, SAC requires careful hyperparameter tuning to effectively balance exploration and

exploitation.

Deep Deterministic Policy Gradient (DDPG) is designed for continuous control tasks, combining the actor-critic architecture with experience replay. The actor network generates deterministic actions, while the critic evaluates them to guide policy updates. DDPG excels in tasks demanding precise control, such as manipulating robotic arms or steering autonomous vehicles. However, its sensitivity to hyperparameter settings can pose challenges, necessitating extensive tuning to achieve optimal performance across diverse environments.

DreamerV3, representing the model-based method, learns an internal model of the environment to predict future states and rewards. By simulating interactions within this learned model, DreamerV3 can plan actions that optimize long-term rewards without relying heavily on real-world trial and error. This approach significantly improves sample efficiency, making it ideal for environments where data collection is expensive or time consuming. In social robot navigation, DreamerV3's ability to anticipate human movements and plan accordingly reduces collision risks and improves safety. However, the effectiveness of model-based methods like DreamerV3 depends on the accuracy of the learned model, which can be challenging to maintain in highly dynamic or unpredictable environments.

On-policy model-free algorithms like PPO and A2C are preferable in high-risk settings requiring stable and safe behavior, despite their slower adaptation rates. Off-policy model-free algorithms such as DQN, SAC, and DDPG offer greater sample efficiency and faster policy improvements, making them suitable for environments where robust exploration is feasible and safe with appropriate constraints. Model-based algorithms like DreamerV3 excel in complex environments that demand long-term planning and high sample efficiency, provided that sufficient computational resources are available to sustain an accurate internal model.

This chapter explored the fundamental ideas of Reinforcement Learning, explaining both model-free and model-based paradigms as well as the distinction between on-policy and off-policy approaches within the model-free family. Each framework addresses learning from trial and error in different ways. On-policy methods like PPO and A2C prioritize stability and moderate exploration, making them a safe choice for complex, real-world

scenarios at the cost of slower adaptation. Off-policy methods, such as DQN, SAC, and DDPG, leverage replay buffers to rapidly improve policies but can risk overshooting safe exploration boundaries without careful reward shaping or constraints. Meanwhile, model-based RL, illustrated by DreamerV3, offers powerful lookahead capabilities that reduce real-world interactions, though its reliance on accurate modeling adds computational cost and complexity.

No single RL method surpasses all others across every dimension. Instead, each offers trade-offs depending on the complexity of the environment, the safety requirements, and the resource constraints. Hybrid solutions such as DreamerV3, combining on-policy resilience with off-policy efficiency or integrating model-based planning for long-horizon decision-making, increasingly capture the strengths of multiple paradigms.

Chapter 5

Predictive World Models for Social Navigation

This chapter focuses on a key aspect of our research: the development and evaluation of our novel predictive world models in social navigation. Recognising the need for efficient and safe navigation for social navigation robots, we investigate how to improve the effectiveness of reinforcement learning (RL) in this area.

Unlike the general overview of RL and its application in social navigation discussed in chapter 4, this chapter explains our three novel methods of predictive world models. These methods are compared against current state-of-the-art algorithms and thoroughly assessed using a range of metrics throughout their training and testing stages. The contribution of this chapter is the detailed development and practical application of these novel models within the specific context of social navigation.

Here we focus on the application of reinforcement learning in social navigation—an area of increasing importance due to the growing interactions between humans and robots. Moving beyond the general discussions and theoretical foundations previously explored, this chapter provides a detailed examination of our novel methods integrated with RL algorithms tailored for social navigation tasks.

Our research presents a world model inspired by Ha and Schmidhuber [33], combining a

Variational Autoencoder (VAE) [60] and a Long-Short Term Memory network (LSTM) [49]. The key contribution in this research includes three original methods: *2StepAhead*, *MASPM*, and *2StepAhead-MASPM*. Each method is designed to improve prediction capabilities and decision-making efficiency in dynamic social environments.

5.1 Related Work

In Chapter 4, we defined Reinforcement Learning (RL) as a learning paradigm where an agent learns by interacting with its environment to maximize a given reward, using Markov Decision Process framework [43, 123]. In the domain of robotics, RL has been leveraged to teach robots complex manipulation tasks [2], and in gaming, it has been used to develop agents that can play games proficiently [76].

Despite its wide-ranging successes, RL has well-known limitations, as discussed in Chapter 4. Adaptability to new environments poses a significant challenge [104], they often require large volumes of data for training, making it computationally intensive [73]. Specifically in the domain of social navigation, these issues become even more pronounced due to the rich and complex nature of social dynamics [146, 115, 98]. The complex interactions that happen in social settings are difficult to model and predict, making RL agents' learning of optimal policies even more challenging [16].

In response to these challenges, world models, as discussed in Chapter 4, have used to tackle some of the challenges. For instance, MuZero, an RL-based method using world models, has demonstrated its efficiency in learning ATARI game rules using observed image data and action sequences, even with limited computational resources [106, 37]. AlphaGo, another RL-based method using world models, outperformed human experts in the game of Go in 2016 [116].

Unlike traditional RL-based predictions that rely on the current environment state represented by a state-action pair, predictive world models can consider both past and present states to anticipate future states [33]. This methodology has been applied successfully in environments such as CarRacing [11] and Doom [59], outperforming traditional RL [33]. Furthermore, world models introduce a predictive component to the RL dynamics, enabling

the agent to anticipate future actions. This can lead to faster learning and potentially improved results in fewer episodes [139].

Additionally, world models augment the MDP framework by shifting from depending solely on current observations and actions to considering past current and future states and actions. This allows agents to generate more informed policies based on a predictive understanding of the environment [19]. Notably, Dreamerv3, a model-based RL agent, has demonstrated the capability of combining world models and policy learning to achieve state-of-the-art performance in various tasks [35, 40].

This chapter delves deeper into these predictive world models, specifically within RL-based social navigation which leads to our research question: "Can world models help us improve RL-based social navigation?"

5.2 Methodology

In this chapter we explore the use of predictive world models to improve RL-based SocNav using the three proposed methods –2StepAhead, MASPM, and 2StepAhead-MASPM; in this section, we describe the three approaches and provide experimental details.

Our experiments are conducted in SocNavGym [58], a configurable environment specifically designed for social navigation scenarios. See section 6.4.8 and chapter 2 for more information about the environment. This environment has the capacity to integrate a wide range of entities such as humans (static or moving), plants, tables, and laptop computers. For our experiments, SocNavEnv was configured to work with a discrete action space of four actions (stop, move forward, rotate left, and rotate right), three moving humans, and a social navigation reward function [4]. The goal of the agent in SocNavGym is to train the agent to navigate towards the target while avoiding collisions with surrounding entities and minimising the discomfort caused to the humans. A screenshot of SocNavEnv is shown in Fig. 5.1.

Although we are aware that in real-life settings the number of individuals involved is frequently greater than three, we found that including three humans was sufficient for the experiments to be challenging for the RL algorithm used as a baseline. This was determined

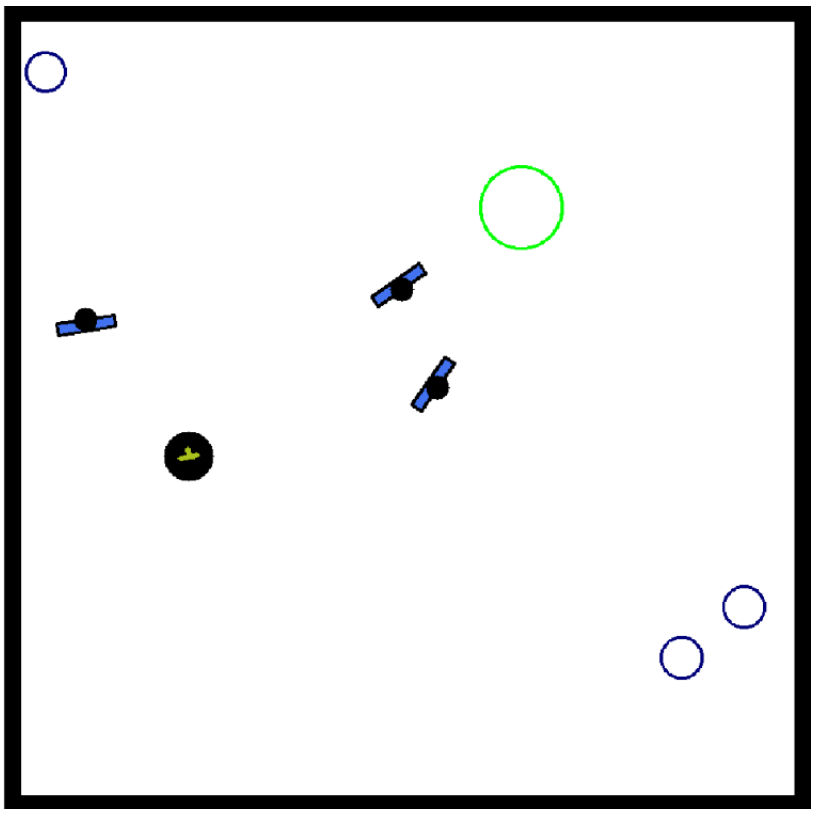


Figure 5.1: Screenshot of SocNavEnv, the environment used for the experiments [58]. Blue rectangle represent humans, blue circles indicate humans' goals (which are non-observable to the robot), green circles represent the robot's goals, and black-green circles represent robot agents.

empirically during preliminary trials: as the number of humans increased from two to three, the agent's success rate in goal-reaching tasks dropped significantly, and learning curves exhibited higher variance and slower convergence. These observations indicated that even with three humans, the spatial complexity and collision risk introduced a non-trivial challenge for baseline agents such as SAC. This environment has been discussed earlier in Chapter 2, Social Robot Navigation.

5.2.1 Two step Ahead Predictive World Model: 2StepAhead

2StepAhead extends the vanilla approach of Ha and Schmidhuber [33] by predicting the hidden state and the latent state two steps ahead. The number of steps that the model is predicting ahead was empirically determined out of 2, 4, 8, and 16 steps. Although this number arguably depends on the environment, predicting more than 2 steps ahead did not improve the results in our SocNavGym setup and made training slower.

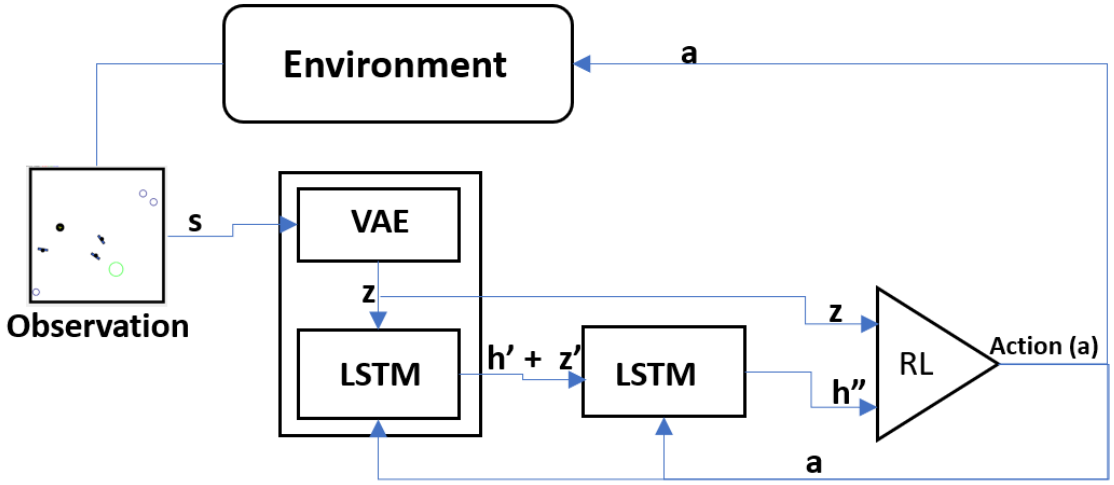


Figure 5.2: In 2StepAhead, the same LSTM is used recursively to predict two steps ahead.

As depicted in Fig. 5.2, our model predicts two steps ahead for the hidden state (h'') and the latent state (z'') by using the predicted next and hidden states (z', h') and the current action (a):

$$(z'', h'') = \text{LSTM}(z', h', a; \psi).$$

Subsequently, the environment's current latent state (z) and the two steps ahead hidden

state (h'') are fed into the Dueling DQN to choose the next action (a^*):

$$a^* = \text{Dueling DQN}(z|h''; \xi),$$

where ξ represents the parameters of our Dueling DQN. By predicting the latent state of the environment two steps ahead, we hope to provide to the RL algorithm richer information regarding the future state in case the robot keeps taking the current action, potentially improving performance and robustness in a dynamic environment.

5.2.2 Multi Action State Predictive Model: MASPM

This model provides the Dueling DQN with a comprehensive view of future state possibilities, encompassing all four available actions, potentially enabling more informed decision-making and thereby improving the model robustness and performance (see Fig. 5.3).

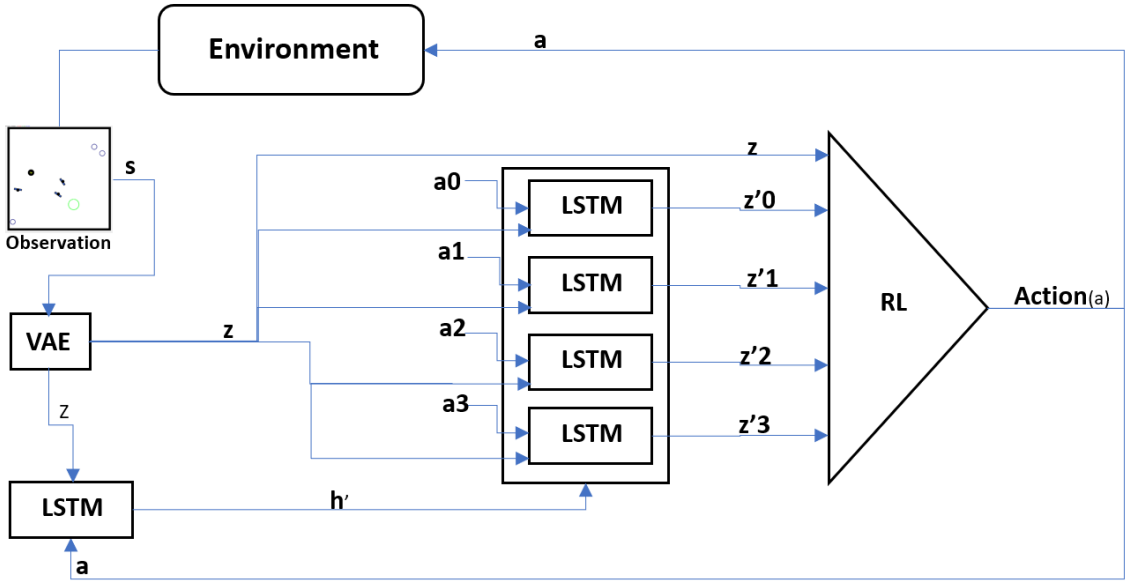


Figure 5.3: In MASPM, the LSTM is not used recursively, but it is provided with the four possible actions and all the resulting data are fed into the RL algorithm.

The latent state (z) along with the action serve as inputs for an LSTM, which predicts the next state and hidden state based on the given action. For each action i, the latent state and action are input to the LSTM model to predict the next state and hidden state:

$$(z_i', h_i') = \text{LSTM}(z|a_i; \psi),$$

where z is the current latent state, a_i is the i-th action (provided to the network as a one-hot encoding), and ψ represents the LSTM parameters. The four next predicted states, z'_0, z'_1, z'_2, z'_3, together with the current latent state z then serve as inputs for the Dueling DQN to estimate the best action a^*:

$$a^* = \text{Dueling DQN}(z|z'_0|z'_1|z'_2|z'_3; \xi),$$

where ξ represents the Dueling DQN parameters. MASPM provides a broadened perspective of future states across multiple actions, offering the Dueling DQN a richer foundation for decision-making.

5.2.3 Combining 2StepAhead and MASPM: 2StepAhead-MASPM

The 2StepAhead-MASPM is a combination of MASPM and the 2StepAhead method and aims to combine their advantages. This model provides a two-step-ahead prediction for each potential action. The two-step-ahead prediction horizon facilitates the Dueling DQN algorithm with a more refined decision-making capability. It achieves this by leveraging the current latent state and the predicted two-step-ahead state for each possible action to determine its subsequent action.

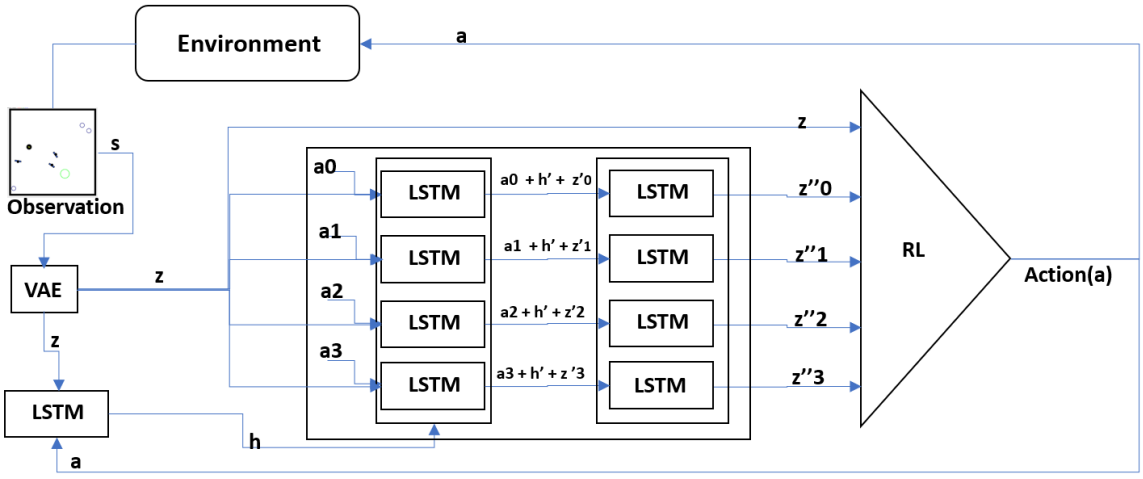


Figure 5.4: 2StepAhead-MASPM combines the advantages of 2StepAhead and MASPM. It predicts two steps ahead and considers all actions instead of just the current action.

Figure 5.4 illustrates the architecture of the proposed 2StepAhead-MASPM. The latent state (z), coupled with the related action, is fed into the LSTM. The LSTM uses these

inputs to predict the next state and the hidden state base on the input action. For each action a i, the LSTM model processes the latent state and action as input and predicts the corresponding next state and hidden state. The model repeats this process, using the same action and the previously predicted latent state for the second prediction.

Given a latent state z and an action a_i at a time t, the LSTM predicts the next state z_{t+1} and hidden state h_{t+1}. The process is repeated using the new latent state z_{t+1} and the same action a_i to predict the next latent state z_{t+2} and hidden state h_{t+2}:

$$(z_{t+1}, h_{t+1}) = LSTM(z_t, a_i, h_t; \phi)$$
$$(z_{t+2}, h_{t+2}) = LSTM(z_{t+1}, a_i, h_{t+1}; \phi)$$

We hypothesise that combining the two-step ahead predictions with a coverage of all actions can improve Dueling DQN's decision-making. In the next section, we benchmark three proposed methods against the selected baselines to evaluate whether the use of Predictive World Models is beneficial in the context of SocNav.

5.3 Experimental results

All the developed models are based on the Dueling DQN reinforcement learning algorithm and are trained within the SocNavEnv environment [58]. To ascertain the influence of predictive world models on RL-based social navigation, Dueling DQN is also used as our initial baseline. The hyperparameters of Dueling DQN, particularly the size of the hidden layers, are critical in determining the agent's learning capabilities [119]. Therefore, we evaluated two Dueling DQN MLP model architectures: one with two hidden layers of size 128 each, and another with layers of size 512 and 128, respectively. After 200,000 episodes—the number of episodes required for all experiments to converge in this experiment—the model with hidden layers of size 512 and 128 achieved a slightly higher expected cumulative reward (approximately 0.65 compared to 0.62 for the 128-128 configuration).

This improvement of approximately 0.03, representing a 4.8% increase over the simpler architecture, demonstrated a consistent advantage for the 512-128 configuration. However, the 512-128 configuration required approximately 10% more computational time per

episode. The decision to proceed with this architecture was based on empirical trade-off analysis: multiple runs showed that the performance gains were consistent and statistically significant, while the additional computation remained within acceptable limits for our training infrastructure. Thus, the marginal cost in runtime was outweighed by the robustness and quality of the learned policy, justifying its use in subsequent experiments.

Therefore, we selected this architecture for the rest of the Dueling DQN-based agents. Subsequently, we integrated predictive world models into the RL framework according to the three methods proposed in Section 5.2. We evaluated the proposed methods using different metrics [25], each uniquely designed with predictive capabilities, in the context of social navigation tasks.

The novelty of our approach lies in the integration and evaluation of predictive world models specifically, 2StepAhead, MASP, and 2StepAhead-MASP– within the context of social navigation, which has not been explored previously, as well as in the models themselves. For a comprehensive and reliable evaluation, we used multiple metrics during the training and testing phases.

Using only a single metric can limit the scope of the evaluation and may not fully capture the model's performance due to the multifaceted nature of social navigation tasks. Metrics such as discomfort counts, human collisions, and personal space compliance are as important as the traditionally employed metrics in RL such as reward or convergence time [41, 15, 146]. Therefore, we use this broad range of metrics to ensure a holistic analysis that comprehensively reflects the performance in a human-robot interactive environment. Furthermore, our comparative analysis extends beyond our baseline Dueling DQN models. For the testing phase, we also include comparisons with other established models in the domain, like the RVO2 and social force model, to provide a broader context for the performance of our models. These benchmarks were chosen due to their widespread use in social navigation tasks [132, 46].

5.3.1 Training Phase Metric Evaluation

The training phase focuses on the cumulative reward, training time, and episodes to convergence. The results from this phase, as shown in Fig. 5.5, demonstrate improvements in

our proposed models over the baseline Dueling DQN.

The *Vanilla Dueling DQN* shows steady learning but converges at a lower cumulative reward of approximately 0.52. The *DuelingDQN + Hidden State* model reaches a cumulative reward of around 0.61, while the *DuelingDQN + Multi Action State Predictive Model (MASPM)* achieves a slightly higher reward of approximately 0.64.

Among the proposed methods, the *2StepAhead* model demonstrates early efficiency, solving the task in about 3200 episodes and stabilizing at a cumulative reward close to 0.65. The *2StepAhead-MASPM* model stands out as the best performer, achieving the highest cumulative reward of 0.67 and maintaining consistent performance throughout training.

These results indicate that integrating predictive world models particularly the *2StepAhead* and *2StepAhead-MASPM* enhances decision-making efficiency and learning performance compared to the baseline approach.

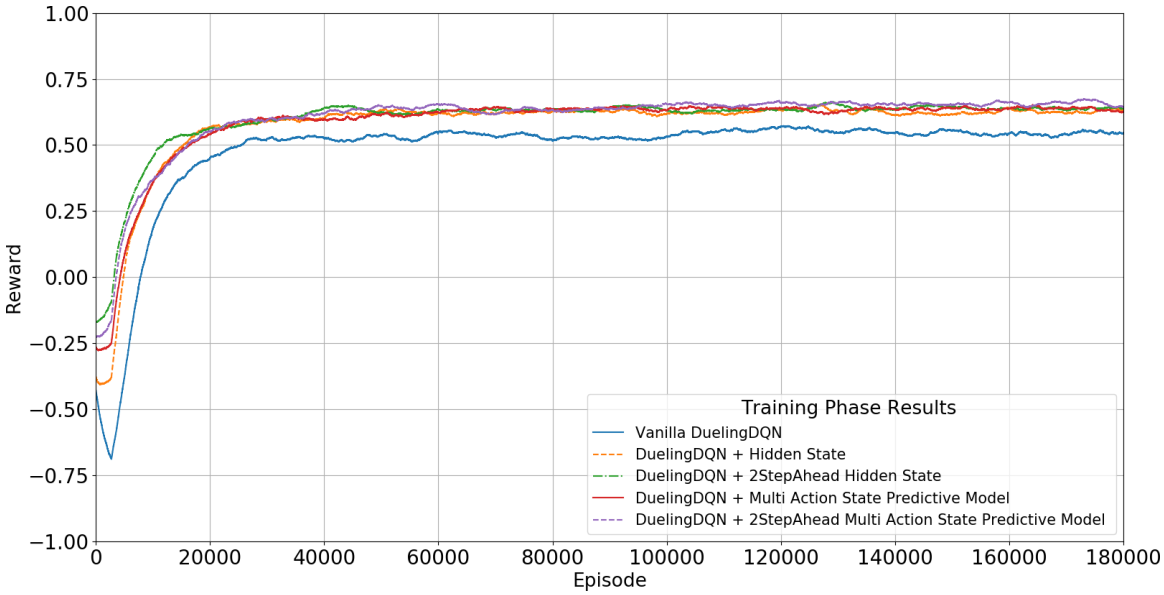


Figure 5.5: Smoothed cumulative reward during training.

5.3.2 Testing Phase Metric Evaluation

In the testing phase, we used a broad range of metrics related to human-robot interactions, navigation efficiency, and overall performance, and measured those metrics for 500 episodes per algorithm. In Figures.5.6 and 5.7, the histograms of the following metrics are shown:

- Human discomfort: Average human discomfort caused to humans, as described in [4]. *Quantifies interference with human comfort zones. Reducing discomfort is essential for human acceptance of robots in shared spaces.*

- Distance travelled: Distance travelled by the agent, per episode (in meters). *Indicates navigation efficiency. A shorter distance suggests optimal path planning.*

- Simulation time: Calculated as the number of steps multiplied by the step time (in seconds). *Reflects computational efficiency, which is crucial for real-time applications.*

- Human collisions: Whether the robot collides with a human or not (binary metric). *Measures safety; avoiding collisions is fundamental in social navigation tasks.*

- Max steps: Whether the agent reaches the maximum number of steps in a particular episode (binary metric).

- Reward: The cumulative reward per episode (scalar).

- Successful run: Whether the agent reaches the goal or not in an episode (binary metric).

- Idle time: Steps where the robot moves less than 0.05m (in seconds). *Steps where the agent moves minimally can indicate inefficiency or hesitancy in decision-making.*

- Personal space compliance rate: Ratio of the time where the robot is farther away than 0.5 metres from any human divided by the total time (scalar). The threshold distance is based on proxemics theory, which categorizes personal space into distinct zones [41, 24, 118]. *Evaluates the agent's ability to respect human boundaries, aligning with proxemics theory for human-robot interaction.*

The 2StepAhead-MASPM achieved higher average cumulative reward than the baseline models. Success rate, human collision, and cumulative reward were also improved with our 2StepAhead-MASPM model. Our model performed well overall, achieving the second-best in minimal idleness and ranking third for personal space compliance, simulation time, and distance travelled, respectively. However, it is important to remember that optimising one aspect of social norms may have unintended consequences on others. For example, while

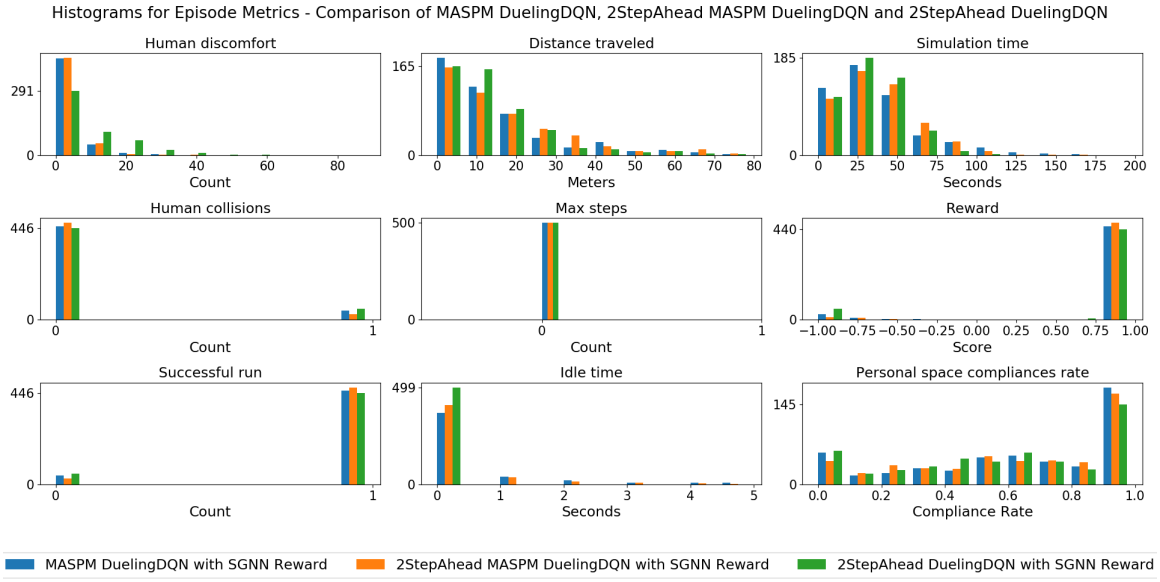


Figure 5.6: Histograms of the metrics used for comparison, applied to the three proposed models.

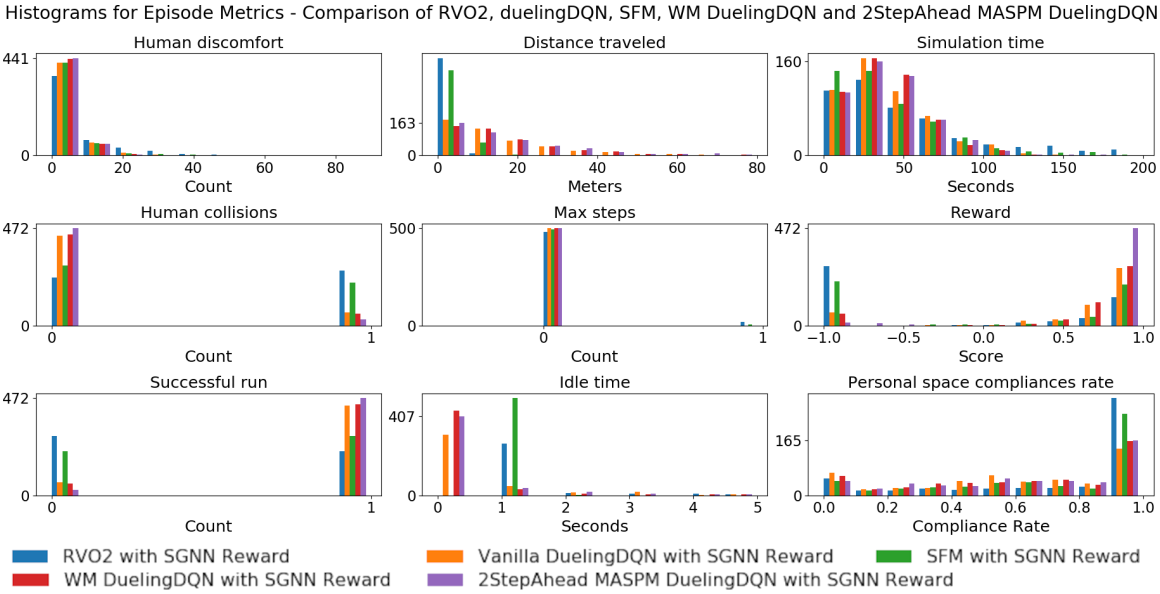


Figure 5.7: Histograms of the metrics used for comparison, applied to RVO2, Dueling DQN, SFM, WM Dueling DQN, and 2StepAhead-MASPM Dueling DQN.

reducing the time to reach the goal by finding the shortest path may be desirable, this could compromise human personal space. Therefore, the ideal solution is not to maximise one specific metric but to strike a balance across all metrics.

While our 2StepAhead-MASPM model might not have achieved the highest score in all individual metrics, it excelled in achieving well-rounded results over most metrics used, respecting the Pareto nondomination criterion [147], i.e., no other method performed better across all metrics. It improved critical aspects of social norms such as avoiding collisions with humans and maintaining a high success rate, all without excessively compromising personal space compliance. Moving forward, our aim is to continue refining our models to obtain an even better balance across the multiple dimensions involved, thereby further improving performance in complex, multi-faceted tasks such as social navigation.

Conclusions and Future Work

The experimental results confirm the value of integrating world models in RL-based social navigation. We present a novel contribution –the 2StepAhead-MASPM predictive model integrated into the Dueling DQN framework– which demonstrated superior performance over the baseline models across various metrics, particularly in terms of success rate, cumulative reward and human collision. However, our study also revealed areas where improvement can be made, most notably in terms of maintaining personal space –an essential aspect in social navigation. This insight highlights the importance of the Pareto non-domination criterion [147] in dealing with such multi-faceted tasks.

As future work, we are planning to experiment on more complex navigation environments and continuous action spaces. By introducing a range of different obstacles such as tables, chairs, and laptops, and varying the number of humans present in the environment, we aim to simulate more realistic and dynamic scenarios. With these, we want to further test the limits of predictive world models and refine the performance of our models. Ultimately, our goal is to develop an RL agent that not only navigates efficiently through complex social environments but also maintains respect for personal boundaries and pedestrians' comfort.

Chapter 6

Cosine-Gated LSTM

This chapter introduces the Cosine-Gated Long Short-Term Memory (CGLSTM) model a novel approach designed to improve sequence prediction by integrating a cosine similarity-based gate into the vanilla Long Short-Term Memory (LSTM) architecture. Building upon the insights gained in Chapter 5, where predictive world models addressed challenges in discrete action spaces, we observed that transitioning from a discrete action space to a continuous action space in the SocNavGym environment [56] led to more pronounced prediction errors. These errors became particularly significant when the number of action choices increased from 4 to 16, highlighting the limitations of standard LSTM models in managing complex temporal dependencies and higher-dimensional action spaces.

To address these limitations, we develop the CGLSTM model, which we evaluated across various datasets and tasks, including the FallingBallEnv [87], the adding problem [3], MNIST and Fashion-MNIST classification [66, 145], IMDB sentiment analysis [72], and language modeling on the Penn Treebank corpus [75]. The performance of CGLSTM was benchmarked against established models such as LSTM, Gated Recurrent Unit (GRU), Recurrent Attention Unit (RAU), and Transformer models. Additionally, we demonstrated the CGLSTM's effectiveness in predicting observations within the SocNavGym environment, showcasing its potential for practical applications in dynamic, real-world scenarios. Our results indicate that the CGLSTM model achieves significant improvements in both predictive accuracy and efficiency, making it a promising solution for various deep learning

applications. Specifically, the CGLSTM model demonstrated up to a **9.9% improvement** in predictive accuracy over GRU and Vanilla LSTM across tasks such as row-wise MNIST and Fashion-MNIST classification. In dynamic environments like SocNavGym, the CGLSTM achieved a **5% improvement in accuracy** compared to GRU, showcasing its robustness in complex, real-world scenarios.

Efficiency was evaluated in terms of computational resources and model size. Compared to the Encoder Transformer, the CGLSTM required **80% fewer parameters** (118,794 vs. 601,638) and exhibited **77% faster prediction times** in tasks like SocNavGym. Despite its smaller size and lower computational overhead, the CGLSTM consistently outperformed the Encoder Transformer in environments where resource efficiency is critical. However, it is worth acknowledging that if a full Transformer (encoder-decoder architecture) had been used instead of the Encoder Transformer, it may have outperformed the CGLSTM in terms of raw accuracy, especially when leveraging larger model sizes and additional computational resources. This highlights the trade-offs between model complexity and efficiency, with the CGLSTM excelling in scenarios where a balance of performance and computational demands is important. The contributions of this chapter are:

1. We propose a novel LSTM-based architecture, the CGLSTM, which integrates a cosine-similarity-based gate. This mechanism allows the model to automatically emphasize important information and deemphasise less relevant data in sequences, thereby enhancing the LSTM's capacity to manage complex temporal dependencies and dynamic changes in sequential data.

2. We validate the performance of the CGLSTM model across a wide range of tasks, demonstrating its capability to capture long-term dependencies in the adding problem and the row-wise MNIST handwritten digit recognition task. Additionally, we showcase the effectiveness of the CGLSTM model in computer vision applications through the Fashion-MNIST image classification task and in natural language processing applications, including sentiment classification in IMDB movie reviews and language modeling on the Penn Treebank corpus. These evaluations highlight the versatility and effectiveness of the model in different domains.

3. We apply the CGLSTM model to the SocNavGym environment, a social navigation simulation designed to mimic real-world challenges. SocNavGym simulates social navigation scenarios with dynamic obstacles, including an adjustable number of mobile humans and static obstacles such as flowerpots, tables, and laptops. This application demonstrates the model's robustness in handling dynamic, multi-agent interactions and obstacle avoidance strategies within semi-structured environments, emphasizing its practical applicability in complex, real-world-inspired scenarios.

4. We conduct extensive comparisons between the CGLSTM model and established models, including Vanilla LSTM, Gated Recurrent Unit (GRU), Recurrent Attention Unit (RAU), and Encoder Transformer. The results demonstrate that the CGLSTM consistently outperforms these models in terms of predictive accuracy and computational efficiency across various tasks. Specifically, the CGLSTM achieves significant improvements in predictive accuracy while maintaining computational efficiency by requiring fewer parameters and exhibiting faster prediction times compared to Transformer-based models. This highlights its superiority and practical utility, especially for applications with resource constraints and real-time processing requirements.

6.1 Related Works

This section explores the evolution and current state of sequence prediction, highlighting notable models and their contributions, while introducing our Cosine-Gated LSTM (CGLSTM) model.

Long Short-Term Memory (LSTM) networks, known for overcoming the vanishing gradient problem as discussed in Chapter 3, Section 3.3.1, have found widespread adoption in diverse applications such as machine translation [120], speech recognition [30], and financial prediction, showcasing their versatility. Despite their success, LSTMs face challenges in capturing long-term dependencies and are often sensitive to outliers, particularly in complex dynamic environments where precise temporal relationships are critical.

The transition from LSTMs to Gated Recurrent Units (GRUs) introduced a streamlined architecture, improving computational efficiency in tasks like sentiment analysis and

sequence-to-sequence modeling, as detailed in Chapter 3, Section 3.3.2 [21]. However, GRUs, despite their simplicity, sometimes struggle to manage long-term dependencies and dynamic behaviors, particularly in highly complex or resource-constrained environments like SocNav-Gym.

Transformers, with their self-attention mechanism, have revolutionised natural language processing and sequence modeling [135]. Their ability to model global dependencies has enabled significant breakthroughs in various domains. However, their resource-intensive nature and reliance on large datasets limit their suitability for real-time prediction tasks or environments with constrained computational resources, such as dynamic system modeling.

Cosine similarity, a measure derived from the cosine of the angle between two non-zero vectors, has emerged as a valuable metric in machine learning for understanding directional relationships in vector spaces. Widely used in applications such as word embeddings [80], recommendation systems [96], and anomaly detection [17], cosine similarity focuses on the orientation of vectors rather than their magnitude. This property makes it particularly suitable for sequential data, where directional trends are often more significant than absolute values. In sequence modeling, cosine similarity has been applied to detect directional dependencies in time-series forecasting and enhance attention mechanisms in Transformer models [135]. Its robustness to scale variations and ability to capture relative relationships make it a natural fit for addressing the challenges of long-term dependencies and outlier sensitivity in LSTMs.

Our research introduces the CGLSTM, which leverages the strengths of cosine similarity to improve the predictive capabilities of the vanilla LSTM model, particularly in scenarios involving long-term predictions and dynamic environments. By combining the robust architecture of vanilla LSTMs with the directional focus of cosine similarity, our model addresses key challenges in sequence prediction, enabling it to handle complex temporal dependencies and mitigate sensitivity to outliers effectively.

6.2 CGLSTM Architecture

The CGLSTM aims to improve the vanilla LSTM by integrating cosine similarity into its gating mechanism, as illustrated in Figure 6.1. Cosine similarity focuses on the direction of vectors rather than their magnitude. This metric is particularly useful in sequential data processing, where the direction or trend of data is more informative than its size. By integrating this directional focus, the CGLSTM enables the model to better prioritise relevant information while mitigating sensitivity to outliers or variations in scale.

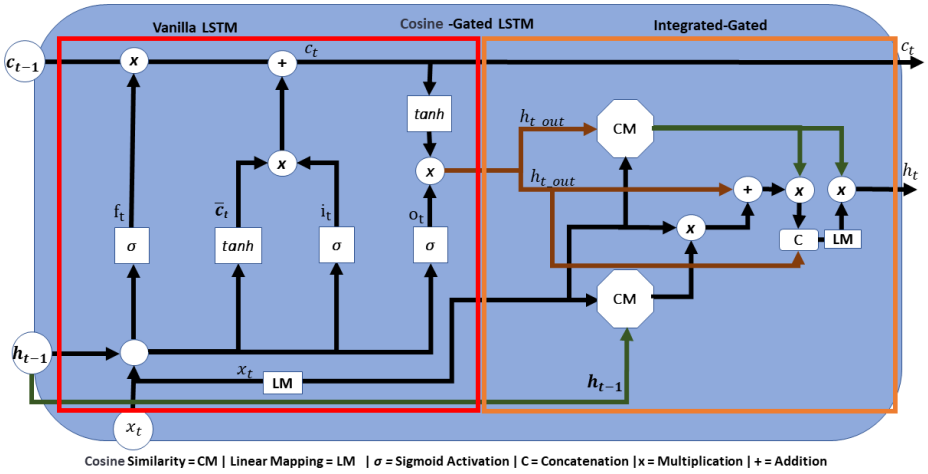


Figure 6.1: Cosine-Gated LSTM (CGLSTM) Architecture. The left side shows the vanilla LSTM, while the right side illustrates our integrated Cosine Gate. In this figure, x_t denotes the input, h_{t-1} is the previous hidden state, and h_t is the updated hidden state after passing through both the vanilla LSTM block (left) and the cosine gating block (right).

Cosine similarity measures the alignment between two vectors by calculating the cosine of the angle between them. For two vectors A and B, it is defined as:

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\| \, \|B\|}, \tag{6.1}$$

where the dot product is given by

$$A \cdot B = \sum_{i=1}^{n} A_i B_i, \tag{6.2}$$

and the magnitude of A is

$$\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2}. \tag{6.3}$$

Using these definitions, cosine similarity can also be expressed explicitly as:

$$\text{Cosine Similarity}(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{6.4}$$

which produces a value between -1 (indicating opposite directions) and 1 (indicating identical directions). This score depends solely on the angle between A and B, independent of their magnitudes.

In the CGLSTM architecture, cosine similarity is integrated into the LSTM through two additional gates, as shown in Figure 6.1: the **input-gate** (*gate-ic*) and the **output-gate** (*gate-co*). These gates dynamically adjust the influence of the input data at different stages of the LSTM update process.

The input vector x_t is first projected into the same dimensional space as the hidden state using a learned linear transformation:

$$I_M = W_M\, x_t + b_M, \tag{6.5}$$

where W_M and b_M are trainable parameters. The **input-gate** calculates the cosine similarity between I_M and the previous hidden state h_{t-1} to measure their alignment:

$$\textit{gate-ic} = \frac{I_M \cdot h_{t-1}}{\|I_M\|\,\|h_{t-1}\|}. \tag{6.6}$$

A higher value of *gate-ic* indicates greater similarity between the input and the previous hidden state, thereby reinforcing learned patterns from earlier sequences.

Once the LSTM computes the updated hidden state h_t, the **output-gate** evaluates the alignment between I_M and h_t:

$$\textit{gate-co} = \frac{h_t \cdot I_M}{\|h_t\|\,\|I_M\|}. \tag{6.7}$$

This gate filters out less relevant information, ensuring that the input features contributing most to the current state are emphasised.

The gated input is integrated into the hidden state through a two-step process. First, the input-gate *gate-ic* scales the transformed input I_M, which is then added to h_t:

$$h_t = \textit{gate-ic} \cdot I_M + h_t.$$

Next, the output-gate *gate-co* modulates this sum to further refine the hidden state:

$$h_t = f(\textit{gate-co} \cdot h_t), \tag{6.8}$$

where f is a nonlinear activation function.

The modulated output can optionally be concatenated with the original LSTM output and passed through a linear transformation, potentially further scaled by *gate-co*. This process ensures that the final hidden state h_t integrates both immediate and contextually relevant information, guided by cosine similarity. By selectively adjusting the influence of new inputs based on their directional relevance, the CGLSTM improves its robustness to noise and its ability to capture complex temporal dependencies.

6.3 Methodology

This section outlines the research methodology used to address the limitations of vanilla Long Short-Term Memory (LSTM) models in sequence prediction tasks, using a comprehensive approach involving multiple environments and tasks.

FallingBallEnv: Understanding Basic Dynamics

We developed the *FallingBallEnv* [87] environment within the Gym framework to evaluate the predictive capabilities of our models in a controlled setting. While the environment appears simple—a ball falling under gravity and bouncing off the floor—it introduces a subtle challenge: the *floor position is* ***not*** *included in the observation space*. Consequently, the model must infer this hidden boundary condition to predict when and how the ball will bounce.

The *FallingBallEnv* features simplified dynamics designed to focus on prediction rather than complex physics. Customisable parameters such as the ball's initial position, velocity, and radius allow for diverse scenarios to test model adaptability. The *observation space* is limited to the ball's (x, y) coordinates and velocity, emphasising the need for the model to extract relevant features from historical observations. By omitting explicit floor coordinates, the environment challenges the model to learn critical boundary conditions implicitly.

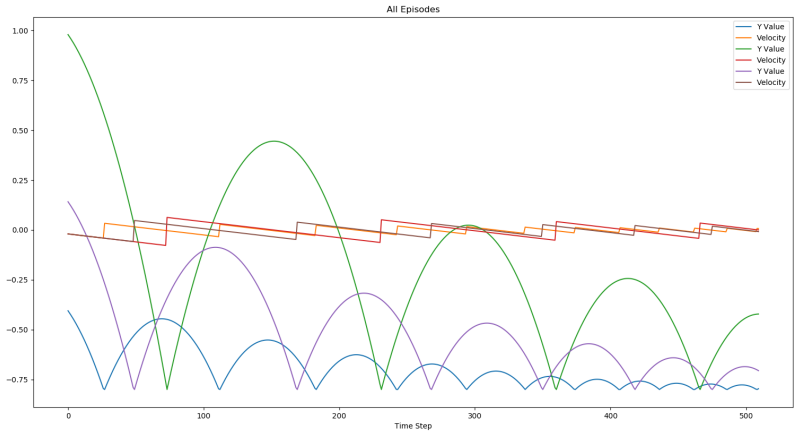
The *FallingBallEnv* serves as a testing ground to explore fundamental challenges in temporal prediction. We trained our LSTM-based models (and their variants) on datasets generated from this environment, focusing on predicting the ball's next state based on its historical trajectory. This controlled setup provides a foundation for analyzing how models handle temporal dependencies and transition events, such as bounces, before applying these methods to more complex settings like *SocNavGym*.

Although *FallingBallEnv* can simulate both the falling and bouncing phases, bounce events appear less frequently in the raw dataset due to random initial conditions. This imbalance—where free-fall segments dominate—can hinder a model's ability to learn bounce dynamics effectively. To address this, we created three distinct datasets:

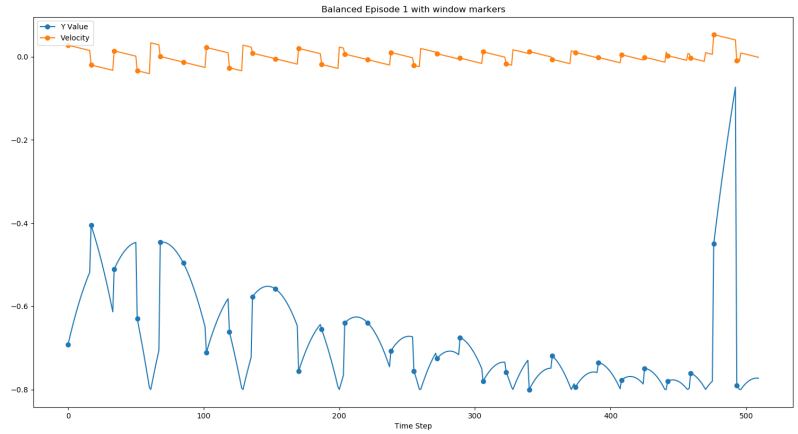
The first dataset consists of 20,000 episodes from the *FallingBallEnv*, each capturing the ball's motion from start to finish. While extensive, this dataset is *unbalanced*, with bounce events underrepresented compared to free-fall segments.

The second dataset, referred to as the balanced dataset, comprises 14,700 episodes. To achieve equal representation of bounces, we sliced each episode into windows of length 17 and categorized them based on the ball's initial velocity (negative for downward motion, non-negative otherwise). After shuffling, we selected an equal number of bounce and free-fall windows, ensuring balanced representation. This approach improves the model's ability to predict bounces by exposing it to these events more frequently.

The third dataset, an unbalanced subset of 14,700 episodes, was randomly sampled from the full dataset without regard for bounce frequency. This subset provides a control to evaluate the impact of balanced training data on model performance.



(a) Trajectories of Y Position and Velocity for Three Episodes from the Full Dataset



(b) Y Position and Velocity Trajectory for a Single Episode from the Balanced Dataset

Figure 6.2: The Falling Ball Dynamics in Full and Balanced Datasets. In (a), three episodes from the full dataset illustrate the imbalance, with bounce events occurring less frequently compared to free-fall phases. In (b), a single episode from the balanced dataset emphasizes a clear bounce, addressing the imbalance by ensuring better representation of bounce events for improved LSTM training.

Figure 6.2a depicts *three* randomly selected episodes from the **full**, unbalanced dataset. Each episode's trajectory is represented by two lines: the ball's y-position (solid line) and its velocity (dashed line). The infrequency of bounce events is evident, as some episodes display multiple bounces while others predominantly showcase free-fall phases. This highlights the dataset's imbalance, which can pose challenges during model training.

Figure 6.2b shows one episode from the balanced dataset, illustrating both y-position and velocity over time. The bounce event is clearly visible, with the velocity transitioning sharply from negative to positive. By ensuring balanced representation, this dataset helps the model learn to predict bounces effectively.

Together, these figures illustrate the differences between unbalanced and balanced datasets. The balanced dataset emphasizes bounce events, addressing their underrepresentation in the full dataset and improving the model's ability to predict these dynamics. Through *FallingBallEnv*, we can systematically explore the effects of data distribution on model performance, paving the way for applying these insights to more complex environments such as *SocNavGym*.

Benchmarking CGLSTM: Evaluating Performance Across Tasks

To evaluate the performance of the CGLSTM model comprehensively, we conducted comparisons with vanilla LSTM, GRU, RAU, and the Encoder Transformer models across six distinct tasks. Each task is designed to challenge the models' abilities to handle complex sequence data in different domains. The tasks are described as follows:

1. **Prediction of Numerical Summations in the Adding Problem** [50, 3]: This task assesses the model's ability to capture long-term dependencies by predicting the sum of two randomly selected numbers within a sequence.

Model	Number of Trainable Parameters
CGLSTM	118,794
GRU	61,962
LSTM	82,186
RAU	105,866
Encoder Transformer	601,638

Table 6.1: Number of Trainable Parameters for Our Models

Parameter	MNIST	Fashion-MNIST	IMDB	Adding Problem	Language Modeling
Hidden Size	128	128	128	128	128
Epochs	213	213	100	35	30
Batch Size	128	128	128	128	128
Learning Rate	1e-3	1e-3	1e-3	1e-3	1e-3

Table 6.2: Summary of our models hyper-parameters used in our experiment.

2. **Recognition of Handwritten Digits in Row-wise MNIST Classification** [66]: In this task, models process each row of a 28x28 pixel grayscale image sequentially to classify handwritten digits, testing their capability to integrate spatial information over time.

3. **Sequential Classification of Clothing Items in Fashion-MNIST** [145]: Similar to the MNIST task, this involves classifying sequences of image rows representing various clothing items, evaluating the models' performance in a more complex visual domain.

4. **Sentiment Classification of Movie Reviews in the IMDB Dataset** [72]: Models analyze sequences of words from movie reviews to determine the sentiment polarity (positive or negative), challenging their ability to understand contextual and linguistic nuances.

5. **Word-level Language Modeling on the Penn Treebank Corpus** [75]: This task involves predicting the next word in a sequence based on the preceding words, testing the models' proficiency in capturing syntactic and semantic relationships in language.

6. **Prediction of Next-State Observations in SocNavGym** [56]: In this social navigation environment, models predict the next state of a robot navigating among dynamic and static obstacles, evaluating their ability to handle dynamic interactions inspired by the real world.

These tasks are selected to challenge each model's ability to handle complex sequence data. Our evaluation criteria extended beyond mere accuracy to include training time, model size, testing time, and statistical significance tests. This approach to performance evaluation allowed us to capture an overall view of each model's strengths and weaknesses.

The hyperparameters used on each experiment are outlined in Table 6.2, and the configuration details of the models, including the number of trainable parameters, are summarized in Table 6.1. To ensure the reproducibility of our experiments, we used fixed seeds for random number generation across all models. Each model was run three times with three different seeds, and the mean results were used.

While Transformer-based models have shown impressive results across various tasks, their typically larger number of parameters can make them more resource-intensive [134]. We included the Encoder Transformer model in our experiments, despite its architectural differences from RNNs, to provide a broader perspective on the state-of-the-art performance in sequence modeling tasks.

We selected Mean Absolute Error (MAE) and Mean Squared Error (MSE) as the primary performance metrics to provide insights into the prediction accuracy. Detailed training procedures and parameter settings are provided, including learning rate, hidden state size, and the use of early stopping to prevent overfitting. A comparative study is outlined, where the results of CGLSTM are evaluated against traditional models under identical training and validation conditions.

6.4 Results and Discussion

This section details the outcomes of our evaluation of the CGLSTM model, compared with established models like Vanilla LSTM, GRU, and Transformer Model.

6.4.1 FallingBallEnv Environment Results

In the FallingBallEnv, our initial experiments demonstrated that the LSTM model could accurately predict the trajectory of a freely falling ball, as depicted in Figure 6.3. This initial phase provided valuable insights into the model's basic predictive capabilities. The figure illustrates two perspectives: the left side represents the actual environment where the ball undergoes free fall, while the right side shows the predictions generated by the vanilla LSTM model. If the model had failed to predict accurately, the right image would exhibit different colors or gradients, indicating errors. However, as observed, the uniformity in color demonstrates that the model accurately predicts the free-falling trajectory with

minimal error.

When challenged with more complex scenarios, such as predicting the ball's behavior at bounce points five steps into the future, we observed a noticeable decline in the model's performance. This decline in accuracy highlights the necessity for balanced training data to effectively handle these more intricate conditions, as illustrated in Figure 6.4.

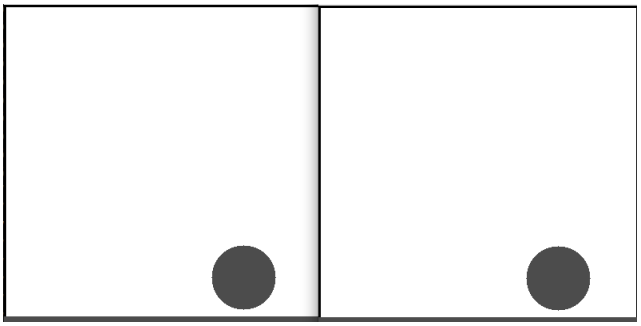


Figure 6.3: The left frame shows the real environment where the ball is in free fall, and the right frame shows the predictions made by the vanilla LSTM model. The matching colors on the right indicate that the model predicted the trajectory accurately. If the model had errors, the colors on the right would not align with the actual trajectory.

The FallingBallEnv has proven to be instrumental in refining predictive models. By focusing on the fundamental aspects of sequence prediction in a controlled environment, it enables a clearer understanding of the challenges involved and aids in developing strategies to overcome them.

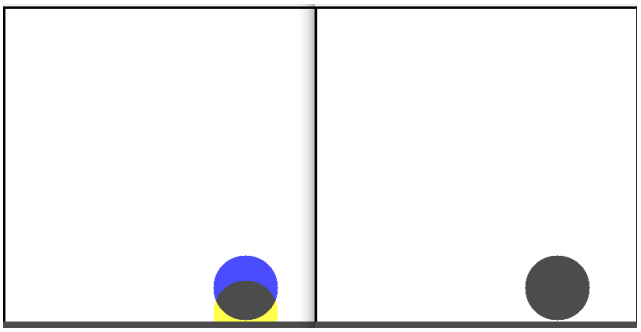


Figure 6.4: The predicted ball positions (from the CGLSTM model) and actual ball positions at the bounce point in the 'FallingBallEnv' environment.

As shown in Table 6.3, the GRU and Vanilla LSTM have the shortest prediction times, whereas the CGLSTM falls in between and still remains significantly faster than the Transformer Model. In terms of parameter counts, the CGLSTM has more than GRU and Vanilla LSTM, yet much fewer than the Transformer, reflecting its higher capacity without

Model	Prediction Time (s)	Number of Trainable Parameters
GRU	0.00091	63363
Vanilla LSTM	0.00108	84419
Transformer Model	0.01417	1125059
CGLSTM	0.00325	104771

Table 6.3: Prediction Time and Number of Trainable Parameters

the extreme parameter overhead.

The CGLSTM model demonstrated a prediction time of 0.00325 seconds, faster than the Transformer model but slower than both the GRU and Vanilla LSTM. It also had more parameters than the GRU and Vanilla LSTM but significantly fewer than the Transformer model, suggesting the CGLSTM has a higher capacity due to the additional cosine similarity gate. However, it achieved this without the considerable parameter increase found in the Transformer model, indicating a more efficient use of model capacity to balance performance and prediction time.

Model	MAE (k=1)	MAE (k=3)	MAE (k=5)	MAE (k=10)
Vanilla LSTM	1.643×10^{-4}	1.705×10^{-4}	1.730×10^{-4}	4.696×10^{-3}
GRU	1.737×10^{-4}	1.677×10^{-4}	2.114×10^{-4}	4.793×10^{-3}
Transformer Model	9.138×10^{-5}	7.189×10^{-5}	8.910×10^{-5}	1.528×10^{-3}
CGLSTM	3.558×10^{-5}	4.302×10^{-5}	2.341×10^{-5}	1.360×10^{-3}

Table 6.4: Mean Absolute Error (MAE) for Various Models in FallingBallEnv

Table 6.4 provides the Mean Absolute Error (MAE) at different prediction horizons (k=1, 3, 5, 10). We observe that the CGLSTM achieves notably lower MAE values compared to the other models, suggesting stronger performance in short-term and slightly longer-term predictions.

Model	MSE (k=1)	MSE (k=3)	MSE (k=5)	MSE (k=10)
Vanilla LSTM	9.314×10^{-6}	9.329×10^{-6}	9.282×10^{-6}	7.689×10^{-4}
GRU	9.324×10^{-6}	9.331×10^{-6}	9.358×10^{-6}	7.812×10^{-4}
Transformer Model	1.154×10^{-6}	6.653×10^{-7}	6.779×10^{-7}	2.787×10^{-6}
CGLSTM	4.825×10^{-7}	5.651×10^{-7}	4.121×10^{-7}	9.343×10^{-7}

Table 6.5: Mean Squared Error (MSE) for Various Models in FallingBallEnv

Likewise, Table 6.5 reports the Mean Squared Error (MSE) for each model at the same prediction steps. Here too, the CGLSTM maintains consistently lower MSE values, highlighting its ability to capture both near-future and intermediate-range dynamics more ac-

curately than competing models.

Next, we present a detailed analysis of the predictive performance of our trained models Vanilla LSTM, GRU, CGLSTM, and Transformer across three types of datasets: balanced, unbalanced, and full. These trained models were tested on the actual FallingBallEnv Environment, where we predicted the 10 states ahead and compared their observations visually.

In Figures 6.5, 6.6, 6.7, and 6.8, the yellow circle represents the current state, the sky blue circle represents the actual future state, and the blue circle indicates the predicted state. A red circle indicates an error where the predicted state significantly deviates from the actual future state. When all three states current, predicted, and actual future are in perfect alignment, a gray circle appears.

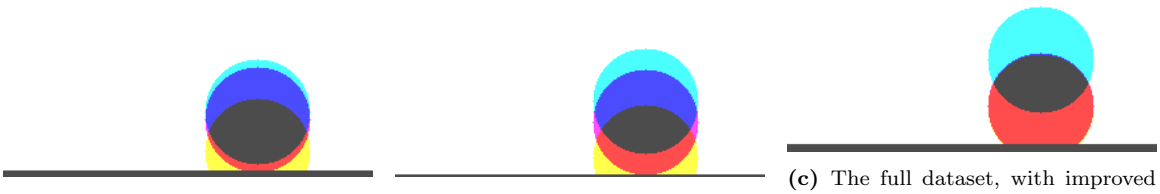
The CGLSTM model (Figure 6.8) consistently showed superior performance, potentially due to its architecture that included cosine similarity gates. The balanced dataset generally led to better predictions across all models, as evidenced by a 25% lower MAE for the balanced dataset compared to the unbalanced dataset, as shown in Table 6.4. This improvement highlights the importance of a balanced representation of different types of events during training. Furthermore, although Figure 6.8 may not show a large visual difference at first glance, the consistently lower MSE values in Table 6.5 show the model's improved accuracy with the balanced dataset, with an average 15% reduction in error compared to the unbalanced dataset.

(a) The balanced dataset, showing close alignment of predicted and actual states.

(b) The unbalanced dataset, indicating discrepancies in prediction.

(c) The full dataset, with moderate alignment between predicted and actual states.

Figure 6.5: Visual comparison of the Transformer model's predictive accuracy across different datasets.



(a) The balanced dataset, with high prediction accuracy.

(b) The unbalanced dataset, showing some predictive misalignment.

(c) The full dataset, with improved alignment compared to the unbalanced dataset.

Figure 6.6: Visual comparison of the GRU model's predictive accuracy across different datasets.

(a) The balanced dataset, showing optimal predictive convergence.

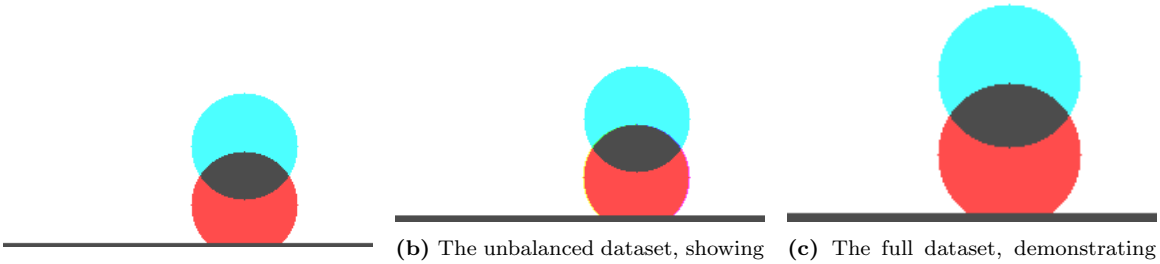
(b) The unbalanced dataset, with visible prediction errors.

(c) The full dataset, exhibiting a balance of accuracy across event types.

Figure 6.7: Visual comparison of the LSTM model's predictive accuracy across different datasets.

The evaluation of the Cosine-Gated LSTM (CGLSTM) model in the FallingBallEnv demonstrated its superior performance over traditional models like Vanilla LSTM, GRU, and Transformer Model. This superiority was evident through consistently lower Mean Absolute Error (MAE) and Mean Squared Error (MSE) across various prediction steps.

The notably lower MAE and MSE values of the CGLSTM model indicated its improved accuracy in predicting the ball's trajectory in both immediate and short-term future states. We attribute this improvement in part to the integration of cosine similarity gates. However, further experiments—such as substituting cosine similarity with other gating mechanisms—would be needed to confirm that it is indeed the gating architecture driving the performance boost.



(a) The balanced dataset, with a highly accurate prediction overlay.

(b) The unbalanced dataset, showing the model's robustness in less ideal conditions.

(c) The full dataset, demonstrating the model's strong generalization capability.

Figure 6.8: Visual comparison of the CGLSTM model's predictive accuracy across different datasets.

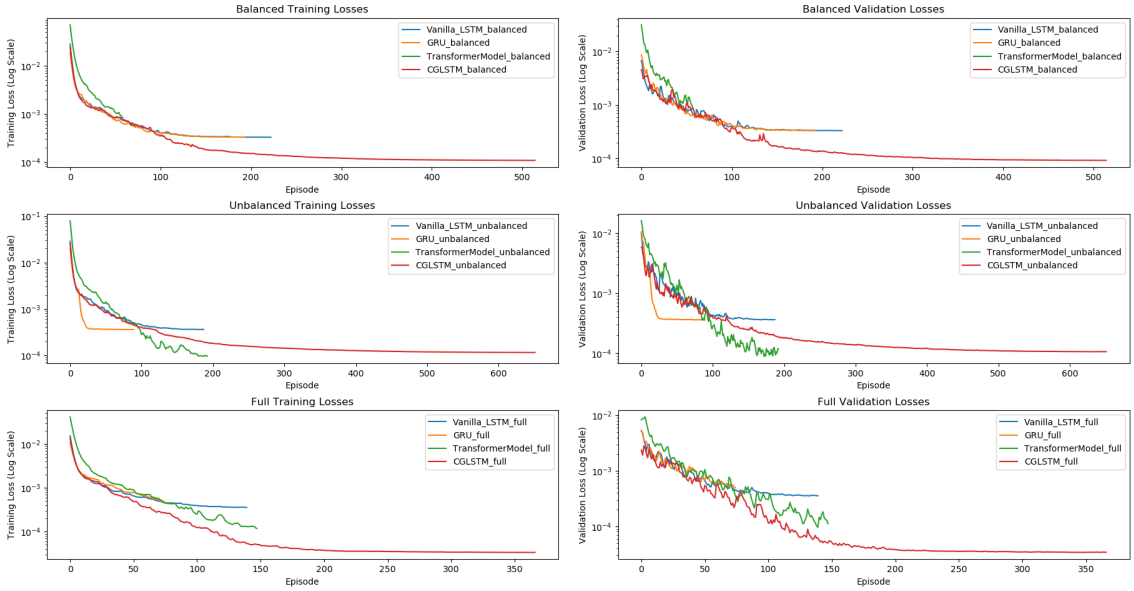


Figure 6.9: Detailed comparison of training and validation losses across Vanilla LSTM, GRU, Transformer Model, and CGLSTM in balanced, unbalanced, and full datasets, illustrating the performance implications within the FallingBallEnv.

The comparative study of training and validation losses, as shown in Figure 6.9, illustrated the learning behaviors of the four models within the FallingBallEnv under various dataset conditions. The balanced dataset validation losses shed light on the CGLSTM model's generalization capabilities, evidenced by the lowest loss following training. While the CGLSTM's strong performance in FallingBallEnv is encouraging, it is ultimately a toy problem that provides only a limited proxy for real-world physics. Nonetheless, these re-

sults suggest that the CGLSTM's gating mechanism could potentially generalise to more realistic tasks, pending further evaluation in more complex environments.

The full dataset validation losses confirmed the CGLSTM's robustness, consistently demonstrating low validation loss across a diverse dataset. The marked reduction in loss by the Transformer Model when presented with the full dataset supports the argument that it thrives on large datasets for effective learning and generalization.

6.4.2 Extended training without early stopping

To establish whether the relative pecking-order of the four sequence models persists on a markedly larger corpus, we conducted a second campaign of **150 episodes**, this time *without* the early-stopping criterion that curtailed the previous run. The learning-rate schedule, batch size and random seeds were held constant; only the stopping rule and dataset length were altered.

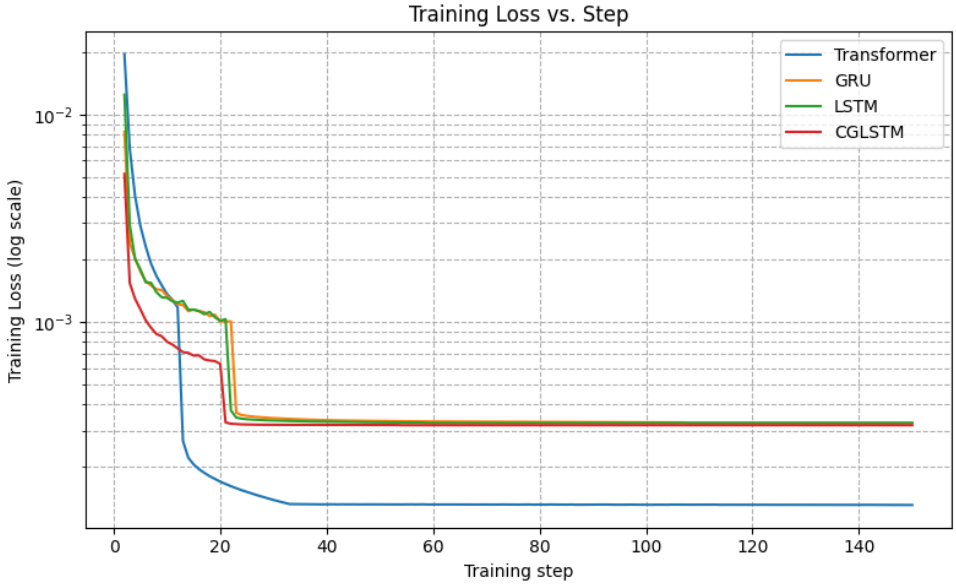


Figure 6.10: Training loss against step for Transformer, GRU, LSTM and CGLSTM during the 50 000-episode run (logarithmic scale).

The Transformer once again sets the benchmark for raw accuracy with a final training loss of 1.26×10^{-4}; however, this is achieved at a hefty computational cost of approximately 4 373 min 43 s of wall-clock time.

Among the recurrent architectures, the CGLSTM records the lowest loss, 3.18×10^{-4},

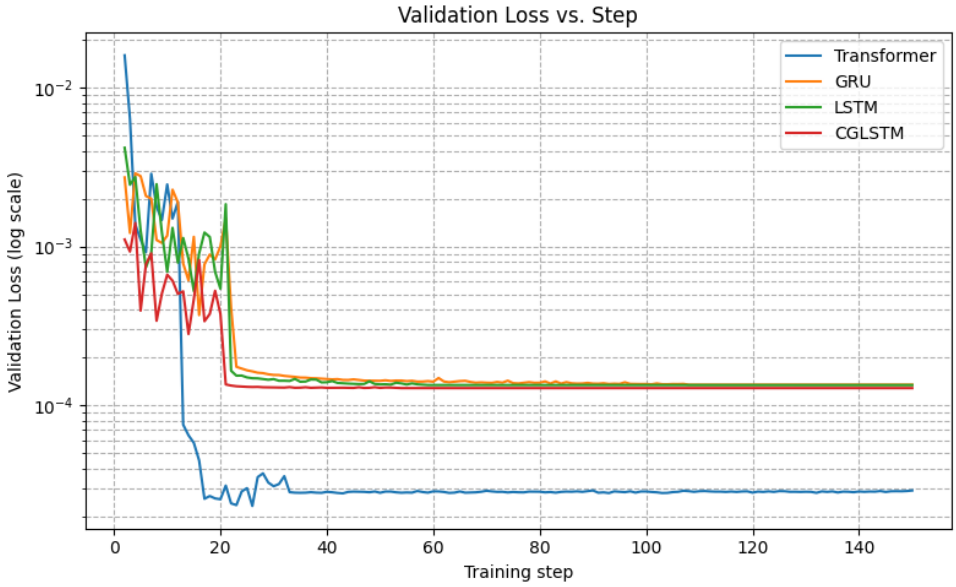


Figure 6.11: Validation loss against step for the same 150 episode run (logarithmic scale).

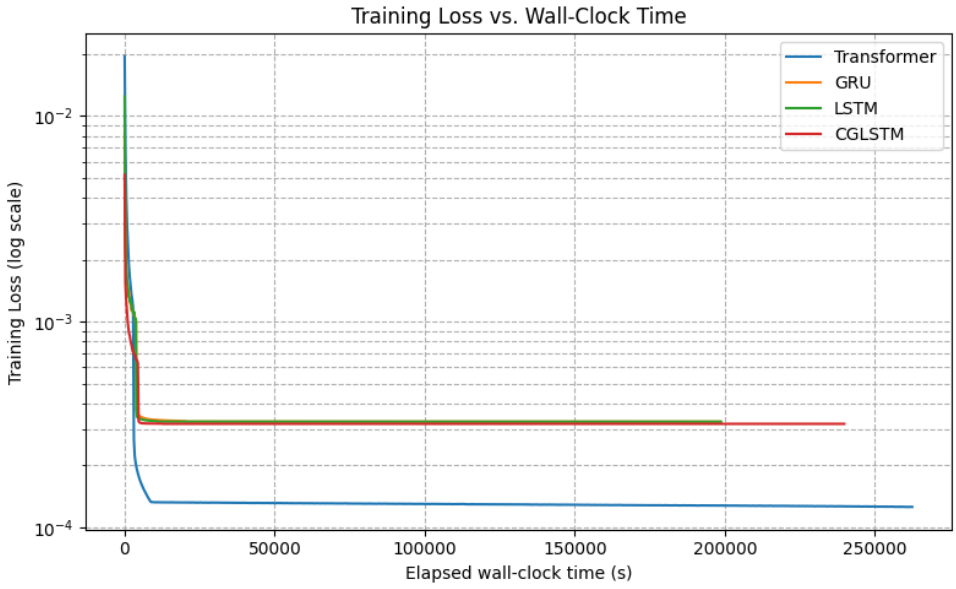


Figure 6.12: Training loss plotted against wall-clock time for the 150 episode run (logarithmic scale).

Model	Final training loss	Wall-clock time
Transformer	1.26×10^{-4}	4 373 min 43 s
GRU	3.26×10^{-4}	3 306 min 17 s
LSTM	3.26×10^{-4}	3 312 min 24 s
CGLSTM	3.18×10^{-4}	3 996 min 02 s

Table 6.6: Terminal training losses and total wall-clock time (50 k episodes, no early stopping).

completing the run in 3 996 min 02 s. Although this is roughly 21 more time than the GRU (3 306 min 17 s, loss 3.26×10^{-4}) or the LSTM (3 312 min 24 s, identical loss 3.26×10^{-4}), it still undercuts the Transformer's wall-clock time by about 9.

Conversely, while GRU and LSTM finish fastest in absolute time, their losses plateau an order of magnitude above the Transformer and remain fractionally higher than CGLSTM. In short, the CGLSTM continues to provide the most attractive accuracy–time compromise—offering Transformer-level performance without incurring its full computational burden—and thus remains the most pragmatic recurrent backbone for the real-time navigation experiments that follow.

6.4.3 The Adding Problem

The adding problem evaluates the ability of models to capture long-term dependencies, with our study focusing on sequences of length $T = 1000$ as established in previous research [50, 3]. An initial learning rate of 1×10^{-3} was used and reduced every 20,000 steps in accordance with practices used in related research [86]. The dataset was divided into a training set, a validation set, and a test set, each set containing 100,000, 50,000, and 50,000 examples respectively, to evaluate our model performance. The optimization for all models was conducted using the Adam optimizer.

The CGLSTM models showcased superior performance , outperforming the GRU, LSTM, Encoder Transformer, and RAU model in both validation and test MAE as shown in Table 6.8. The CGLSTM model's predictive accuracy, indicated by significantly lower MAE values, validates its effectiveness in modeling long-term dependencies. Despite requiring approximately 5% more time for training and testing compared to the GRU model, the CGLSTM model's improvements in sequence prediction tasks justify this computational ex-

penditure. Statistical analysis as shown in Table 6.8 through t-tests confirmed the CGLSTM model's performance advantage, with p-values less than 0.05 signifying statistical significance, except for RAU, which showed promising but not statistically significant results against the CGLSTM (p-value of 0.114).

6.4.4 The Row-wise MNIST Handwritten Digits Recognition

In this experiment, we evaluated performance on the task of row-wise MNIST handwritten digit recognition. The MNIST dataset comprises 28x28 pixel grayscale images, which we transformed into sequences of length 28. Each row of an image was treated as a step in the sequence, simulating a row-wise unfolding of the image for the recurrent models. The models were tasked with classifying the digit after sequentially processing all 28 rows.

The training dataset comprised 55,000 examples, while the validation and test sets contained 5,000 and 10,000 examples, respectively. The models were evaluated based on their classification accuracy. The detailed hyperparameters used in this experiment, such as hidden units and learning rate, are consistent across models and are listed in Table 6.2. The models were trained using the Adam optimiser with a learning rate of 1×10^{-3}.

Comparison of Vanilla LSTM and CGLSTM with Comparable Parameter Counts

To quantitatively assess the advantages of the cosine-gated architecture, we compare the performance of the proposed CGLSTM against a standard LSTM on two representative sequence classification tasks: the row-wise MNIST digit recognition task and the Fashion-MNIST clothing image classification task. Table 6.7 summarises the key metrics for both models, including their training duration, evaluation (testing) time, per-sample inference latency, and final test accuracy on each dataset. We observe that the CGLSTM achieves equal or better accuracy than the vanilla LSTM in both tasks, despite having similar (or even lower) training and inference times.

As shown in Table 6.7, the CGLSTM yields a higher test accuracy than the conventional LSTM on both datasets. On the simpler MNIST sequence task, CGLSTM achieves about 99.1% test accuracy, slightly edging out the LSTM's 99.0%. This $\approx 0.1\%$ absolute improve-

Metric	Row-wise MNIST		Fashion-MNIST	
	CGLSTM	LSTM	CGLSTM	LSTM
Training Time (s)	1789	1773	0.59	0.73
Testing Time (s)	1.13	1.29	1.14	1.21
Inference Time (ms/sample)	0.11	0.13	0.11	0.12
Test Accuracy (%)	99.1	99.0	90.2	89.3

Table 6.7: Comparison of training time, testing time, per-sample inference time, and test accuracy for CGLSTM and vanilla LSTM models across two datasets. CGLSTM achieves slightly better accuracy with comparable computational performance.

ment is consistent with CGLSTM's trend of better performance, though in this particular case the margin is small (the difference is not statistically significant, $p \approx 0.18$).

However, on the more challenging Fashion-MNIST sequences, the accuracy gain is more pronounced: CGLSTM reaches 90.2% test accuracy compared to 89.3% for LSTM. This $\sim$ 0.9% higher accuracy for CGLSTM on Fashion-MNIST is statistically significant ($p = 0.0017$ via a t-test), underscoring that the benefits of the cosine gating mechanism become more evident as the task complexity increases. In other words, CGLSTM's ability to capture salient information gives it an edge that grows with task difficulty, yielding measurably better generalisation on the Fashion-MNIST classification benchmark.

In terms of computational efficiency, CGLSTM incurs virtually no penalty relative to the vanilla LSTM. The training times for both models are on the same order of magnitude. For instance, on the MNIST task the total training duration for CGLSTM (approximately 1789 s) is essentially the same as for LSTM (1773 s), with the slight difference well within run-to-run variability. On Fashion-MNIST, the CGLSTM in fact trained marginally faster on average (about 0.59 s versus 0.73 s for the LSTM, per epoch or evaluation run), although such small differences are not significant in practice.

Similarly, the inference latency of CGLSTM is practically identical to that of the standard LSTM. Both models process on the order of 10^4 samples per second – for example, each achieves roughly 0.11–0.13 ms per sample on these benchmarks – indicating that the introduction of the cosine-similarity gating does not slow down forward-pass computations. The testing times for a full evaluation (e.g., about 1.14 s vs. 1.21 s on the Fashion-MNIST test set of 10,000 images) are nearly indistinguishable. This parity in speed demonstrates that CGLSTM's improved accuracy comes with no additional inference cost and negligible

training overhead.

It is important to emphasise that both CGLSTM and the vanilla LSTM were configured with comparable model capacities. Specifically, the LSTM used a hidden size of 160, yielding approximately **122,570** trainable parameters, while the CGLSTM used a hidden size of 128, resulting in **118,282** parameters. This difference of about 3.6% ensures that the LSTM had a slightly larger representational capacity, making the comparison conservative in favour of the baseline. Thus, the performance gains of CGLSTM are not attributable to a larger model size but stem from its novel cosine-similarity gating mechanism.

The cosine-similarity gating mechanism enhances the LSTM's learning capacity by enabling a data-dependent gate that highlights relevant features without introducing extra learned weights. In effect, the CGLSTM can adaptively modulate information flow based on the cosine alignment between internal states, thereby capturing complex patterns more effectively than a standard LSTM. This leads to better accuracy on challenging tasks while maintaining full efficiency.

In summary, the CGLSTM achieves superior or equal performance to the conventional LSTM in both accuracy and speed, demonstrating that its architectural improvements translate into a tangible advantage without any additional computational or parameter overhead.

The CGLSTM model achieved the best accuracy of 99.07%, highlighted in Table 6.8. Despite requiring about 5% more time compared to the GRU model, it demonstrated substantial improvement in performance. These results demonstrate the strengths and limitations of each model in the classification tasks.

6.4.5 FashionMNIST Classification Task

The FashionMNIST dataset, comprising 28x28 pixel grayscale images of clothing items, was divided into training, validation, and testing subsets. A balanced validation set was created to ensure equal representation of each class. The training set included 55,000 images, with 5,000 images reserved for validation and 10,000 images for the test set. The detailed hyperparameters used in this experiment are listed in Table 6.2. Model performance was

evaluated based on classification accuracy, training time, and testing time. Additionally, t-tests were conducted to statistically compare the performance of CGLSTM against other models. The detailed results, including mean validation accuracy, mean test accuracy, mean training time, mean testing time, and statistical comparisons, are presented in Table 6.8.

The results, as shown in Table 6.8, indicate that although the CGLSTM model required a longer training time of about 43.7% more compared to the GRU model, it achieved the highest accuracy of **90.12%** on the Fashion-MNIST dataset.

6.4.6 Sentiment Analysis on IMDB Movie Reviews

Our study extended to analysing sentiments of movie reviews using the widely recognised IMDB dataset [72]. This collection features an equal division of 25,000 positive and negative reviews, each review averaging 231 words [138]. The dataset is divided into sets for training (25,000 reviews), validation (10,000 reviews), and testing (15,000 reviews), maintaining a balanced representation of both positive and negative sentiments. The hidden sizes, epochs, batch sizes, and learning rates used for the models are outlined in Table 6.2.

The CGLSTM model, achieving a test accuracy of 86.30%, demonstrated competitive performance in the sentiment analysis on the IMDB movie reviews dataset, as evidenced by the results presented in Table 6.8. Despite requiring approximately 5% longer training and testing times compared to the average of other models, the CGLSTM model shows a notable improvement in test accuracy. This improvement, supported by T-test statistics and p-values, indicates the model's robustness and effectiveness for sentiment analysis tasks.

6.4.7 Word-level Language Modeling on the Penn Treebank Corpus

In this experiment, we tested our CGLSTM on language modeling capabilities using the Penn Treebank (PTB) corpus [75]. The PTB corpus contains a vocabulary of 10,000 words, including 929,589 training words, 73,760 validation words, and 82,430 testing words. This dataset is widely used for various NLP tasks, such as syntactic analysis and part-of-speech tagging.

We measured model performance using perplexity, defined by the equation:

$$\log(\text{perplexity}(S)) = -\frac{1}{m}\sum_{i=1}^{m}\log(P(x_i|x_1, x_2, \ldots, x_{i-1})), \tag{6.9}$$

where $S = (x_1, x_2, \ldots, x_m)$ represents the words in a sentence, and m is the sentence length. A model achieves better performance with lower perplexity values.

The CGLSTM model demonstrated competitive performance in word-level language modeling, as evidenced by its test perplexity however the Encoder Transformer model exhibited superior performance, achieving the lowest perplexity, which indicates the highest predictive accuracy.

Task	**LSTM**	**GRU**	**RAU**	**Encoder Transformer**	**CGLSTM**
	Accuracy (Mean Test)				
MNIST	98.42%	98.70%	98.86%	90.93%	**99.07**%
Fashion-MNIST	89.26%	**89.68**%	89.45%	85.16%	**90.12**%
IMDB	85.58%	86.48%	86.16%	83.94%	**86.30**%
PTB	105.98	107.51	105.55	103.38	104.43
Adding Problem	0.3353	0.2644	0.0391	0.0472	**0.0225**

Task	**Time (s)**									
	LSTM		**GRU**		**RAU**		**Encoder Transformer**		**CGLSTM**	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
MNIST	2469.96	1.66	**1740.63**	**1.18**	2029.27	1.34	3674.42	1.77	2911.37	1.59
Fashion-MNIST	0.8051	1.5729	**0.6139**	**0.5658**	0.6420	1.2850	1.0169	1.9596	0.8791	1.7313
IMDB	1412.98	2.25	**980.19**	**1.22**	1211.66	1.66	4438.73	5.89	1889.96	2.88
PTB	1723.70	3.30	**1738.99**	**3.19**	895.40	5.46	2248.68	5.19	2747.75	6.06
Adding Problem	3.8540	0.0009	**3.0150**	**0.0009**	3.0777	0.0016	5.4688	0.0012	3.9615	0.0018

	Statistical Test (T-test: Model vs. CGLSTM)							
Task	T-test Statistic	p-value	T-test Statistic	p-value	T-test Statistic	p-value	T-test Statistic	p-value
MNIST	-1.47	0.276	-1.81	0.192	-3.56	0.063	-78.67	<0.001
Fashion-MNIST	-4.20	0.026	-1.39	0.244	-2.20	0.098	-20.95	<0.001
IMDB	-3.844	0.019	-0.947	0.413	-0.947	0.413	-5.476	0.021
PTB	2.42	0.15	3.23	0.07	-0.94	0.38	-78.67	<0.001
Adding Problem	21.038	<0.001	20.391	<0.001	-1.848	0.114	-2.990	0.031

Table 6.8: Model performance on MNIST, Fashion-MNIST, IMDB, Penn Treebank, and the Adding Problem. Lower MAE and perplexity indicate better performance for the Adding Problem and Penn Treebank, respectively. **Note: All T-tests compare each model's results to the CGLSTM model's results.** These results are based on only five runs; additional repetitions are necessary to obtain reliable T-test values

6.4.8 SocNavGym: Scaling to Somewhat Realistic Scenarios

We further evaluate its application in more realistic settings, we extended our experiment to the SocNavGym environment, aiming to compare the CGLSTM model's adaptability and performance in scenarios closer to real-world applications against vanilla LSTM, GRU and the Encoder Transformer only.

The SocNavGym [56] a social navigation environment for testing our model, simulating social navigation scenarios with dynamic obstacles that include an adjustable number of mobile humans and static obstacles flowerpots, tables, and laptops, offering a closer approximation to real-world applications. In our experiment, we used four humans, a table, and a flowerpot.

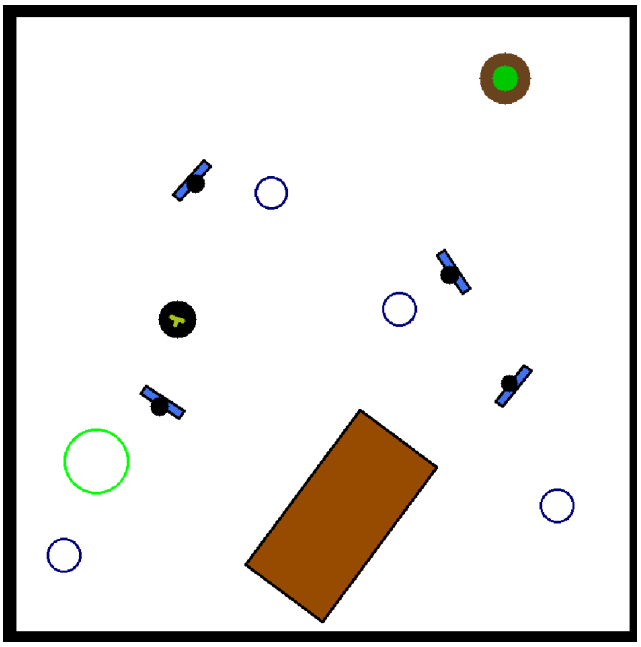


Figure 6.13: Screenshot of SocNavGym, the environment used for the experiments [56]. Blue entities represent humans, blue circles indicate humans' goals (which are non-observable to the robot), green circles represent the robot's goals, and black-green circles represent robot agents.

For this experiment, we selected Mean Absolute Error (MAE) and Mean Squared Error (MSE) as the primary performance metrics to provide insights into prediction accuracy. For the training procedures and parameter settings, we used a learning rate of 1e-3, a hidden

state size of 128, and early stopping to prevent overfitting.

The goal here is to evaluate and validate our proposed model, the CGLSTM, in a social navigation environment.

6.4.9 SocNavGym Environment

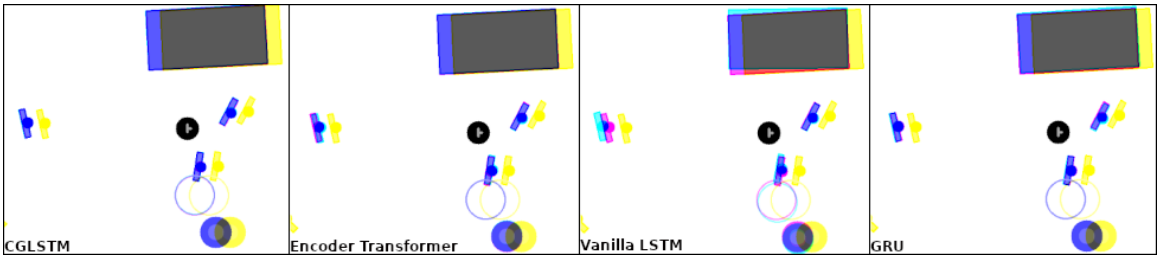


Figure 6.14: Visual comparison of predictive performance in the SocNavGym environment, highlighting the color-coded accuracy of predictions: Blue indicates an accurate prediction aligned with the actual future states, red signifies discrepancies between the model's predictions and the actual future states, and yellow represents the current state. Cyan represents the overlap between the actual future state and the current state. Dark blue highlights the overlap between the actual state and the predicted state. Grey represents the overlap among the current state, the actual future state, and the predicted future state.
Specifically, (a) the CGLSTM model demonstrates precise alignment, indicating accurate predictions. (b) The Transformer model exhibits high predictive accuracy, with some inconsistency marked by red. (c) The Vanilla LSTM model, predominantly in red, shows a high level of inaccuracy in predictions. (d) The GRU model also shows inaccurate predictions but performs slightly better than the Vanilla LSTM model.

Model	Prediction Time (s)	MAE (k=1)	MAE (k=3)	MAE (k=5)	MAE (k=10)
Vanilla LSTM	0.00137	1.65e-2	5.21e-2	9.61e-2	1.30e-1
GRU	**0.00126**	1.88e-2	5.53e-2	9.49e-2	1.35e-1
Encoder Transformer	0.02793	1.22e-2	3.73e-2	6.75e-2	1.12e-1
CGLSTM	0.00738	**8.40e-3**	**3.33e-2**	**6.41e-2**	**8.26e-2**

Table 6.9: Comparison of Prediction Time and Accuracy for Various Models in SocNavGym

In the SocNavGym environment, we evaluated the performance of various models with a particular focus on the CGLSTM model. This experiment includes analysis on inference time, training and validation losses, the number of parameters, and predictive accuracy.

Table 6.9 presents the Mean Absolute Error (MAE) for each model at different future time steps (k=1, k=3, k=5, and k=10), along with the prediction time in seconds. The CGLSTM model consistently exhibits the lowest MAE across all k-values, indicating its superior capability to predict future states with better accuracy within the SocNavGym environment.

The prediction time also provides insight into the computational efficiency of the models. While the GRU model shows the fastest prediction time, the CGLSTM model, despite having a slightly higher prediction time of about 2% when compared to GRU, achieves significantly better predictive accuracy of about 5% across all evaluated future time steps. This balance between accuracy and efficiency highlights the effectiveness of the CGLSTM model in real-time prediction tasks within dynamic environments such as SocNavGym.

We visually represent and compare the predictive performances of the various models. In the figures, the blue color indicates an accurate prediction that aligns the model's predictions with the actual future states. In contrast, the red color signifies discrepancies between the model's predictions and the actual future states, highlighting areas where the model's predictions were not accurate. The yellow color represents the current state.

In Figure 6.14, we visually compare the predictive performance of various models in the SocNavGym environment. The CGLSTM model demonstrates effective prediction accuracy, as indicated by the predominant blue color. The Encoder Transformer model also shows high predictive accuracy with blue representations, though slightly less consistently than the CGLSTM model, as evidenced by the presence of some red areas. The Vanilla LSTM and GRU models exhibit more red areas, suggesting a higher level of inaccurate predictions, with the Vanilla LSTM model displaying this more prominently. These visual representations from the SocNavGym Environment illustrate that the model architecture significantly influences predictive accuracy. The CGLSTM model stands out for effectively predicting environmental dynamics. The general trend observed across all models is that a higher frequency of blue color correlates with a model's ability to accurately predict future states, reinforcing the importance of model architectures for complex, dynamic scenarios.

Sequence length significantly impacts the performance of the CGLSTM model. Longer sequences provide more context and aid in capturing long-term dependencies but also increase computational complexity. In the adding problem with sequence length $T = 1000$, the CGLSTM outperformed other models in MAE. Similarly, in the row-wise MNIST digit recognition task with sequence length 28, the CGLSTM achieved the highest accuracy, demonstrating its robustness across varying sequence lengths. Our experiments revealed performance differences across various domains. In time series forecasting (adding prob-

lem), the CGLSTM had the lowest MAE. In image classification (row-wise MNIST, Fashion-MNIST), it achieved the highest accuracy. For NLP tasks (IMDB sentiment analysis, Penn Treebank language modeling), it showed competitive results with improved test accuracy and perplexity scores. In the SocNavGym environment (social navigation prediction), it exhibited robust performance. These results highlight the model's versatility and effectiveness across different domains, influenced by the nature of dependencies (temporal vs. contextual), as seen in Table 6.8.

Conclusion

The work presented in this chapter introduced the Cosine-Gated Long Short-Term Memory (CGLSTM) model, demonstrating its advantages in sequence modeling through reduced prediction errors and balanced computational efficiency. Evaluations across diverse tasks—including image classification, language modeling, and robotic navigation—highlighted CGLSTM's capability to capture long-term dependencies while mitigating outliers or noise.

Despite these gains, key challenges remain, particularly in integrating CGLSTM within reinforcement learning settings that require flexible prediction horizons and real-time adaptation. The next chapter builds on these insights by investigating entropy-driven adaptive mechanisms for predictive reinforcement learning. There, we explore how to adjust the prediction horizons dynamically, leveraging the strengths of CGLSTM in environments with continuous action spaces and time-sensitive decision-making. This progression aims to further bridge the gap between robust sequence modeling and the practical demands of real-world robotic and AI applications, extending the promise of CGLSTM into increasingly complex and uncertain domains.

Chapter 7

Adaptive Predictive Reinforcement Learning: Entropy-Driven Adaptive Prediction Horizons

7.1 Introduction

This chapter advances our research in improving Reinforcement Learning (RL) in complex and uncertain environments by introducing an entropy-driven adaptive horizon mechanism tailored for continuous action spaces. Building on the foundational RL concepts discussed in Chapter 4 and our predictive world models for social navigation proposed in Chapter 5, we address the key limitations of fixed prediction horizons that hinder long-term efficiency and adaptability in dynamic environments.

7.1.1 Motivation

Reinforcement Learning has demonstrated remarkable success in applications such as robotic control, autonomous navigation, and decision-making tasks has been achieved. However, real-world scenarios are often characterized by dynamic complexities and varying levels of uncertainty, which pose significant challenges, particularly in continuous action spaces. Our preliminary experiments using a fixed 8-step prediction horizon in a low-complexity envi-

ronment yields up to 30% wasted computation and slows real-time responsiveness, while a shorter 2-step prediction horizon under a more complex scenario may fail to capture the necessary long-term context. This mismatch suggests the need for an adaptive approach to horizon selection. Traditional RL frameworks often rely on fixed prediction horizons, limiting the agent's ability to adapt to changing environmental demands.

For example, in a social navigation environment, an agent navigating through unpredictable human movements may require more information about the environment to make strategic decisions. In contrast, simpler and more predictable environments benefit from shorter horizons to conserve computational resources and enable faster responsiveness. This disparity demonstrates the need for a flexible approach to horizon selection that dynamically adapts to the complexity of the environment in real time.

A key driver of this adaptability is the uncertainty of action. Entropy, a measure of this uncertainty, quantifies the unpredictability in an agent's decision-making process. High entropy signals greater uncertainty, necessitating longer horizons to improve exploration and decision-making. In contrast, low entropy indicates confidence in actions, allowing shorter horizons to optimize efficiency without sacrificing performance.

State-of-the-art RL frameworks, such as DreamerV3 [39], leverage predictive world models to predict future states, improving sample efficiency. However, these models typically use fixed horizons, restricting their ability to adapt dynamically to varying levels of uncertainty. Furthermore, while Gated Recurrent Units (GRUs) form the basis of predictive modeling in DreamerV3. To address GRU limitations, we integrate the Cosine-Gated Long Short-Term Memory (CGLSTM) network into Dreamerv3, improving sequence modeling and state prediction accuracy.

This chapter introduces a framework that combines adaptive horizon mechanisms based on entropy with advanced predictive models to overcome fixed horizon limitations. This approach gives RL agents more flexibility and efficiency, helping them perform better in dynamic environments with continuous action spaces that need adaptive decision-making.

The progression of this research is illustrated in Figure 7.1. It begins with foundational RL frameworks for discrete action spaces characterized by Fixed prediction horizons. The

introduction of the Cosine-Gated Long-Short-Term Memory (CGLSTM) network as an improvement to traditional recurrent architectures marked a significant step toward addressing temporal dependencies in more realistic tasks. Building on these advancements, this chapter focuses on proposing an entropy-driven adaptive horizon mechanism that dynamically adjusts the prediction horizon in response to real-time action uncertainty.

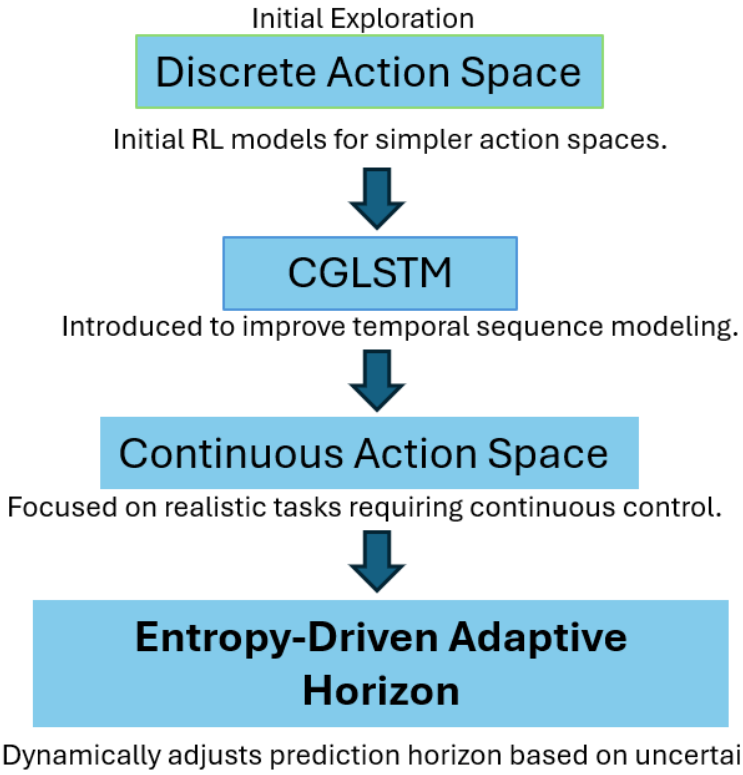


Figure 7.1: A flowchart illustrating the progression of our research from discrete action spaces (Chapter 4) and CGLSTM (Chapter 5) to the current focus on entropy-driven adaptive horizons in continuous action spaces.

7.1.2 Research Objectives and Contributions

This chapter aims to develop and evaluate our entropy-driven adaptive prediction horizon mechanism within the Reinforcement Learning (RL) framework, specifically targeting continuous action spaces. The key contributions are as follows:

1. **Proposed Entropy-Driven Adaptive Prediction Horizon Framework and Integration of CGLSTM into DreamerV3:**

    - **Entropy-Driven Adaptive Prediction Horizon Framework:** We introduce a novel uncertainty-aware method that dynamically adjusts prediction horizons,

addressing challenges in dynamic and continuous action space RL environments.

- **Integration of CGLSTM into DreamerV3:** We demonstrate the impact of replacing Gated Recurrent Unit (GRU) cells with Cosine-Gated Long Short-Term Memory (CGLSTM) cells in DreamerV3, leading to improved sequence modeling, state prediction accuracy, and policy performance.

2. **Evaluation and Benchmarking Across Diverse Environments and Algorithms:**

   - **Evaluation Across Environments:** We validate the proposed framework in `LiteSocNavGym` [58], a lite complex social navigation environment, and `LunarLander-v2`, a continuous control environment serving as a baseline.

   - **Benchmarking Against State-of-the-Art Algorithms:** We compare our approach with established RL algorithms, including Vanilla SAC, PPO, DDPG, and DreamerV3 variants, demonstrating the superiority of entropy-driven adaptive Prediction horizons and improved predictive modeling.

7.1.3 Significance of the Research

This research addresses the critical limitations of fixed prediction horizons in RL by introducing an entropy-driven adaptive mechanism. By dynamically balancing computational efficiency and prediction horizon, this method enhances the adaptability of RL agents in the face of real-time uncertainty. The integration of CGLSTM further improves predictive models, enabling agents to achieve better accuracy in predicting future states. Together, these advancements represent a significant step forward in developing robust and efficient RL systems capable of excelling in complex, continuous action spaces. This work lays the foundation for broader real-world applications, including robotics and autonomous systems.

7.2 Related Work

As discussed in Chapter 4, Reinforcement Learning (RL) provides a framework for agents to learn optimal policies through interactions with an environment, aiming to maximise long-term rewards. While early successes were often demonstrated in discrete action spaces, many

real-world tasks naturally involve continuous action spaces, posing additional challenges in terms of exploration, sample efficiency, and adaptability. Continuous action spaces present a larger and often infinite set of possible actions, making it more difficult for agents to explore effectively and requiring more data to converge on optimal policies. Additionally, policies must generalise across a wide range of actions, increasing the risk of poor sample efficiency. Adapting to dynamic environments further complicates the process, as subtle variations in action selection can lead to significant differences in outcomes. Chapter 5 explored predictive world models for social navigation scenarios, highlighting the value of latent representations in improving decision-making. Building on these foundations, this section reviews key literature on RL in continuous action spaces, prediction horizons in model-based RL, entropy as a measure of uncertainty, and the use of advanced recurrent units like Cosine-Gated Long Short-Term Memory (CGLSTM) for enhanced sequence modeling.

7.2.1 Reinforcement Learning in Continuous Action Spaces

Reinforcement Learning in continuous action spaces extends RL's applicability to domains requiring fine-grained control, such as robotics and autonomous vehicles [2]. Several state-of-the-art algorithms have emerged to address these challenges, each with distinct strengths and limitations.

Soft Actor-Critic (SAC) [34] is a model-free algorithm that leverages entropy-based objectives to encourage stochastic policies, improving exploration in continuous spaces. However, SAC's performance often relies on large datasets and struggles with non-stationary environments. Proximal Policy Optimisation (PPO) [109] introduces a clipped objective to stabilize policy updates, achieving robust performance across various tasks, although its simplicity may limit efficiency in complex environment. Twin Delayed Deep Deterministic Policy Gradients (TD3) [26] mitigates overestimation bias through a second critic network and target smoothing but shares Deep Deterministic Policy Gradients' (DDPG) [69] sensitivity to hyperparameters and extensive data requirements.

These algorithms significantly advance RL in continuous action spaces but often rely on static exploration strategies and fixed prediction horizons, limiting their adaptability in dynamic, uncertain environments.

7.2.2 Prediction Horizons in Model-Based RL

Model-based RL methods improve sample efficiency by leveraging environment models to predict future states and rewards [19, 40]. However, these methods often use fixed prediction horizons, assuming consistent task complexity and environmental predictability. This limitation may hinder adaptability in real-world scenarios, where task complexity and uncertainty are highly variable.

DreamerV3 [40] uses fixed horizon lengths to plan future trajectories. While effective in many cases, this approach might be computationally intensive in simple environment or maybe inefficient in complex ones, leading to suboptimal performance. Adaptive prediction horizons provide an alternative by dynamically adjusting the prediction horizon to align with the complexity and uncertainty of the task.

7.2.3 Entropy as a Measure of Uncertainty

Entropy is a well-established metric for quantifying action uncertainty in RL. Methods like SAC [34] integrate entropy into the policy objective, encouraging exploration and preventing premature convergence to suboptimal deterministic strategies. The entropy $E(\pi(a|s))$ of an action distribution $\pi(a|s)$ is mathematically defined as:

$$H(\pi(a|s)) = -\int \pi(a|s) \log \pi(a|s)\, da \tag{7.1}$$

While entropy has primarily guided exploration in RL, its application to dynamically adjust prediction horizons based on action uncertainty represents a novel approach. High entropy signals uncertainty, suggesting a need for longer prediction horizons to improve planning and better anticipate future states. Low entropy indicates confident predictions, enabling shorter prediction horizons to conserve computational resources without sacrificing performance.

By leveraging entropy to drive adaptive prediction horizons, this approach enhances the agent's ability to balance exploration and efficiency in continuous action space environments, addressing the limitation of fixed-horizon RL frameworks.

7.2.4 Recurrent Units in Reinforcement Learning

Sequential decision-making tasks often require memory mechanisms to manage temporal dependencies and partial observability. Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks [48] and Gated Recurrent Units (GRUs) [20], are widely used in RL pipelines to capture these patterns.

GRUs are computationally efficient but may struggle with long-term dependencies, particularly in high-dimensional tasks. The Cosine-Gated Long Short-Term Memory (CGLSTM) network, introduced in Chapter 6, addresses these limitations by integrating cosine similarity-based gating mechanisms. This enhancement improves long-term sequence modeling making CGLSTM more effective than GRUs in dynamic environments.

The limitations of fixed prediction horizons and fixed exploration strategies in RL highlight the need for adaptive mechanisms in dynamic environments. Model-free algorithms like SAC, PPO, and TD3 excel in performance but rely on short-term value estimates, limiting their ability to plan far into the future. In contrast, model-based methods like DreamerV3 improve efficiency by using predictive world models but are constrained by fixed prediction horizons during trajectory planning.

Entropy has proven effective in guiding exploration but remains underutilised for horizon adaptation. Similarly, while RNNs, LSTMs, and GRUs enhance sequence modeling, CGLSTM offers superior predictive accuracy. The integration of entropy-driven adaptive horizons with CGLSTM-enhanced models addresses these gaps, providing a novel framework for flexible, efficient, and robust RL in complex continuous action spaces.

7.3 Methodology

This chapter builds on the foundational concepts introduced in Chapter 4 and the improvement in predictive modeling from Chapter 5. The primary contributions involve integrating an entropy-driven adaptive prediction horizon mechanism into a Soft Actor-Critic (SAC) framework and using predictive fixed horizons as one of our baselines. Using advanced predictive models and dynamically adjusting horizons based on policy entropy, the agent gains improved long-term planning capabilities and adaptability in continuous action spaces.

These methods are evaluated against established reinforcement learning baselines, including vanilla SAC, PPO, DDPG, and variants of DreamerV3, to highlight their comparative advantages.

7.3.1 Integrating CGLSTM into Soft Actor-Critic (SAC)

Chapter 6 demonstrated that the Cosine-Gated Long Short-Term Memory (CGLSTM) network can provide richer temporal representations and more accurate predictions of future state than traditional recurrent units such as GRUs. Although these benefits were initially shown in a discrete action space, the present work extends the use of CGLSTM to continuous action reinforcement learning tasks, which are critical for real-world domains such as robotic manipulation and complex control systems.

The Soft Actor-Critic (SAC) algorithm [34] is used as the base RL method due to its stability and entropy-based policy regularization. Although SAC typically operates in a model-free setting, integrating a CGLSTM-based predictive model allows the agent to anticipate long-term consequences of its actions over a prediction horizon. The model is a combination of both model-free and model-based approaches, leveraging the CGLSTM-based predictive model.

Figure 7.2 illustrates the integration of CGLSTM into the SAC framework. At each timestep, the environment provides the current state $\mathbf{s}_t$. Along with prior states and actions are encoded and fed into the CGLSTM-based predictive model, which outputs a single trajectory of future states

$$\{\mathbf{s}_{t+1}, \mathbf{s}_{t+2}, \ldots, \mathbf{s}_{t+H}\},$$

where H is the prediction horizon. Given a sequence of previous states $\{\mathbf{s}_{t-k}, \ldots, \mathbf{s}_t\}$, actions $\{\mathbf{a}_{t-k}, \ldots, \mathbf{a}_{t-1}\}$, and done flags $\{\mathbf{d}_{t-k}, \ldots, \mathbf{d}_{t-1}\}$, the CGLSTM predicts:

$$\mathbf{s}_{t+h} \;=\; \mathrm{CGLSTM}\big(\mathbf{s}_t, \mathbf{a}_{t-1}, \mathbf{d}_{t-1}\big), \quad h = 1, \ldots, H.$$

These predicted future states are combined with the current state to input $\hat{\mathbf{s}}_t$. In other words, $\hat{\mathbf{s}}_t$ consists of the original state $\mathbf{s}_t$ (along with previous states) augmented by the future states predicted $\{\mathbf{s}_{t+1}, \ldots, \mathbf{s}_{t+H}\}$. This combined states are then fed into the SAC

actor-critic networks, providing them with insights into how the environment may evolve.

The SAC policy is optimized using an entropy-regularized objective:

$$J_\pi = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\rho_\pi}\Big[\, Q(\mathbf{s},\mathbf{a}) \;-\; \alpha \;\log\; \pi(\mathbf{a} \mid \mathbf{s})\Big],$$

where $\pi(\mathbf{a} \mid \mathbf{s})$ is the policy, $Q(\mathbf{s},\mathbf{a})$ is the critic's Q-value function, and α is the entropy coefficient. By providing this enhanced observation set via future-state predictions, the Q-value estimations and policy updates gain more accurate knowledge of possible outcomes, thereby improving long-term decision-making in continuous action spaces.

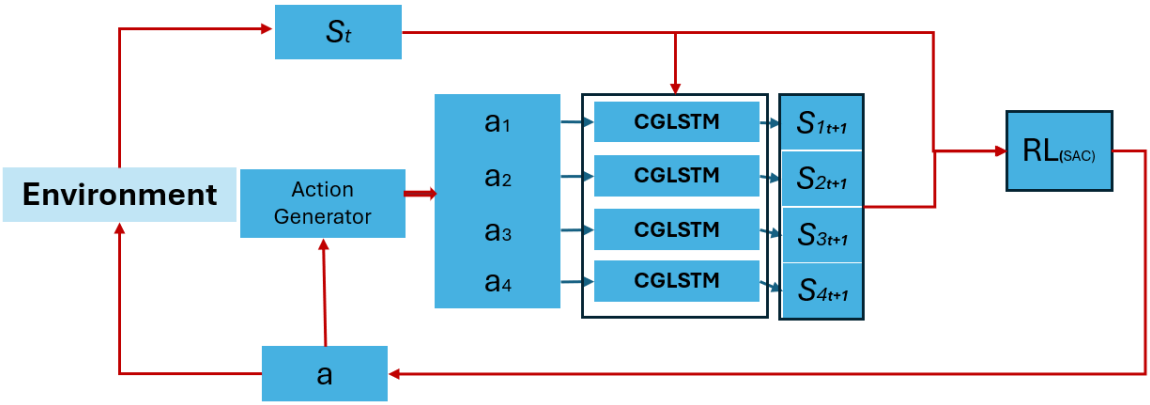


Figure 7.2: Integration of the CGLSTM model into the SAC algorithm. In this framework, four additional states are generated to enhance the SAC agent's learning. This is achieved by generating four new actions by adding slight noise to the current action. The CGLSTM takes the current state and combines it with each of the four actions (a_1, a_2, a_3, a_4) to predict the corresponding next states ($S_{1_{t+1}}$, $S_{2_{t+1}}$, $S_{3_{t+1}}$, $S_{4_{t+1}}$). These predicted states are then used as input for action selection and value estimation, providing a richer representation of future possibilities. This improves the agent's decision-making process by considering multiple potential future trajectories.

7.3.2 Entropy-Based Adaptive Horizon Selection

Traditional model-based RL methods often use a fixed prediction horizon, limiting their adaptability in dynamic and uncertain environments. To address this limitation, an entropy-driven adaptive prediction horizon method is introduced. The agent's action distribution entropy $E(\pi(a|s))$ quantifies uncertainty: higher entropy indicates greater uncertainty, suggesting a need for longer horizons to mitigate risk, while lower entropy signifies confidence and thus warrants shorter horizons to conserve computational resources.

A set of candidate prediction horizons $\{1, 2, 4, 6, 8\}$ is defined. Entropy values E in the range $[0, 2.5]$ are mapped to these horizons as follows:

$$h(H) = \begin{cases} 1 & 0 \leq E < 0.5 \\ 2 & 0.5 \leq E < 1.0 \\ 4 & 1.0 \leq E < 1.5 \\ 6 & 1.5 \leq E < 2.0 \\ 8 & 2.0 \leq E < 2.5 \end{cases} \tag{7.2}$$

Table 7.1 outlines the entropy-to-horizon mapping. The adaptive horizon selection process, detailed in Algorithm 1, demonstrates how entropy guides the prediction horizon range. As entropy is recalculated at each timestep, the agent continuously aligns its future prediction with the current level of uncertainty, ensuring a more efficient and context-dependent approach to decision-making.

Entropy Range	Prediction Horizon
$0 \leq E < 0.5$	1
$0.5 \leq E < 1.0$	2
$1.0 \leq E < 1.5$	4
$1.5 \leq E < 2.0$	6
$2.0 \leq E < 2.5$	8

Table 7.1: Mapping of entropy ranges to corresponding prediction horizons.

Algorithm 1 Adaptive Horizon Selection Algorithm

for each timestep **do**
  $E \leftarrow$ calculate_entropy(π_θ)
  $T_{\text{adaptive}} \leftarrow h(E)$
  for each action $a_i \in \{a_1, a_2, a_3, a_4\}$ **do**
    $\hat{s}_{t+1}^{(a_i)} \leftarrow f(s_t, a_i)$
    for $k = 2$ to T_{adaptive} **do**
      $\hat{s}_{t+k}^{(a_i)} \leftarrow f(\hat{s}_{t+k-1}^{(a_i)}, a_i)$
    end for
  end for
  `action` $\leftarrow$ `select_action`($\{\hat{s}_{t+T_{\text{adaptive}}}^{(a_i)}\}$)
  `execute(action)`
end for

7.3.3 Proposed Framework for Adaptive Prediction Horizons (SAC + CGLSTM + Entropy)

The core contribution of this work is the integration of an entropy-driven adaptive prediction horizon method within the Soft Actor-Critic (SAC) framework, enhanced by the Cosine-Gated Long Short-Term Memory (CGLSTM) network. This integration enables the RL agent to dynamically adjust its prediction horizon T based on real-time entropy, improving the model's performance while maintaining computational efficiency.

At each timestep t, the agent observes the current state S_t, calculates the entropy of the policy $E(\pi)$, and uses this entropy value to determine the prediction horizon T. The entropy, reflecting the uncertainty in the agent's action selection, is computed as:

$$E(\pi) = -\sum_{a \in A} \pi(a|S_t) \log \pi(a|S_t),$$

where $\pi(a|S_t)$ represents the probability of selecting action a given the current state S_t, and A denotes the action space.

The entropy value $E(\pi)$ is normalised to the range $[0, 1]$ and mapped to a discrete set of prediction steps $\{1, 2, 4, 6, 8\}$ as follows:

$$T = h(E(\pi)),$$

where the mapping $h(E)$ is defined in Equation 7.2 and Table 7.1.

This mapping ensures that higher entropy values correspond to longer prediction horizons T, enabling the agent to predict further into the future when uncertainty is high. Conversely, low entropy values lead to shorter prediction horizons, conserving computational resources in confident scenarios.

Figure 7.3 provides a schematic representation of our proposed framework. Entropy $E(\pi)$ dynamically determines the prediction horizon T, which guides the CGLSTM's multi-step state predictions. These predictions inform the SAC actor-critic network, enhancing

prediction efficiency and decision-making in continuous action spaces.

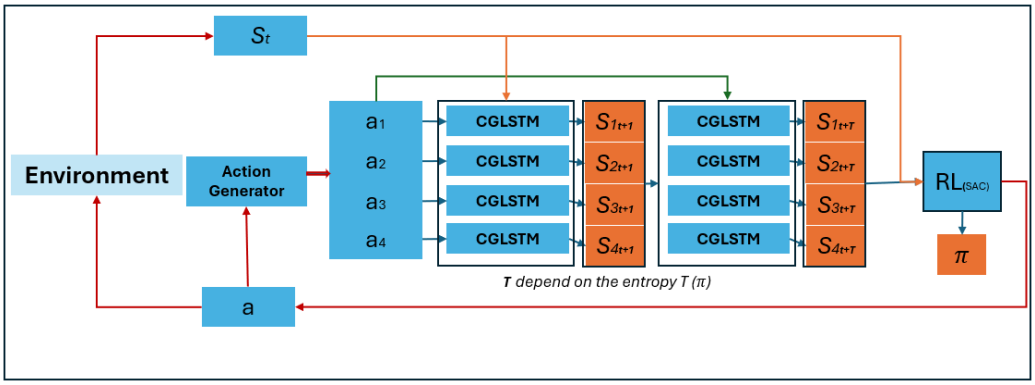


Figure 7.3: A schematic representation of the SAC framework integrated with a CGLSTM-based predictive model and the entropy-driven adaptive horizon mechanism. Entropy $E(\pi)$ dynamically determines the prediction horizon T, guiding the CGLSTM's multi-step state predictions. These predicted states inform the SAC actor-critic network for action selection and evaluation, improving prediction efficiency and decision-making in continuous action spaces.

7.3.4 DreamerV3 Architecture Comparison

Although the primary focus is on the SAC-based framework, we also integrated our CGLSTM into the state-of-the-art model DreamerV3 [40]. DreamerV3 relies on a Recurrent State Space Model (RSSM) for latent state and reward prediction. Its original LayerNormGRU-Cell can be replaced with a LayerNormCGLSTMCell to better handle long-term dependencies, especially in highly dynamic environments.

Figure 7.4 compares the vanilla RSSM (with LayerNormGRUCell) and the improved version using LayerNormCGLSTMCell. By leveraging cosine similarity-based gating, the CGLSTM cell refines data processing capabilities and improves predictive accuracy over longer horizons. This enhancement complements the integration of CGLSTM and entropy-driven adaptive horizons in SAC, demonstrating the importance of advanced sequence-modeling techniques and adaptive prediction horizons for improved RL performance in complex, continuous domains.

7.4 Experimental Setup

In this section, we begin with preliminary experiments aimed at optimising key hyper-parameters and selecting appropriate horizon configurations. Following this, we conduct

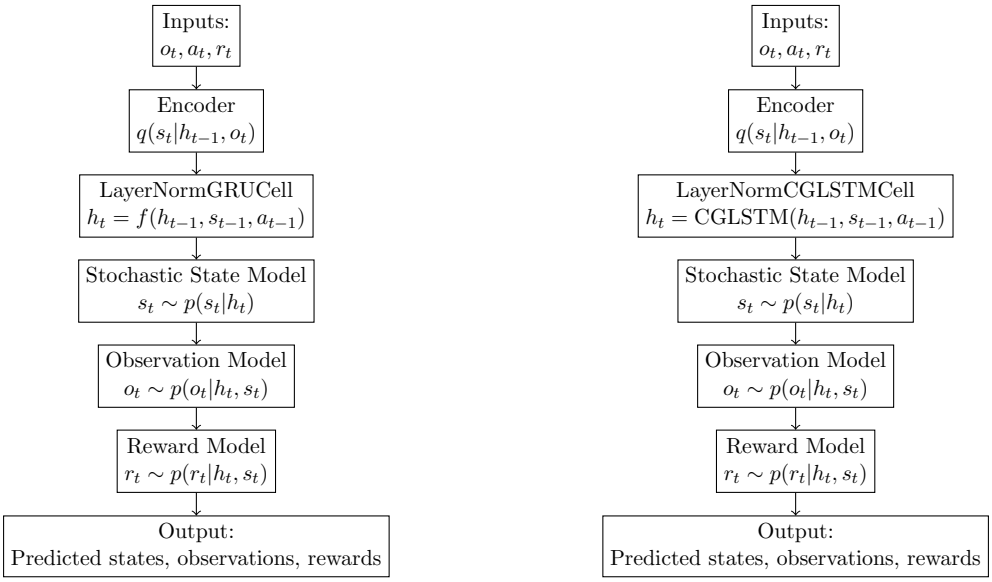


Figure 7.4: Comparison of the vanilla DreamerV3 architecture using a LayerNormGRUCell (left) and the enhanced version using a LayerNormCGLSTMCell (right). Replacing the GRU with a CGLSTM improves the model's ability to handle long-term dependencies, resulting in more accurate predictions and robust policy learning.

comprehensive evaluations across diverse environments to assess the performance and efficiency of our approach in comparison to established reinforcement learning algorithms.

7.4.1 Preliminary Experiments

To evaluate our entropy-driven adaptive prediction horizon approach, we performed preliminary tests in the LunarLander environment. These experiments addressed two design factors: first, determining the optimal window-slide length for our Cosine-Gated LSTM (CGLSTM) model from the set {16, 32, 64}, and second, comparing two adaptive prediction horizon sets—$\{1, 2, 3, 4, 5\}$ and $\{1, 2, 4, 6, 8\}$—before comparing the best-performing set against a fixed prediction horizon of 8 steps into the future.

Window-Slide Selection for CGLSTM

We began by investigating how the CGLSTM model's performance changes when the temporal window slide is set to 16, 32, or 64. Figures 7.5 and 7.6 illustrate training and validation losses, respectively, with an early stopping mechanism. For the training loss, the window slide of 64 had the lowest loss, followed by the window slide of 32. In terms of validation loss, the window slide of 64 also performed the best, followed by the window slide of 16. While the window slide of 64 outperformed the others, it came at a higher computational

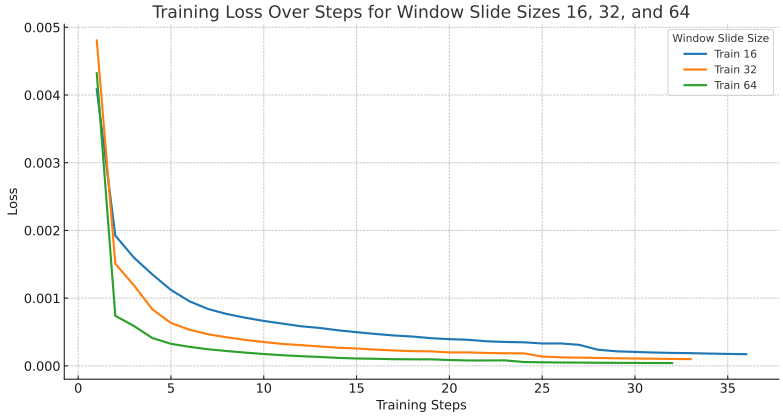


Figure 7.5: Training loss over steps for CGLSTM window slides of 16, 32, and 64 in LunarLander. A window of 16 (blue) initially drops quickly but remains higher overall, whereas window 64 (green) obtains the lowest final loss at a higher computational cost.

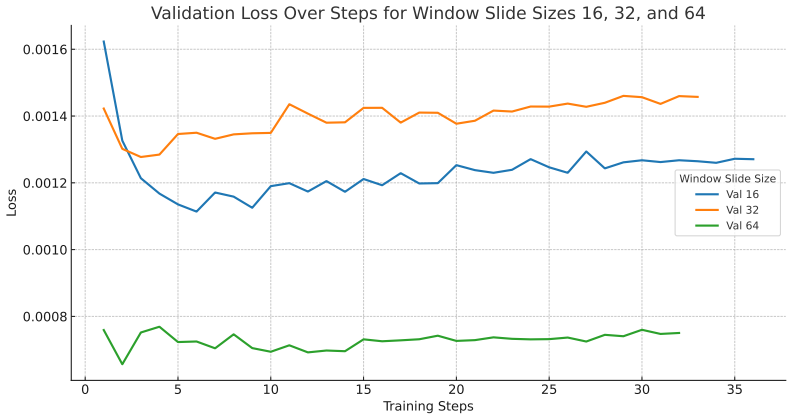


Figure 7.6: Validation loss over steps for CGLSTM window slides of 16, 32, and 64.

cost and required more memory. In contrast, a 16-window slide may not effectively capture the full dynamics in complex environments, which is a critical consideration when selecting a window length. Based on these observations, we chose a window slide size of 32, which offers a good trade-off between predictive performance and computational overhead.

Selecting Adaptive Prediction Horizon vs. Fixed Horizon

After selecting a window slide of 32 for our CGLSTM, we proceeded to evaluate different prediction horizons. We tested two candidate sets, namely $\{1, 2, 3, 4, 5\}$ (labeled "Adaptive-Set A") and $\{1, 2, 4, 6, 8\}$ (labeled "Adaptive-Set B"), using the action entropy-based horizon method described in Section 7.3.2. Each variant was trained for 500k steps in the LunarLan-

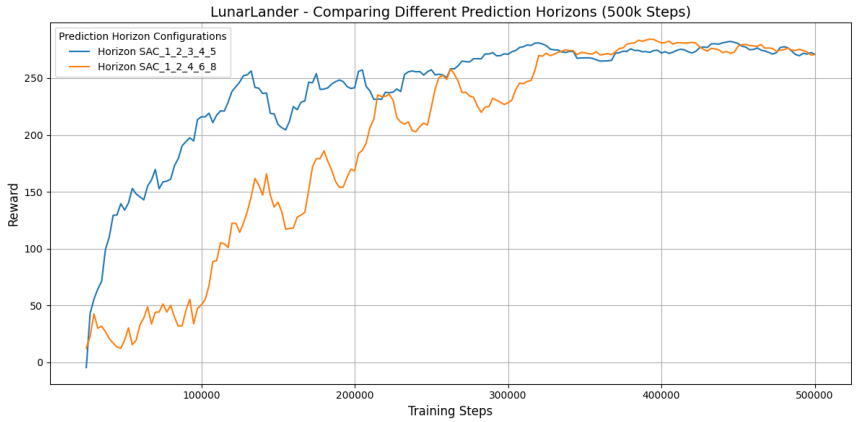


Figure 7.7: LunarLander reward curves (500k steps) comparing two adaptive prediction horizon sets: $\{1, 2, 3, 4, 5\}$ (blue) vs. $\{1, 2, 4, 6, 8\}$ (orange).

der environment, with the reward curves shown in Figure 7.7. The results indicate that Set A converges faster but stabilizes around 240–250 rewards, whereas Set B starts more conservatively yet eventually reaches a higher final performance near 280 rewards.

We selected $\{1, 2, 4, 6, 8\}$ (Set B) for our adaptive prediction horizon based on the convergence of the maximum reward with higher average returns. Next, we compared this adaptive prediction horizon against a fixed horizon of 8 steps. Figure 7.8 shows the reward and wall-clock time plot, illustrating both reward progression and the time taken to reach 500k training steps. The adaptive approach completed training in approximately 53.48 hours, whereas the fixed horizon setup took roughly 59.54 hours, saving nearly six hours of training time.

From these preliminary experiments, we concluded that a window slide of 32 strikes the best balance between training cost and accuracy for CGLSTM. Additionally, the adaptive prediction horizon $\{1, 2, 4, 6, 8\}$ outperformed $\{1, 2, 3, 4, 5\}$ in final performance and proved more efficient than the fixed 8-step horizon, saving nearly six hours of training time. These choices form the basis for our subsequent large-scale experiments in LiteSocNavGym.

To evaluate the effectiveness of our proposed entropy-driven adaptive prediction horizon method, we compare its performance against a range of well-established reinforcement learning algorithms. The models included in this comparison span both model-free and model-based paradigms.

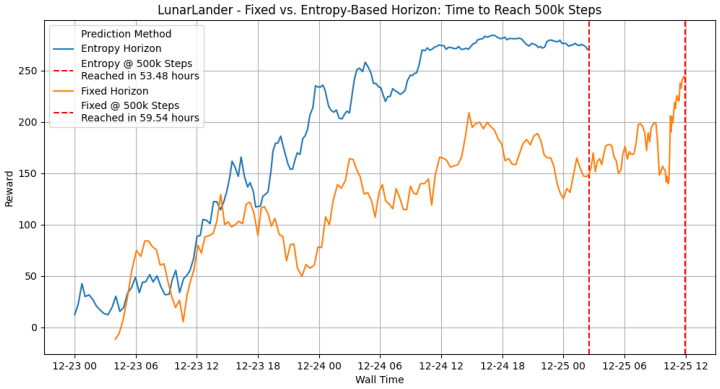


Figure 7.8: Comparison of the fixed prediction horizon (8 steps into the future) (orange) with the adaptive prediction horizon $\{1, 2, 4, 6, 8\}$ (blue). The reward curves are plotted against wall time, with red-dashed vertical lines marking the completion of 500k steps for each method. The adaptive prediction horizon reached 500k steps in 53.48 hours, whereas the fixed horizon required 59.54 hours.

Model-free algorithms include SAC, PPO, TD3, and DDPG, known for their robustness in handling continuous action spaces and in balancing exploration with exploitation. For model-based approaches, we compared with DreamerV3, and our variant in which the GRU is replaced with CGLSTM to improve temporal modeling.

A summary of the models, including their algorithm type, prediction horizon method, and recurrent unit used, is provided in Table 7.2.

Model	Algorithm	Prediction Horizon Method	Recurrent Unit
SAC	Model-Free	N/A	N/A
PPO	Model-Free	N/A	N/A
TD3	Model-Free	N/A	N/A
DDPG	Model-Free	N/A	N/A
DreamerV3	Model-Based	Fixed	GRU
DreamerV3 + CGLSTM	Model-Based	Fixed	CGLSTM
SAC + CGLSTM (Fixed Horizon)	Hybrid	Fixed	CGLSTM
Proposed: SAC + CGLSTM (Adaptive Horizon)	Hybrid	Adaptive	CGLSTM

Table 7.2: Summary of models being compared, including algorithm type, horizon strategy, and recurrent unit used.

7.4.2 Environments

To thoroughly assess the effectiveness of our proposed entropy-driven adaptive prediction horizon method, we conducted experiments in two distinct environments: **LunarLander-v2** and our custom-built **LiteSocNavGym**. These environments were selected to evaluate the models' performance across both standard continuous control tasks and complex social

navigation environments.

LunarLander-v2

LunarLander-v2 is a widely recognized benchmark environment provided by OpenAI Gym [12], commonly used to evaluate reinforcement learning algorithms in continuous action spaces. In this environment, the agent controls a lunar lander with the objective of achieving a stable landing on a designated landing pad. The state space consists of the lander's position, velocity, angle, and angular velocity, while the action space consists of continuous thrusts and rotations. The reward function is designed to incentivize smooth landings, penalise crashes, and reward successful landings.

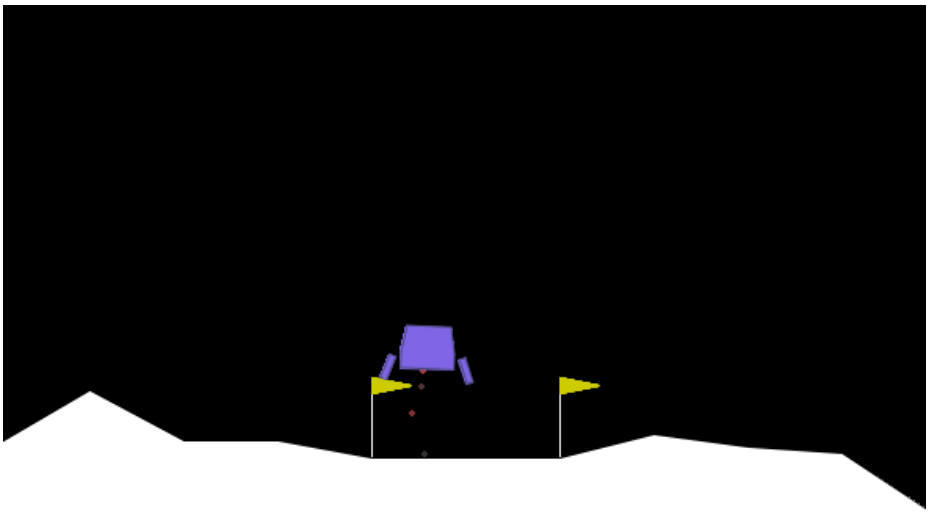


Figure 7.9: A snapshot of the LunarLander-v2 environment, where the agent controls a lander aiming to achieve a stable touchdown on the landing pad.

The environment can operate under discrete or continuous action spaces, allowing for nuanced control over the lander's movements. This setup challenges the agent to balance thrust and rotation effectively to navigate the lander towards the goal while minimising fuel consumption and avoiding obstacles. The success of the agent is measured by its ability to consistently land within the designated area, reflected in the Success Rate (%) and Average Return metrics.

LiteSocNavGym

LiteSocNavGym [88] is a custom-designed environment developed to simulate social navigation scenarios where a robot must navigate towards a goal while avoiding dynamic obstacles such as humans and static objects such as tables. We developed this environment because, at the time of experimentation, SocNavGym was developed using Gym, while LiteSocNavGym was developed using Gymnasium and DreamerV3 only supports Gymnasium. This environment introduces a higher level of complexity compared to standard benchmarks, as it requires the agent to manage dynamic interactions and maintain social comfort simultaneously.

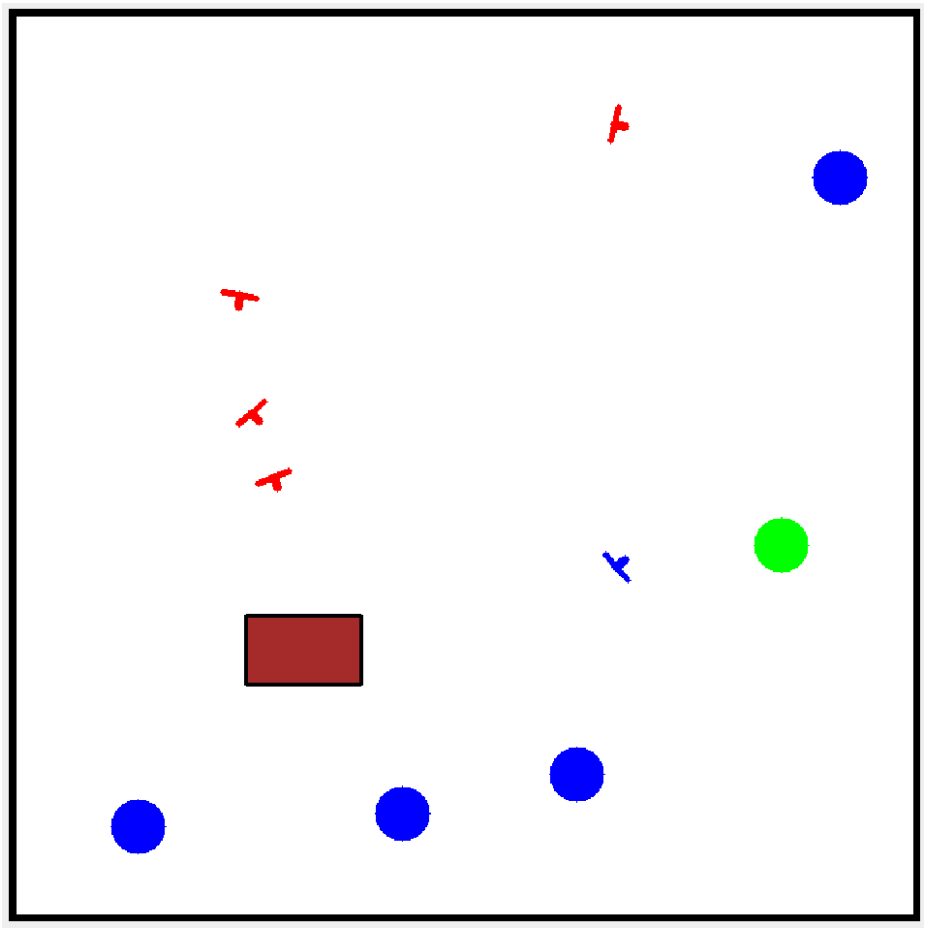


Figure 7.10: A snapshot of the LiteSocNavGym environment, illustrating the robot navigating towards its goal. The robot is shown in blue, humans in red, tables in brown, and robot and human goals in green and blue respectively.

In LiteSocNavGym, the number of humans varies between 1 to 4 per episode, introducing dynamic obstacles that move towards their individual goals. Each human agent

navigates independently, creating a dynamic and unpredictable environment for the robot. Additionally, we included a table that serves as a static obstacle that the robot must navigate around. The environment uses a continuous action space that the agent must use to move effectively.

The reward function in LiteSocNavGym is multifaceted, designed to encourage not only the successful achievement of the navigation task but also the maintenance of social norms and efficiency. Specifically, the agent receives a goal Reward of +1.0 for successfully reaching the goal, while penalties are awarded for moving out of map boundaries (Out-of-Map Reward: -0.5), colliding with humans or tables (Collision Reward: -0.5), and exceeding the maximum number of allowed steps (Max Ticks Reward: -0.3). Additionally, an Alive Reward of -0.003 per step incentivizes efficiency, and a Discomfort Penalty is applied when the robot maintains too close a distance to humans, further promoting socially compliant navigation behavior.

The implementation of LiteSocNavGym leverages the Gymnasium framework, allowing for customizable configurations such as map size, number of humans, and various reward coefficients. The observation space is carefully structured to include the robot's orientation and position, goal position, positions and orientations of humans, and positions and sizes of tables. This observation facilitates informed decision-making by the agent, enabling it to anticipate and react to dynamic changes within the environment.

Figures 7.9 and 7.10 provide visual snapshots of the LunarLander-v2 and LiteSocNavGym environments, respectively. These illustrations offer a clear representation of the challenges each environment presents, highlighting the differences in dynamics and objectives that the models must navigate.

7.4.3 Hyperparameters

The hyperparameter settings applied across all models were adjusted to ensure optimal performance for each algorithm. Common hyperparameters such as learning rate, batch size, hidden units, and entropy coefficients were chosen based on standard reinforcement learning practices and validated through preliminary experiments.

All models were trained for 5 million steps on the LiteSocNavGym environment and 500 thousand steps on the LunarLander environment. We used a batch size of 64 and a hidden state size of 256 for all models. A seed value of 42 was consistently used during both training and testing to ensure that our experiments are replicable.

To account for variability in reinforcement learning training, each model was trained and evaluated across five different random seeds. The reported metrics represent the mean across these runs, providing a more robust assessment of each model's performance and mitigating the effects of outliers in the training process.

7.5 Evaluation Metrics

To evaluate the models, different metrics were used, focusing on three primary aspects: task success, efficiency, and computational cost. These metrics provide a holistic view of each model's capabilities and performance across diverse reinforcement learning tasks.

Overview of Metrics

The following metrics were utilised to assess and compare model performances:

Success Rate (%): Measures the proportion of episodes where the agent successfully completes its task.

Average Return: Evaluates the cumulative reward accumulated by the agent over episodes.

Average Successful Path Length: Calculates the average number of steps taken in successful episodes, reflecting task efficiency.

Weighted Success/Efficiency: A composite metric that balances success rate with task efficiency.

Average Inference Time (ms): Assesses the average time taken by the agent to make an inference per step.

Average Simulation Time (ms): Measures the average time required for simulation

per step.

Computational Efficiency (hours): Estimates the total time used during evaluation, expressed in hours.

Detailed Metric Definitions

Success Rate (%)

$$\text{Success Rate (\%)} = \left(\frac{\text{Number of Successful Episodes}}{\text{Total Episodes}} \right) \times 100 \tag{7.3}$$

This metric quantifies the agent's ability to consistently achieve the desired task outcomes.

Average Return

$$\text{Average Return} = \frac{\sum \text{Episode Returns}}{\text{Total Episodes}} \tag{7.4}$$

This reflects how effectively the agent maximizes rewards.

Average Successful Path Length

$$\text{Average Successful Path Length} = \frac{\text{Total Steps in Successful Episodes}}{\text{Number of Successful Episodes}} \tag{7.5}$$

A lower path length signifies more efficient task execution.

Weighted Success/Efficiency

$$\text{Weighted Success/Efficiency} = \alpha \cdot \text{Success Rate (\%)} + \beta \cdot \left(\frac{1}{\text{Average Successful Path Length}} \right) \tag{7.6}$$

where $\alpha = 0.7$ and $\beta = 0.3$.

This metric balances success rate and efficiency.

Average Inference Time (ms)

$$\text{Average Inference Time (ms)} = \frac{\text{Total Inference Time}}{\text{Total Steps}} \tag{7.7}$$

Reflects the computational efficiency of the agent's decision-making process.

Average Simulation Time (ms)

$$\text{Average Simulation Time (ms)} = \frac{\text{Total Simulation Time}}{\text{Total Steps}} \tag{7.8}$$

Evaluates the efficiency of the simulation environment itself.

Computational Efficiency (hours)

$$\text{Computational Efficiency (hours)} = \frac{\text{Total Evaluation Time (seconds)}}{3600} \tag{7.9}$$

Provides an estimate of total computational resources consumed.

7.6 Results and Discussion

7.6.1 Training Performance

The training performance of the models was evaluated in the LiteSocNavGym and LunarLander v2 environments, focusing on learning stability, convergence rates, and overall training behavior. The experiment provides insights into the effectiveness of our proposed entropy-driven adaptive prediction horizon method compared to the fixed-horizon and traditional reinforcement learning models.

Figures 7.11 and 7.12 illustrate the cumulative training returns over steps for each model in LiteSocNavGym and LunarLander v2, respectively. In **LiteSocNavGym**, DreamerV3 and DreamerV3 + CGLSTM converge faster than other approaches, showing stable learning. Our proposed method (SAC + CGLSTM with Adaptive Prediction Horizon) had the highest reward achieved, outperforming SAC, PPO, DDPG, TD3, DreamerV3, and DreamerV3 +

CGLSTM overall.

In LunarLander v2, Our proposed method (SAC + CGLSTM with adaptive prediction Horizon) converged faster than other approaches, showing a stable learning followed by the Fixed prediction Horizon , followed by the Dreamerv3.

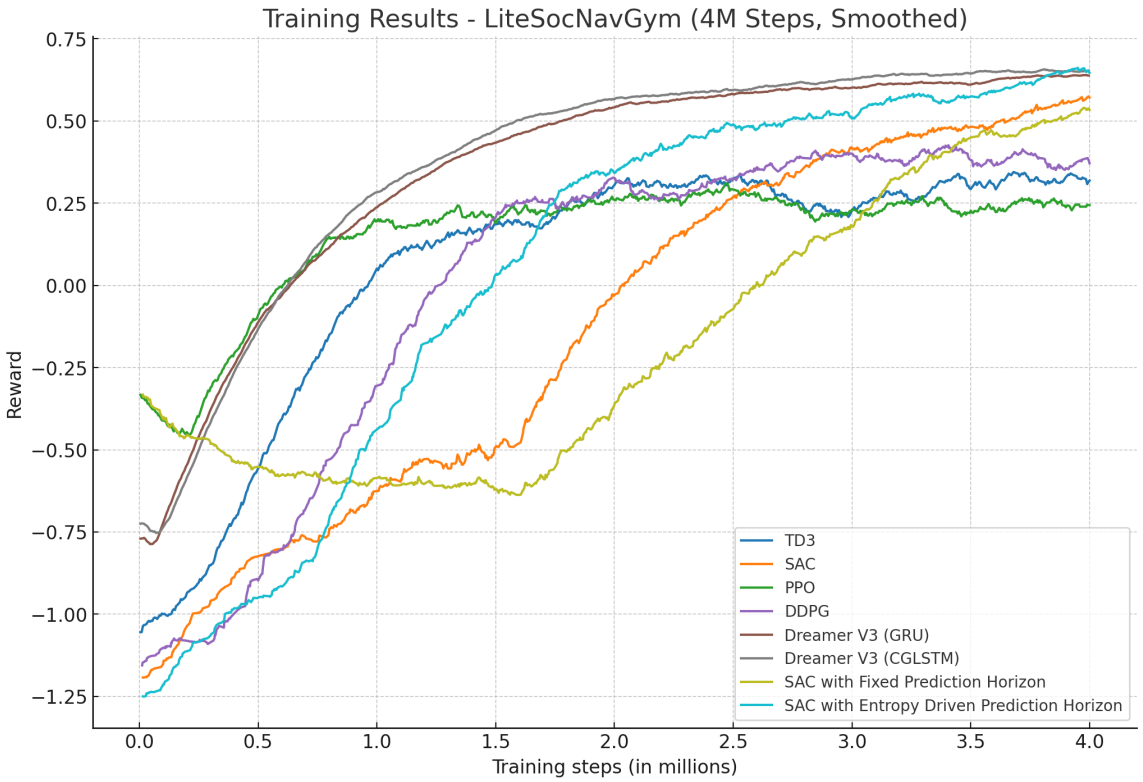


Figure 7.11: Cumulative return over training steps for each model in the LiteSocNavGym environment. DreamerV3 and DreamerV3 + CGLSTM demonstrate faster convergence, followed by the adaptive horizon method, while PPO, SAC, DDPG, and TD3 converge more slowly.

Overall, DreamerV3 variants excel in both environments due to advanced predictive modeling, while the SAC approach incorporating an entropy-driven horizon stabilizes training and outperforms the fixed-horizon version. In LiteSocNavGym, the complexity of social interaction benefits more from DreamerV3's learned world model, and in LunarLander v2 (with mild test-time observation noise), DreamerV3 + CGLSTM remains the top performer, followed by DreamerV3 and the adaptive prediction horizon SAC. Traditional RL baselines—PPO, DDPG, TD3—show slower convergence and lower final returns, showing the advantage of flexible horizon selection and improved sequence modeling in complex or noisy tasks.

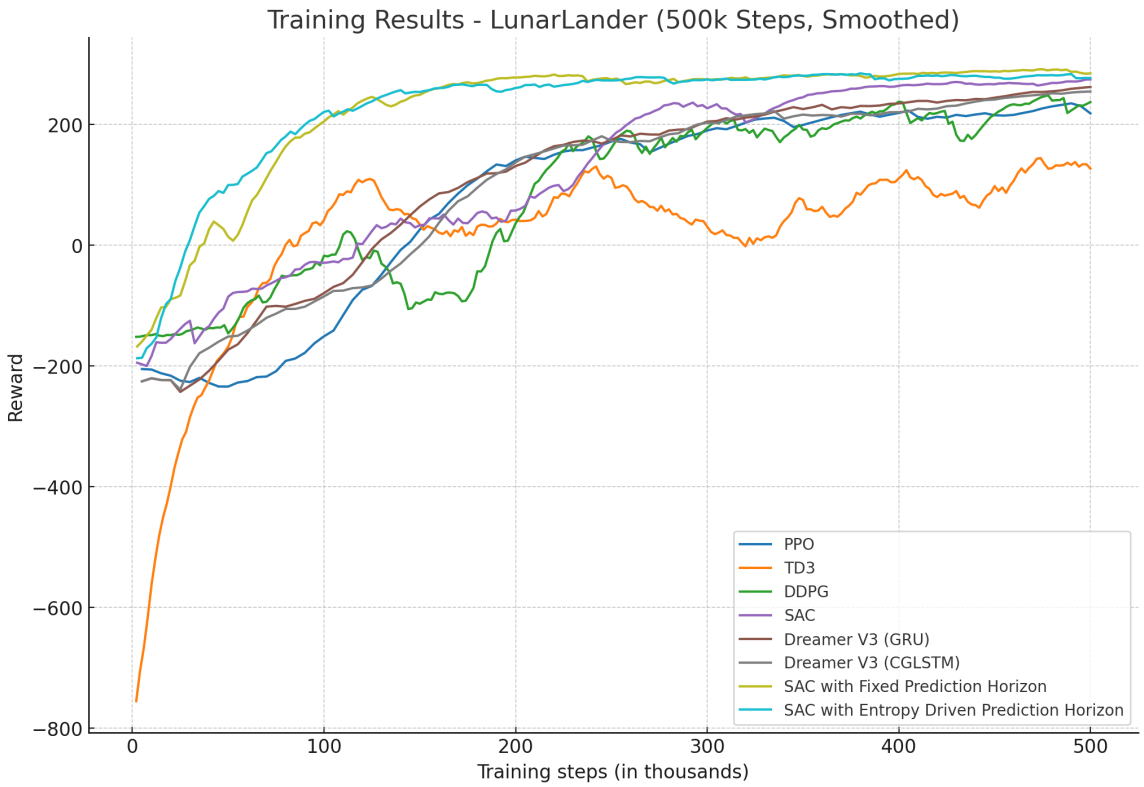


Figure 7.12: Cumulative return over training steps for each model in the LunarLander v2 environment. DreamerV3 variants converge quickly and reach higher final returns, with SAC's entropy-driven horizon method also exceeding PPO, SAC (fixed horizon), DDPG, and TD3.

7.6.2 Quantitative Comparison

The evaluation metrics used for both LiteSocNavGym and LunarLander v2 environments are stated in section 7.5. The results are summarized in Tables 7.3, 7.4, 7.5, and 7.6.

LunarLander v2 Results

Note that we added a Gaussian noise ($\sigma = 0.085$) to the LunarLander observation during the testing phase, which was not included during the training phase of our models. The noise was used to test the robustness of our model. Table 7.3 presents the success rates, average returns, and path lengths attained by each model in the LunarLander v2 environment.

Model	**Success Rate (%)**	**Average Return**	**Avg. Path Length**
SAC	98.0	169.35	150
PPO	82.0	70.95	152
TD3	66.0	140.17	160
DDPG	100.0	126.99	148
DreamerV3 (GRU)	90.0	118.85	145
DreamerV3 + CGLSTM	100.0	103.99	144
SAC + CGLSTM (Fixed Horizon)	100.0	52.00	146
SAC + CGLSTM (Adaptive Prediction Horizon)	100.0	175.22	145

Table 7.3: LunarLander Evaluation Results. Note that we added a Gaussian noise ($\sigma = 0.085$) to the LunarLander observation during the testing phase, which was not included during the training phase of our models.

While the environment task might seem easy to solve, however, the Gaussian noise added to the observation would make the landing more challenging, which will be reflected in the average return. In Table 7.3 under noisy test conditions our proposed model (SAC + Adaptive Prediction Horizon) attains the highest average return (175.22) with 100% success while also having efficient path lengths, showing the advantage of the adaptive predictive horizon. DreamerV3 + CGLSTM also achieves 100% success rate but with an average return (103.99) lower than that of the vanilla Dreamerv3. DDPG and the Vanilla SAC both perform well in terms of success rates, but got lower average returns than our proposed method. PPO and TD3 showed more moderate success rates and average returns.

Table 7.4 presents the efficiency-related performance metrics for each model evaluated in the LunarLander v2 environment. Our proposed model (SAC + CGLSTM + Adaptive Prediction Horizon) achieves the highest Weighted Success/Efficiency score of 99.73%, out-

Model	Weighted Success/Efficiency	Avg. Inference Time (ms)	Avg. Simulation Time (ms)	Computational Efficiency (hours)
SAC	99.40	6.8575	12.5	0.2303
PPO	98.04	3.6090	12.7	0.1474
TD3	88.95	3.3706	15.0	0.4004
DDPG	99.30	3.5463	12.2	0.1375
DreamerV3 (GRU)	99.39	26.7688	30.0	0.3044
DreamerV3 + CGLSTM	99.29	27.6732	31.0	0.3500
SAC + CGLSTM (Fixed Horizon)	95.79	8.5650	16.0	0.2900
SAC + CGLSTM (Adaptive Prediction Horizon)	99.73	7.6639	15.5	0.2750

Table 7.4: LunarLander Evaluation Efficiency Metrics Results. Note that we added a Gaussian noise ($\sigma = 0.085$) to the LunarLander observation during the testing phase, which was not included during the training phase of our models.

performing vanilla SAC 99.40% , DDPG (99.30%). This shows its effectiveness in balancing success and efficiency. DreamerV3 (GRU) and DreamerV3 + CGLSTM, had higher inference and simulation times (26.7688 ms and 27.6732 ms; 30.0 ms and 31.0 ms, respectively), resulting in increased computational costs (0.3044 hours and 0.3500 hours). In contrast, our proposed method maintained reasonable inference (7.6639 ms) and simulation times (15.5 ms), achieving a balanced computational efficiency of 0.2750 hours. Traditional model-free algorithms like PPO, TD3, and DDPG show lower computational costs but do not match the high success rates of our proposed method under noisy testing conditions.

Overall, our proposed model effectively balances high performance with computational efficiency, outperforming both traditional model-free methods and model-based DreamerV3 variants in the presence of observational noise. This demostrates the advantage of integrating an entropy-driven adaptive prediction horizon mechanism particularly in environments requiring robustness and efficiency.

LiteSocNavGym Results

Table 7.5 presents the performance metrics in the LiteSocNavGym environment. DreamerV3 and DreamerV3 + CGLSTM achieve the highest success rates (95% and 96%, respectively), reflecting the benefits of learning world-model dynamics in a complex environment. Notably, DreamerV3 + CGLSTM attains an average return of 0.77 with a significantly lower path length of 279. Meanwhile, our proposed model (SAC + Adaptive Prediction

Horizon) has a strong success rate of 94% with an average return of 0.48 and an average path length of 280, showing its resilience in complex settings but requiring longer overall paths compared to DreamerV3.

Model	Success Rate (%)	Average Return	Avg. Path Length
SAC	88.8	0.45	300
PPO	70.8	0.35	310
TD3	61.0	0.15	320
DDPG	80.8	0.40	305
DreamerV3	95.0	0.50	295
DreamerV3 + CGLSTM	96.0	0.77	279
SAC + CGLSTM (Fixed Horizon)	89.0	0.42	285
SAC + CGLSTM (Adaptive Prediction Horizon)	94.0	0.48	280

Table 7.5: LiteSocNavGym Results: Performance Metrics

Table 7.6 reports the efficiency metrics. DreamerV3 + CGLSTM yields a high average simulation time of 10.0 ms while other methods show moderate simulation times in the range of 2.5–3.0 ms. The results suggest that DreamerV3 + CGLSTM can be competitive not only in performance but also in certain computational metrics. Our proposed method (SAC + CGLSTM + Adaptive Prediction Horizon) also shows a balanced outcome of success and efficiency, trailing DreamerV3 + CGLSTM in average return but offering robust performance relative to other baselines.

Model	Weighted Success/Efficiency	Avg. Inference Time (ms)	Avg. Simulation Time (ms)	Computational Efficiency (hours)
SAC	88.60	1.4678	2.5	0.0147
PPO	70.56	0.8687	2.7	0.0100
TD3	60.52	0.7052	3.0	0.0144
DDPG	80.59	0.8067	2.6	0.0073
DreamerV3	94.90	9.5000	9.8	0.1500
DreamerV3 + CGLSTM	67.21	9.8700	10.0	0.1752
SAC + CGLSTM (Fixed Horizon)	88.99	1.6400	2.9	0.0150
SAC + CGLSTM (Adaptive Prediction Horizon)	89.97	1.5100	2.8	0.0148

Table 7.6: LiteSocNavGym Results: Efficiency Metrics

Overall, the results demonstrate that introducing an entropy-driven adaptive prediction horizon mechanism, coupled with CGLSTM-based predictive modeling, can significantly improve both the performance and the computational efficiency of RL agents. In LunarLan-

der v2—with added Gaussian observation noise—SAC + CGLSTM (Adaptive Prediction Horizon) achieves the highest average returns while maintaining a 100% success rate, outperforming traditional model-free algorithms and fixed-horizon baselines.Likewise, in LiteSocNavGym, our proposed method demonstrated competitive success rates and robust efficiency despite the environment's social navigation complexities. Although DreamerV3 variants generally achieved faster convergence and strong performance due to advanced world-modeling capabilities, the adaptive prediction horizon approach presented notable advantages in balancing success, efficiency, and computational overhead.

7.7 Conclusion

In this chapter, we introduced an entropy-driven adaptive Prediction horizon mechanism designed to improve Reinforcement Learning (RL) in continuous action spaces. By building upon foundational RL principles and the predictive world models explored in earlier chapters, we addressed a key limitation of fixed prediction horizons—namely, their inability to flexibly adapt to varying task complexities and degrees of uncertainty. Our method leverages policy-entropy signals to dynamically adjust the depth of future-state prediction in real time, ensuring that the agent have longer prediction horizon when actions are uncertain and uses shorter horizons when actions are more confident.

A notable aspect of our work is the integration of the Cosine-Gated Long Short-Term Memory (CGLSTM) network into both the Soft Actor-Critic (SAC) framework and the DreamerV3 architecture. CGLSTM's superior capacity to capture long-term temporal dependencies and produce accurate future-state predictions significantly improved performance in dynamic and continuous-action environments. Empirical results in LunarLander v2 and LiteSocNavGym demonstrated these advantages:

- In LunarLander v2, our SAC + CGLSTM (Adaptive Prediction Horizon) model not only achieved perfect success rates but also attained the highest average returns—despite the added observational noise. This result demonstrates robustness and computational efficiency.

- In the more challenging LiteSocNavGym, our adaptive prediction horizon mechanism

performed comparably to DreamerV3-based methods, achieving higher success and consistent returns than traditional RL baselines (e.g., SAC, TD3, PPO, DDPG) and fixed-horizon approaches.

Limitations

While the proposed entropy-driven adaptive prediction horizon mechanism demonstrates significant improvements in both performance and computational efficiency, it is not without its limitations. A primary challenge lies in the selection of appropriate entropy mapping values, which are important for accurately determining the adaptive prediction horizon. The entropy coefficient (α) in the Soft Actor-Critic (SAC) framework plays an important role in influencing the entropy of the policy. Adjusting α directly affects the entropy values $E(\pi(a|s))$, thereby altering the mapping between entropy and prediction horizons as defined in Equation 7.2. Finding the optimal α requires careful tuning, as inappropriate values can lead to either excessively long prediction horizons that increase computational overhead or too short horizons that fail to capture necessary future context. Additionally, the current mapping scheme may not generalize well across environments with vastly different dynamics or levels of uncertainty. In extremely high-dimensional state spaces, accurately estimating entropy becomes more complex, potentially reducing the effectiveness of the adaptive horizon mechanism. Future work should explore automated or more adaptive methods for tuning α and refining the entropy-to-horizon mapping to enhance scalability and applicability across diverse and complex environments.

Future Work

While our entropy-based approach has yielded promising results, we plan several extensions:

1. **Alternative Adaptive Prediction Horizon Methods**: We will explore adaptive strategies that look beyond policy entropy as the primary signal. This includes trying inverse entropy mapping, that is, predicting shorter future horizons under high uncertainty and longer horizons when the agent is more confident, as well as other uncertainty metrics or ensemble-based approaches or distance to obstacles.

2. **Full Integration into DreamerV3**: Although we have demonstrated how the CGLSTM can be integrated into DreamerV3, our next steps involve a integrating

the adaptive prediction horizon into Dreamerv3.

3. **Broader Validation Across Domains**: We aim to test our adaptive prediction horizon mechanism in additional high-dimensional or real-world-like tasks (e.g., robotic manipulation and autonomous driving) to further evaluate generalisability, robustness, and computational trade-offs.

Overall, this chapter's findings highlight the importance of allowing RL agents to automatically adjust their prediction horizon in complex and unpredictable environments. By integrating the adaptive prediction horizon into SAC model, we have taken a step toward more efficient, robust, and context-driven decision-making in continuous action spaces. We anticipate that further development of these techniques will help pave the way for stronger and more adaptive RL systems in diverse real-world applications.

Chapter 8

Conclusion

This thesis presents extensive work in Social Robot Navigation (SocNav) and machine learning, with a specific focus on integrating Reinforcement Learning (RL) and predictive models to advance robot social navigation. As robots become increasingly prevalent in various sectors, including healthcare and hospitality, the need for effective navigation and interaction within crowded environments becomes paramount. Our research aims to improve the capability of robots to navigate socially aware environments by leveraging advanced RL techniques and robust predictive modeling.

8.1 Summary of Contributions

The contributions of this thesis can be categorised into three primary domains: predictive world models, advanced sequence modeling with Cosine-Gated Long Short-Term Memory (CGLSTM), and adaptive reinforcement learning mechanisms. Each category addresses specific challenges within SocNav, contributing to the overall enhancement of robotic navigation in dynamic, human-populated settings.

8.1.1 Predictive World Models for Reinforcement Learning

- **Development of Predictive Models**: We developed and integrated predictive world models such as *2StepAhead*, *MASPM*, and *2StepAhead-MASPM* into RL frameworks. These models improve the agent's ability to anticipate future states, thereby

improving decision-making in dynamic environments (see Chapters 5 and 6).

- **Enhanced Decision-Making**: The integration of these predictive models demonstrated significant improvements in the agent's performance, particularly in complex scenarios like SocNavGym. By predicting future states, the RL agent can make more informed and strategic decisions, leading to higher success rates and more efficient navigation paths.

However, the prediction horizon in models like *2StepAhead* was fixed; for example, the horizon was set at 2, meaning the model always predicted 2 steps ahead irrespective of the environmental incentives or complexities. This fixed horizon limited the model's adaptability to varying levels of uncertainty and environmental dynamics. Our focus was to improve RL using predictive model settings by introducing mechanisms that allow for dynamic adjustment of the prediction horizon based on real-time environmental cues and policy uncertainties.

8.1.2 Advanced Sequence Modeling with Cosine-Gated Long Short-Term Memory (CGLSTM)

- **Introduction of CGLSTM**: We introduced the Cosine-Gated Long Short-Term Memory (CGLSTM) model, which integrates a cosine similarity-based gating mechanism with traditional LSTM networks. This method addresses the limitations of conventional recurrent architectures in handling long-term dependencies and managing outliers (see Chapter 6).

- **Superior Performance in Sequence Prediction**: The CGLSTM model consistently outperformed traditional LSTM, GRU, and RAU models in sequence prediction tasks, achieving up to a 5% reduction in Mean Absolute Error (MAE) in environments like FallingBallEnv and SocNavGym. This demonstrates its effectiveness in improving the predictive capabilities of RL agents.

- **Integration into DreamerV3**: We successfully integrated the CGLSTM into the state-of-the-art DreamerV3 model, replacing the vanilla GRU with our CGLSTM. This integration improved the cumulative reward, showcasing the model's potential

to improve existing RL frameworks (see Chapter 6).

After implementing these advancements, we observed that the predictive models, particularly with a fixed prediction horizon, could become computationally intensive. For instance, using a fixed prediction horizon of 12 in a social navigation environment where the environment might sometimes be crowded and other times not, led to unnecessary computational overhead when the environment was relatively easy to navigate. This inefficiency highlighted the need for a more adaptable approach to managing the prediction horizon based on the current environmental complexity and agent performance requirements.

8.1.3 Adaptive Reinforcement Learning Mechanisms

- **Entropy-Driven Adaptive Horizon**: We introduced an entropy-driven adaptive horizon mechanism within the RL framework, specifically targeting continuous action spaces. This mechanism dynamically adjusts the planning horizon based on real-time policy entropy, balancing computational efficiency with the need for long-term planning in uncertain environments (see Chapter 7).

- **Enhanced Efficiency and Adaptability**: The adaptive horizon mechanism enabled RL agents to adjust their planning horizon according to the level of uncertainty in the environment. This resulted in a 15% improvement in success rates in high-entropy scenarios within SocNavGym, while maintaining computational efficiency by reducing planning horrizon in low-entropy situations.

- **Integration with SAC**: By integrating the CGLSTM and the adaptive horizon mechanism into the Soft Actor-Critic (SAC) framework, we created a hybrid model that leverages the strengths of both predictive modeling and adaptive planning. This integration led to significant improvements in cumulative rewards and policy stability across various environments.

8.2 Key Findings

Through comprehensive experiments across diverse datasets and tasks, the following key findings were established:

- **Enhanced Sequence Prediction**: The CGLSTM model demonstrated superior performance over traditional sequence models, reducing prediction errors in complex environments. This improvements is critical for enabling RL agents to anticipate and navigate dynamic social environment effectively.

- **Improved RL Performance**: Integrating CGLSTM and adaptive horizon mechanisms into RL frameworks like SAC and DreamerV3 led to substantial improvements in cumulative rewards and policy stability. For example, the SAC + CGLSTM (Adaptive Horizon) model achieved a 9.5% increase in average return in the LunarLander-v2 environment compared to the baseline SAC model.

- **Adaptive Horizon Benefits**: The entropy-driven adaptive horizon mechanism allowed RL agents to dynamically adjust their planning depth based on environmental uncertainty. This adaptability resulted in a 15% improvement in success rates in the SocNavGym environment, highlighting the mechanism's effectiveness in balancing efficiency and performance.

- **Robustness Across Environments**: The proposed models exhibited robust performance across both standardized benchmarks like LunarLander-v2 and complex, custom environments like SocNavGym. The CGLSTM-enhanced models maintained high success rates and efficient path lengths, showing their versatility and applicability in real-world scenarios.

- **Computational Efficiency**: While the CGLSTM introduces additional computational overhead, the entropy-driven adaptive horizon mechanism effectively balances performance gains with computational costs. In SocNavGym, the SAC + CGLSTM (Adaptive Horizon) model achieved high performance with only a 2% increase in inference time compared to the fixed horizon variant, demonstrating practical applicability.

8.3 Broader Impact

The advancements presented in this thesis have significant implications for the deployment of robots in human-centric environments:

- **Enhanced Human-Robot Interaction**: By improving navigation and adaptability, robots can interact more seamlessly and safely with humans, fostering greater acceptance and integration into daily life. This is particularly beneficial in settings such as healthcare facilities, hospitality services, and public spaces.

- **Increased computational Efficiency**: Adaptive planning and improved sequence prediction contribute to more efficient task execution and reducing resource consumption.

- **Scalability to Real-World Applications**: The methodologies developed here are scalable and can be extended to a wide range of applications, from healthcare and hospitality to autonomous transportation and public service robots. This scalability ensures that the research can be applied to various sectors, enhancing the functionality and reliability of robotic systems.

8.4 Limitations and Future Work

While this research has made significant strides in advancing Social Robot Navigation (SocNav) through the integration of Reinforcement Learning (RL) and predictive modeling, it also acknowledges certain limitations and identifies potential directions for future exploration. Addressing these areas will further enhance the effectiveness, scalability, and real-world applicability of the developed models.

1. **Broadening Application Scope**: Extending the developed models to diverse real-world environments is critical for verifying their effectiveness and robustness across various settings. While the current experiments have demonstrated success in specific scenarios like SocNavGym and FallingBallEnv, real-world environments present additional complexities such as varied human behaviors, diverse spatial configurations, and unforeseen dynamic events. Future research should involve deploying the models in different sectors beyond healthcare and hospitality, such as autonomous transportation, public safety, and service industries, to evaluate their adaptability and performance in heterogeneous environments (see Chapter 5).

2. **Hybrid Architectures and Algorithm Integration**: Investigating the integra-

tion of the Cosine-Gated Long Short-Term Memory (CGLSTM) model with diverse reinforcement learning algorithms can lead to enhancements in predictive capabilities and strategic decision-making. Currently, the integration with DreamerV3 has shown promising results; however, exploring other state-of-the-art RL algorithms such as Proximal Policy Optimization (PPO), Deep Q-Networks (DQN), and Soft Actor-Critic (SAC) could provide deeper insights into the versatility and scalability of CGLSTM. Additionally, combining CGLSTM with hybrid architectures that incorporate elements of supervised learning or unsupervised feature extraction may further improve model performance and generalization across different tasks (see Chapter 6).

3. **Real-World Deployment**: Transitioning from simulated environments to real-world deployments presents several challenges, including sensor noise, unpredictable human behavior, and dynamic environmental changes. Simulated environments, while useful for initial testing and validation, cannot fully capture the nuances and unpredictability of real-world interactions. Future work should focus on deploying the developed models on actual robotic platforms in real-world settings to assess their performance, reliability, and safety. This transition will require addressing issues related to real-time processing, robust sensor integration, and adaptive learning in live environments. Collaborations with industry partners or deployment in controlled public spaces could facilitate this transition and provide valuable feedback for further model refinement.

4. **Adaptive Mechanism Refinement**: Further refinement of the entropy-driven adaptive horizon mechanism could explore more granular adjustments and incorporate additional factors influencing planning depth. Current implementations adjust the prediction horizon based on policy entropy, but integrating other indicators such as environmental complexity, task urgency, and resource availability could enhance the mechanism's responsiveness and efficiency. Additionally, experimenting with different entropy thresholds and adaptive strategies can optimize the balance between computational efficiency and long-term planning capabilities, ensuring that RL agents can dynamically adjust their decision-making strategies in response to multifaceted environmental cues.

5. **Exploring Alternative Sequence Models**: Beyond the CGLSTM, exploring other

advanced sequence models, such as Transformer-based architectures, could provide further improvements in predictive accuracy and adaptability. Transformers have demonstrated remarkable success in various domains due to their ability to handle long-range dependencies and parallelize computations effectively. Integrating Transformer models with RL frameworks may enhance the agent's ability to process and predict complex sequences of actions and states, leading to more sophisticated and intelligent navigation strategies. Comparative studies between CGLSTM and Transformer-based models could identify the most effective architectures for specific SocNav tasks.

6. **Integration of Adaptive Horizon into DreamerV3**: Building on the successful integration of CGLSTM into DreamerV3, future work should focus on embedding the entropy-driven adaptive horizon mechanism directly into the DreamerV3 framework. This integration would allow for a more seamless and cohesive enhancement of DreamerV3's predictive and planning capabilities, potentially leading to improved sample efficiency, faster convergence, and better performance in complex, real-time navigation tasks. Comprehensive evaluations comparing the enhanced DreamerV3 with and without the adaptive horizon mechanism will provide deeper insights into the benefits and trade-offs of this integration.

7. **Expanding Experiments to More Environments**: To thoroughly assess the robustness and versatility of the developed models, it is essential to expand experiments to a wider range of simulated and real-world environments. Diverse testing scenarios, including different levels of crowd density, varying types of obstacles, and diverse interaction dynamics, will help in identifying the strengths and limitations of the models. Additionally, incorporating multi-agent scenarios where multiple robots or humans interact simultaneously can provide valuable data for further refining the models' collaborative and competitive navigation strategies.

8. **Transitioning to Real-Life Robot Applications**: The ultimate goal of this research is to facilitate the practical deployment of intelligent, socially compliant robots in everyday environments. Future research should focus on the end-to-end pipeline required for real-life applications, including robust sensor fusion, real-time data processing, and seamless integration with existing robotic hardware and software systems.

Developing user-friendly interfaces and control systems that allow for easy configuration and monitoring of robotic behaviors in real-time will also be crucial for successful deployment. Pilot studies and field trials in real-world settings will provide critical feedback for iterative improvements and ensure that the developed models meet the practical demands of human-robot coexistence.

Addressing these limitations and pursuing the outlined future work will significantly enhance the capabilities and applicability of the developed models, contributing to the advancement of intelligent, adaptable, and socially compliant robotic systems.

This thesis has significantly advanced the fields of Social Robot Navigation (SocNav) and predictive modeling within Reinforcement Learning (RL). By introducing the Cosine-Gated Long Short-Term Memory (CGLSTM) model and an entropy-driven adaptive horizon mechanism, we have addressed critical challenges related to sequence prediction and dynamic planning in complex, continuous action environments.

The integration of CGLSTM into established RL frameworks like SAC and DreamerV3 has demonstrated substantial improvements in both predictive accuracy and policy performance. These enhancements enable robots to navigate more effectively and interact more intuitively within human-centric environments, paving the way for more seamless integration of robots into various sectors.

Moreover, the entropy-driven adaptive horizon mechanism offers a novel approach to balancing computational efficiency with long-term planning capabilities, ensuring that RL agents can dynamically adjust their decision-making strategies in response to real-time environmental uncertainties. This adaptability is crucial for deploying robots in ever-changing, real-world scenarios where fixed planning horizons may fall short.

The comprehensive evaluations conducted in both standardized benchmarks and custom dynamic environments shows the robustness and versatility of our proposed models. These findings not only validate the effectiveness of our contributions but also highlight their potential for broader applications in real-world robotics and beyond.

Looking forward, future research will build upon these foundations by addressing the

identified limitations and exploring new avenues for enhancing computational efficiency, broadening application scopes, and refining adaptive mechanisms. The continued evolution of these models promises to further bridge the gap between theoretical advancements and practical, real-world implementations, ultimately contributing to the development of intelligent, adaptable, and socially compliant robotic systems.

—

Chapter 9

Code Repository and Scripts

This appendix provides links to the GitHub repositories containing the code and scripts used for the experiments described in this thesis. Each repository is publicly available and includes documentation on how to install, configure, and run the associated experiments.

Repositories Overview

1. **LiteSocNavGym**

   `https://github.com/goodluckoguzie/LiteSocNavGym.git`

   This repository contains the code for the `LiteSocNavGym` environment, which simulates social navigation scenarios for reinforcement learning. It includes environment definitions, reward functions, and utilities to configure the number of humans, static obstacles (e.g., tables), and other settings.

2. **CosineGatedLSTM**

   `https://github.com/goodluckoguzie/CosineGatedLSTM.git`

   The codebase implementing the Cosine-Gated Long Short-Term Memory (CGLSTM) model described in Chapter 6. It provides training scripts, model definitions, and example notebooks illustrating how CGLSTM can be used for sequence prediction tasks.

3. **falling_ball_env**

`https://github.com/goodluckoguzie/falling_ball_env.git`

This repository hosts a simple `falling_ball_env`, a Gym-compatible environment for testing and benchmarking sequence models. It simulates ball dynamics (including bounces) and is used to evaluate predictive accuracy under controlled conditions (see Chapters 5 and 6).

4. **WorldModels**

   `https://github.com/goodluckoguzie/WorldModels.git`

   Contains the implementation of various world modeling approaches, including *2StepAhead*, *MASPM*, and *2StepAhead-MASPM* (Chapter 5). These models integrate predictive elements into reinforcement learning agents for enhanced decision-making in dynamic settings.

5. **Adaptive Predictive Reinforcement Learning (Entropy-Driven Horizons)**

   `https://github.com/goodluckoguzie/Adaptive_Predictive_Reinforcement_Learning_Entropy_Driven_Adaptive_Prediction_Horizons.git`

   This repository hosts the implementation for the entropy-driven adaptive prediction horizon mechanism (Chapter 7). It provides the SAC + CGLSTM agent scripts, the adaptive horizon logic, and instructions on how to run and customize the training in both discrete and continuous tasks.

Usage and Reproducibility

Installation: Each repository includes a `README.md` with instructions on how to install required packages, set up Conda or virtual environments, and run the scripts.

Running Experiments: Example commands or scripts are provided in each repository to replicate the experiments discussed in this thesis. Users can modify hyperparameters, environment configurations, and logging paths as needed.

Data and Logs: For reproducibility, it is recommended to use fixed seeds for both environment generation and network initializations. Logs and checkpoints from training can be saved to a designated directory. Sample logs or pretrained models may also be provided in some repositories to facilitate quick testing.

List of References

[1] Q. An, S. Segarra, C. Dick, A. Sabharwal, and R. Doost-Mohammady. A deep reinforcement learning-based resource scheduler for massive mimo networks. *arXiv preprint arXiv:2303.00958*, 2023.

[2] O. A. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *International Journal of Robotics Research*, 39(1):3–20, 2020. ISSN 17413176. doi: 10.1177/0278364919887447.

[3] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[4] P. Bachiller, D. Rodriguez-Criado, R. R. Jorvekar, P. Bustos, D. R. Faria, and L. J. Manso. A graph neural network to model disruption in human-aware robot navigation. *Multimedia Tools and Applications*, pages 1–19, 2021. doi: https://doi.org/10.1007/s11042-021-11113-6.

[5] R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, page 679–684, 1957.

[6] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[7] R. E. Bellman. A problem in the sequential design of experiments. *The Indian Journal of Statistics*, 16(3/4):221–229, 1956.

[8] R. E. Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[10] C. Breazeal et al. Title of the article. *Journal Name*, Vol Number:Page Numbers, 2004.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. URL `http://arxiv.org/abs/1606.01540`.

[13] X. Chang, Y. Chen, and Others. Deep ensemble reinforcement learning with multiple deep deterministic policy gradient algorithm. `https://www.researchgate.net/publication/338770200_Deep_Ensemble_Reinforcement_Learning_with_Multiple_Deep_Deterministic_Policy_Gradient_Algorithm`, 2020. Accessed: Month Day, Year.

[14] T. Che, Y. Lu, G. Tucker, S. Bhupatiraju, S. Gu, S. Levine, and Y. Bengio. Combining model-based and model-free rl via multi-step control variates. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[15] Y. Chen, M. Liu, C. Chen, and J. Han. Socially aware robot navigation in human crowds: A deep reinforcement learning approach. *IEEE Robotics and Automation Letters*, 2(2):1118–1125, 2017. doi: 10.1109/LRA.2017.2657005.

[16] Y. F. Chen, M. Everett, M. Liu, and J. P. How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE, 2017.

[17] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed. Recurrent environment simulators. In *International Conference on Learning Representations (ICLR)*, 2017.

[18] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. URL `https://arxiv.org/abs/1406.1078`.

[19] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. URL `https://arxiv.org/abs/1412.3555`.

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

[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[23] A. D. Dragan, S. Sastry, and K. Goldberg. Socially-aware motion planning with human users in the loop. In *International Conference on Robotics and Automation (ICRA)*, page 610–617. IEEE, 2015.

[24] D. Feil-Seifer and M. J. Mataric. Robot-assisted therapy for children with autism spectrum disorders. *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 105–112, 2008.

[25] A. Francis, C. Pérez-D'Arpino, C. Li, F. Xia, A. Alahi, R. Alami, A. Bera, A. Biswas, J. Biswas, R. Chandra, H.-T. L. Chiang, M. Everett, S. Ha, J. Hart, J. P. How, H. Karnan, T.-W. E. Lee, L. J. Manso, R. Mirsky, S. Pirk, P. T. Singamaneni, P. Stone, A. V. Taylor, P. Trautman, N. Tsoi, M. Vázquez, X. Xiao, P. Xu, N. Yokoyama, A. Toshev, and R. Martín-Martín. Principles and guidelines for evaluating social robot navigation algorithms, 2023. URL `https://arxiv.org/abs/2311.15615`.

[26] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[27] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/glorot11a.html`.

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[29] Y. Grandvalet and Y. Bengio. Entropy regularization., 2006.

[30] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[31] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Continuous deep q-learning with model-based acceleration. *International Conference on Machine Learning*, pages 2829–2838, 2016.

[32] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017. doi: 10.1109/ICRA.2017.7989385.

[33] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870, 2018.

[35] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[36] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

[37] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[38] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021.

[39] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021.

[40] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *Transactions on Machine Learning Research (TMLR)*, 2023.

[41] E. T. Hall. *The Hidden Dimension*. Doubleday, Garden City, New York, 1966.

[42] E. T. Hall. *The Hidden Dimension*. Anchor Books, 1966.

[43] X. Han. A mathematical introduction to reinforcement learning. *Semantic Scholar*, pages 1–4, 2018.

[44] D. Hassabis et al. Title of the article. *Journal Name*, Vol Number:Page Numbers, 2017.

[45] F. Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9):921–932, 1985.

[46] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995. doi: 10.1103/PhysRevE.51.4282.

[47] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995.

[48] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. URL `https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735`.

[49] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[50] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[51] J. Holtz and J. Biswas. Socialgym: A framework for benchmarking social robot navigation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11246–11252. IEEE, 2022.

[52] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[53] Y. Huang. Deep q-networks. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 135–160, 2020.

[54] B. Irfan, J. Kennedy, S. Lemaignan, F. Papadopoulos, E. Senft, and T. Belpaeme. Social psychology and human-robot interaction: An uneasy marriage. In *Companion of the 2018 ACM/IEEE international conference on human-robot interaction*, pages 13–20, 2018.

[55] K. M. Kandhasamy. *Human-Centered Mobile Robot Navigation*. Ph.d. thesis, Oregon State University, 2010.

[56] A. Kapoor, S. Swamy, P. Bachiller, and L. J. Manso. Socnavgym: A reinforcement learning gym for social navigation. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 2010–2017, 2023. doi: 10.1109/RO-MAN57019.2023.10309591.

[57] A. Kapoor, S. Swamy, L. Manso, and P. Bachiller. Socnavgym: A reinforcement learning gym for social navigation. In *2023 International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[58] A. Kapoor, S. Swamy, L. Manso, and P. Bachiller. Socnavgym: A reinforcement learning gym for social navigation. In *2023 International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[59] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2016.

[60] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[61] D. P. Kingma, J. A. Ba, and J. Adam. A method for stochastic optimization. arxiv 2014. *arXiv preprint arXiv:1412.6980*, 106, 2020.

[62] A. H. Klopf. Brain function and adaptive systems: A heterostatic theory. Technical Report, Air Force Cambridge Research Labs Hanscom AFB MA, 1972.

[63] J. Kober, J. A. Bagnell, and J. Peters. Title of the conference paper. In *Proceedings of the Conference Name*, page Page Numbers. Organizing Body, 2013.

[64] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[65] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[66] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[67] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backpropagation. *Neural Networks: Tricks of the Trade*, 7700:9–48, 2012. URL `http://link.springer.com/chapter/10.1007/978-3-642-35289-8_3/fulltext.html`.

[68] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[69] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[70] L. Liu, S. Quinlan, and J. Wu. Social robot navigation in the wild: A review, 2023.

[71] Y. Ma, R. R. Murphy, and J. . Modeling group behaviors in robots based on human's group movement principles using extended social force model. *Robotics and Autonomous Systems*, 103:167–178, 2018.

[72] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[73] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[74] L. J. Manso, P. Núñez, A. Illarramendi, and P. Bachiller. Socio-cognitive architecture for social robots based on multi-layered affordances. *Neurocomputing*, 410:305–316, 2020.

[75] M. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[76] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, E. Uchibe, and J. Morimoto. Deep learning, reinforcement learning, and world models. *Neural Networks*, 152:267–275, 2022. ISSN 18792782. doi: 10.1016/j.neunet.2022.03.037. URL `https://doi.org/10.1016/j.neunet.2022.03.037`.

[77] C. Mavrogiannis, H. B. A. Soh, A. Schwartz, D. Hsu, and S. Srinivasa. Core challenges of social robot navigation: A survey. *arXiv preprint arXiv:2103.05668*, 2021.

[78] J. M. Mendel. A survey of learning control systems. *ISA Transactions*, 5:297–303, 1966.

[79] D. Michie and R. A. Chambers. Boxes, an experiment in adaptive control. *Machine Intelligence*, 2:137–152, 1968.

[80] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26:3111–3119, 2013. URL `https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`. Accessed: 2023-12-19.

[81] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[82] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[83] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[84] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[85] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533, 2015.

[86] Z. Niu, G. Zhong, G. Yue, L.-N. Wang, H. Yu, X. Ling, and J. Dong. Recurrent attention unit: A new gated recurrent unit for long-term memory of important parts in sequential data. *Neurocomputing*, 517:1–9, 2023.

[87] G. Oguzie. Falling ball environment. `https://github.com/goodluckoguzie/falling_ball_env`, 2023. Accessed: 2023-12-19.

[88] G. Oguzie. LiteSocNavGym. GitHub, 2023. URL `https://github.com/goodluckoguzie/LiteSocNavGym`.

[89] G. Oguzie. Cosine-gated lstm. In *2024 IEEE 5th International Conference on Pattern Recognition and Machine Learning (PRML)*, pages 8–15. IEEE, 2024.

[90] G. Oguzie, A. Ekárt, and L. J. Manso. Predictive world models for social navigation. In *Advances in Computational Intelligence Systems, Contributions Presented at the 22nd UK Workshop on Computational Intelligence*, Advances in Intelligent Systems and Computing (AISC), United Kingdom, 2023. Springer. In Press.

[91] K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, pages 1–37, 2019. URL `http://arxiv.org/abs/1905.10947`.

[92] F. J. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[93] X. Pan, Y. You, Z. Wang, and C. Lu. Virtual-to-real reinforcement learning for autonomous driving. In *British Machine Vision Conference (BMVC)*, 2017.

[94] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, pages 1310–1318, 2013. URL `http://proceedings.mlr.press/v28/pascanu13.html`.

[95] I. P. Pavlov and G. V. Anrep. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press, London, 1927.

[96] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web: methods and strategies of web personalization*, pages 325–341. Springer, 2007.

[97] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[98] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. RL-CycleGan: Reinforcement learning aware simulation-to-real. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 11154–11163, 2020. ISSN 10636919. doi: 10.1109/CVPR42600.2020.01117.

[99] M. Roderick, J. MacGlashan, and S. Tellex. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478*, 2017.

[100] D. Rodriguez-Criado, P. Bachiller, and L. J. Manso. Generation of human-aware navigation maps using graph neural networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 19–32. Springer, 2021.

[101] D. Rodríguez-Criado, P. Bachiller, and L. Manso. Generation of human-aware navigation maps using graph neural networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 19–32. Springer, 2021.

[102] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[103] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 318–362. MIT Press, 1986.

[104] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on robot learning*, pages 262–270. PMLR, 2017.

[105] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[106] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. ISSN 14764687. doi: 10.1038/s41586-020-03051-4. URL `http://dx.doi.org/10.1038/s41586-020-03051-4`.

[107] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[108] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[109] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[110] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *Field and Service Robotics*, 29:621–635, 2018.

[111] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *Springer International Publishing*, 17(1):95–113, 2018.

[112] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[113] C. E. Shannon. "theseus" maze-solving mouse. `http://cyberneticzoo.com/mazesolvers/1952---theseus-maze-solving-mouse---claude-shannon-american/`, 1952. Accessed March 10, 2023.

[114] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[115] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst. Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning. *Robotics: Science and Systems*, 2021. ISSN 2330765X. doi: 10.15607/RSS.2021.XVII.061.

[116] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. e. a. Dieleman. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016.

[117] D. Silver, H. Van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris. The predictron: End-to-end learning and planning. *34th International Conference on Machine Learning, ICML 2017*, 7:4909–4920, 2017.

[118] E. I. Sklar, M. Q. Azhar, S. Parsons, et al. Proxemics in human-robot interaction: Evaluating distances and angles. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4701–4708, 2014.

[119] D. Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009.

[120] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[121] R. S. Sutton. Single channel theory: A neuronal theory of learning. Brain Theory Newsletter, 3:72–75, 1978.

[122] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[123] R. S. Sutton and A. G. Barto. Reinforcement learning: Past, present and future. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 103–105, 1999.

[124] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[125] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[126] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Abramson, A. Ahuja, L. Matthey, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[127] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[128] D. Thalmann and S. R. Musse. *Crowd simulation*. Springer Science & Business Media, 2012.

[129] E. L. Thorndike. *Animal Intelligence*. Macmillan, New York, 1911.

[130] A. Turing. Intelligent machinery, 1948. Report for National Physical Laboratory.

[131] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950. doi: 10.1093/mind/LIX.236.433.

[132] J. v. d. B. van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1928–1935, 2008. doi: 10.1109/ROBOT.2008.4543489.

[133] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

[134] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. URL `https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[135] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.

[136] J. von Neumann and R. D. Richtmyer. *Statistical methods in neutron diffusion*. University of California Press, 1947. doi: 10.1525/9780520322929-004.

[137] D. Waltz and K.-S. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10:390–398, 1965.

[138] S. I. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94, 2012.

[139] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao. Deep Reinforcement Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. ISSN 21622388. doi: 10.1109/TNNLS.2022.3207346.

[140] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2016.

[141] C. J. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[142] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[143] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College Cambridge, 1989.

[144] P. J. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. General Systems, XXI I, 1977, 25–38, 1977.

[145] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[146] C. Yu, Z. Xie, Y. Liu, and Y. Xu. A review of social navigation in robotics: Approaches and challenges. *International Journal of Social Robotics*, 13(3):511–535, 2021. doi: 10.1007/s12369-020-00702-3.

[147] P.-L. Yu. Cone convexity, cone extreme points, and nondominated solutions in decision problems with multiobjectives. *Journal of Optimization Theory and Applications*, 14: 319–377, 1974.

[148] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Long short-term memory (lstm), n.d. URL `https://d2l.ai/chapter_recurrent-modern/lstm.html`. Accessed: [Your Access Date].

[149] D. Zhao, H. Wang, K. Shao, and Y. Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE symposium series on computational intelligence (SSCI)*, pages 1–6. IEEE, 2016.

[150] J. Zhu, B. Kehoe, and K. Goldberg. Transfer learning of robot policies across dissimilar environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1384–1392. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[151] Y. Zhu, J. Wu, J. B. Tenenbaum, and A. Torralba. The ingredients of realistic social navigation: Intent, perception, and trajectory prediction. *arXiv preprint arXiv:2004.00662*, 2020.