

Robust Reinforcement Learning for Linear Temporal Logic Specifications with Finite Trajectory Duration

by

Soroush Mortazavi Moghaddam

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

© Soroush Mortazavi Moghaddam 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

[Linear Temporal Logic \(LTL\)](#) is a formal behavioral specification language that offers a mathematically unambiguous and succinct way to represent operating requirements for a wide variety of systems, including autonomous and robotic systems. Traditional methods in this domain rely on model-checking approaches to ensure that a devised policy adheres to the provided specification. However, these methods are limited in the scope of problems they can solve and often lack generalizability to novel specifications and environments. Despite progress in synthesizing satisfying policies for [LTL](#) specifications under different operating conditions, learning policies that reliably satisfy complex [LTL](#) specifications in challenging environments remains an open problem. With the emergence of [Machine Learning \(ML\)](#) approaches, researchers have explored the use of [ML](#)-based techniques with LTL policy synthesis. Among the various approaches investigated, [Reinforcement Learning \(RL\)](#) has garnered particular attention for this task.

While [LTL](#) specifications are evaluated over infinite-length trajectories, this work focuses on satisfying a class of specifications within a finite number of steps, as is to be expected in most real-world applications involving robotic or autonomous systems where the run-time of the robot is limited before it needs to recharge itself, e.g., a robot vacuum which has to perform certain cleaning tasks before recharge. Therefore, in this work, an [RL](#)-based technique is developed for the problem of generating trajectories of a system that satisfy a given [LTL_f](#) specification in a system with finite (discrete) states and actions and a priori unknown transition probabilities modeled as a [Markov Decision Process \(MDP\)](#). The proposed approach builds upon the popular AlphaGo Zero [Reinforcement Learning \(RL\)](#) framework, which has found great success in the two-player game of Go, to learn policies that can satisfy an [LTL_f](#) specification given a limit on the trajectory duration.

In this thesis, first the motivation and the necessary background on the problem are provided, followed by a brief overview of existing methods. Then the problem statement is introduced, the proposed methodology and its variants are presented, and extensive simulations of complex robot motion planning problems are conducted and their results are explained. These simulations demonstrate how the approach achieves higher success rates under time constraints compared to state-of-the-art methods. The thesis concludes with a section discussing potential directions for future work and examining the results and their implications for the work completed.

Key words: [Reinforcement Learning \(RL\)](#), [Monte Carlo Tree Search \(MCTS\)](#), [Linear Temporal Logic \(LTL\)](#), Planning, Robotics

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Yash Vardhan Pant, and Sebastian Fischmeister for their constant support and invaluable guidance throughout the entirety of this research journey. Their mentorship has been a source of inspiration, and I am grateful for the opportunity to learn under their guidance.

I am also indebted to my family and friends for their support, both in and out of research, and for consistently pushing me to do my best.

Dedication

Dedicated to my parents, MohammadReza and Fakhry, whose love and support have been the guiding light throughout life and my journey. None of this would have been possible without them.

Table of Contents

Author’s Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	ix
List of Tables	xi
List of Abbreviations	xii
1 Introduction	1
1.1 The Problem at a High Level	1
1.2 RL for LTL	3
1.3 Linear Temporal Logic over Finite Traces (LTL_f)	5
1.4 AlphaGo Zero and LTL_f Synthesis	5
1.5 Outline of the Thesis	7

2	Literature Review	8
2.1	Requirement Specification and Verification via LTL	9
2.2	Traditional LTL Task Satisfaction Methods	9
2.3	MCTS and Neural Networks	11
2.4	LTL and Policy Synthesis via RL	12
2.5	Proposed Model-Free RL Method for LTL_f Specifications	16
3	Preliminaries	17
3.1	Markov Decision Processes (MDPs):	17
3.2	Linear Temporal Logic over Finite Traces (LTL_f):	19
3.2.1	LTL_f Semantics	20
3.2.2	Evaluating Trajectories of Labeled MDP States	21
3.3	Limit Deterministic Büchi Automata (LDBA):	22
3.4	The Product MDP	23
3.5	AlphaGo Zero And The Game of Go	25
3.6	Problem Formulation	27
4	Methodology	28
4.1	From the Game of Go to LTL_f Tasks	28
4.1.1	Adapting MCTS for LTL_f Objectives	29
4.1.2	Overall View of the Proposed Method	30
4.2	Neural Network for Policy and Value Function Estimation	31
4.2.1	Network Input Format	32
4.2.2	Network Architecture	33
4.3	MCTS for Policy Improvement	34
4.3.1	Reward Function Engineering	38
4.4	Training the Network and Extracting a Policy	39

5	Simulation Studuies	41
5.1	Environment Dynamics	41
5.1.1	Case 1 - Sequential Delivery with Varying Number of Unsafe Zones	42
5.1.2	Case 2 - Craft-world	43
5.1.3	Case 3 - Office World	46
5.2	Simulation Results and Discussion	47
5.2.1	Sequential Delivery with Varying Number of Unsafe Zones	47
5.2.2	The Craft-World Tasks	49
5.2.3	The Office-World Task	52
5.3	Time Analysis of the Different Approaches	53
5.3.1	Summary of Results	54
6	Extensions for the Current Work and Conclusion	55
6.1	Improved Performance for New Environments via Pre-Training	56
6.2	Scaling to Multiple Agents	58
6.3	Discussion and Conclusion	63
6.3.1	Limitations and future work	64
6.3.2	Conclusion	65
	References	66

List of Figures

1.1	An example for a task of gathering wood, iron, and taking it to a factory, in the craft-world (see Sec. 5.1.2 for more details about the craft-world environment) and a suggested trajectory that satisfies it. See Example 1.1.1 for more details on this example.	2
3.1	A simple 3-cell environment and its MDP. Actions are going up, right, down and left and are shown on the arrows in the MDP.	18
3.2	The LDBA of the LTL formula: $\Box \neg d \wedge \Diamond a$. Double circles indicate accepting states. Arrow labels indicate input alphabets to the LDBA.	23
3.3	Illustrative example of the product MDP of the LDBA of the LTL specification $\Box \neg d \wedge \Diamond a$ corresponding to the example in Fig. 3.3, and the MDP in Fig. 3.1	25
3.4	A game of Go being played on a grid board.	26
4.1	A single iteration of stages in Fig. 4.4 depicted for better visualization. At each step, the agent gathers many MCTS samples and chooses its next move from the result of these samples. This process continues until either the task is solved or time has run out.	30
4.2	Illustrative example of the encoding of an MDP state for a simple 4-celled environment, and its channeled format for the network	33
4.3	Example network architecture for a 10×10 grid-world, with a hypothetical LTL_f specification that results in 7 LDBA states	34
4.4	The overall procedure of how the proposed method works. At each state, actions are sampled from the improved policies gathered via MCTS, then used to train the network which in turn helps the MCTS with better rollouts later on.	35

4.5	Illustration of how MCTS works. On the left a single rollout has unfolded, while on the right the result of many such rollouts has resulted in a probability distribution over the next immediate actions, i.e., a policy.	36
5.1	Environmets for the sequential delivery case (Fig. 5.1a), with up to six unsafe zones (Fig. 5.1c).	43
5.2	The 10×10 craft-world, as implemented for this work. The tasks to carry out for this environment were borrowed from [5].	44
5.3	Environment for the office-world (see Sec. 5.1.3).	47
5.4	Results for the sequential delivery case study in stochastic environments. .	48
5.5	Results for the Craft-World Tasks in the Stochastic Setting (Part 1)	49
5.5	Results for the Craft-World Tasks in the Stochastic Setting (Part 2)	50
5.5	Results for the Craft-World Tasks in the Stochastic Setting (Part 3)	50
5.5	Results for the Craft-World Tasks in the Stochastic Setting (Part 4)	51
5.5	Results for the Craft-World Tasks in the Stochastic Setting (Part 5)	51
5.6	Results for the office world in the stochastic environment.	52
6.1	An example of the random 10×10 grild-world created.	57
6.2	Compariosn of average success rates for the ten test worlds.	60
6.3	Environment for the multi-agent setup.	61
6.4	Success rates for the multi-agent setups with different number of agents. .	63

List of Tables

5.1	Training time comparison between our method and CSRL for different numbers of danger zones. In this table, m stands for minutes and h stands for hours.	53
5.2	Training time comparison for the office-world task (m stands for minutes).	53
5.3	Training time comparison for craft-world tasks between our method and CSRL. The CSRL training time for the ninth task is not available. In this table, m stands for minutes and h stands for hours.	54
6.1	Average Success Rates for Each Iteration of Training on Test Set Worlds .	58
6.2	Average Success Rates for Each Iteration with Pre-trained Model	59

List of Abbreviations

LT_L Linear Temporal Logic over Finite Traces [iii](#), [vi](#), [vii](#), [ix](#), [2](#), [4–7](#), [14–16](#), [18](#), [19](#), [21](#), [23](#), [24](#), [27–29](#), [31](#), [32](#), [34](#), [40](#), [42–44](#), [46](#), [48](#), [49](#), [56](#), [61–63](#), [65](#)

AI Artificial Intelligence [6](#), [11](#)

DFA Deterministic Finite Automata [15](#)

GNNs Graph Neural Networks [13](#)

LDBA Limit Deterministic Büchi Automata [vii](#), [ix](#), [12](#), [18](#), [22–25](#), [32–34](#), [38](#), [39](#), [41](#), [48](#), [56](#), [61](#), [64](#)

LTL Linear Temporal Logic [iii](#), [vi](#), [vii](#), [ix](#), [xii](#), [1–10](#), [12–16](#), [19](#), [21–27](#), [37](#), [39](#), [64](#), [65](#)

MCTS Monte Carlo Tree Search [iii](#), [vii](#), [ix](#), [x](#), [4](#), [6–8](#), [11](#), [12](#), [19](#), [26](#), [28–32](#), [34–40](#), [42](#), [48](#), [56](#), [59–65](#)

MDP Markov Decision Process [iii](#), [ix](#), [3](#), [6](#), [7](#), [12](#), [15–18](#), [21](#), [23–25](#), [27](#), [28](#), [31–35](#), [37–41](#), [56](#), [59](#), [61](#), [64](#), [65](#)

MDPs Markov Decision Processes [vii](#), [3](#), [6](#), [17](#), [18](#), [25](#)

ML Machine Learning [iii](#), [3](#), [6](#), [15](#)

NAS Neural Architecture Search [11](#)

NN Neural Network [11](#), [12](#), [26](#)

NNs Neural Networks [11](#)

QRM Q-learning for Reward Machines 13

REPS Relative Entropy Policy Search 14, 15

RL Reinforcement Learning iii, vi, vii, 1, 3–8, 12, 15, 16, 19, 29, 30, 42, 65

RMs Reward Machines 13, 41

Chapter 1

Introduction

This chapter provides an informal overview of the main problem this thesis aims to address. In it [Reinforcement Learning \(RL\)](#) is proposed as a potential framework that could provide a solution for the problem at hand. The conclusion highlights the main contributions of this research and outlines a roadmap for the remainder of the thesis.

1.1 The Problem at a High Level

[LTL](#) [77] provides a concise and expressive language to formally express objectives, constraints, and temporal requirements in a broad range of applications, including autonomous systems [54, 57, 111]. That is why it has become a popular choice for high-level task specification for agents for tasks such as planning, robot control, and navigation, especially in recent years (see chapter 2 for more details).

Although the current work explores applications of [LTL](#) in the mentioned domains, historically, its applicability has not been limited to them. [LTL](#) first gained popularity in the field of formal verification of software [17, 22, 44, 77], where it was used to specify and verify properties of systems that evolve over time, such as ensuring that a particular state is eventually reached or that certain conditions always hold throughout the execution of a program. [LTL](#) has been used in a variety of other areas, providing a robust framework for formal verification, control synthesis, and the specification of system behaviors in many engineering and computer science disciplines [13, 46, 100] (see chapter 2 for more examples).

This thesis addresses the challenge of solving task specifications using [LTL](#) within a system where states and actions are finite and discrete (the case study systems in this

1.2 RL for LTL

The problem of synthesizing satisfactory policies for LTL specifications has a long history [1]. As a formal language capable of expressing a wide range of system behaviors and requirements, LTL has become increasingly relevant in the fields of robotics and autonomous systems [54, 57, 111]. Traditionally, model checking approaches have dominated this domain due to their ability to devise precise policies for given specifications [91, 102, 104]. However, these methods often struggle with generalizability and scalability when facing stochastic or complex environments [86, 103]. One of the objectives of the current work is to solve such cases.

With the rise of advanced ML techniques, researchers have begun integrating these methodologies into the policy synthesis process. RL in particular, has shown significant promise in policy generation [12, 13, 45, 46, 62, 100], largely due to its adaptability and robust learning capabilities.

There is a growing body of work that applies Markov Decision Processes (MDPs) to account for the uncertainty in both the agent and the environment when solving LTL tasks [36, 50, 58, 59, 110]. MDPs provide a structured way to model decision-making in uncertain environments, capturing the complexities of both the agent’s behavior and the stochastic nature of the environment [28, 37]. However, solving LTL can be challenging, especially when transition probabilities are not known in advance and must be learned through interaction with the environment.

RL methods are particularly well-suited for this task, as they excel in scenarios where the MDP’s transition probabilities are unknown [96]. RL algorithms can effectively learn these probabilities through sampling and experience [109], adapting the agent’s policy over time to better satisfy the LTL specifications. This combination of MDPs and RL methods offers a powerful approach to handle the complexities and uncertainties inherent in LTL tasks, making it a promising area of exploration.

At the core of RL is the concept of the reward function, which is pivotal for guiding the learning process toward desirable behaviors. The integration of LTL into RL presents a promising approach to defining reward functions. Specifying LTL objectives as reward functions offers distinct advantages and challenges. The primary advantage lies in the precision of the reward specification. Typically, crafting a reward function that accurately encapsulates the desired requirements is challenging [4, 38]. Reward functions are often difficult to define because they must succinctly express complex objectives, and inadvertently overlooking certain aspects of a problem can lead to undesirable behaviors. However, by employing LTL as a means to defining rewards, it becomes possible to align the reward

mechanism closely with specific behavioral requirements [62]. This alignment ensures that, as long as the **LTL** specification is correctly formulated, the resultant behaviors of the system will closely match the intended outcomes.

Moreover, the use of **LTL** in **RL** leverages its strength in expressing temporal and logical constraints, providing a structured and formal method to encapsulate long-term goals and complex conditions [47]. This is particularly useful in environments where the sequence and timing of actions are crucial, and where traditional reward functions might fail to capture the nuances of such requirements [41]. However, this approach is not without its challenges. One of the primary challenges in utilizing **LTL** for reward function specification in **RL** is the precision required in writing **LTL** specifications [15]. Crafting accurate **LTL** formulations demands meticulous attention to details, as the omission or misplacement of even a single element can lead to specifications that fail to capture the intended objectives [3]. This sensitivity makes it imperative that specifications are developed with careful analysis to ensure that they truly reflect the desired system behaviors [70].

Another significant challenge when employing **LTL**-specified reward functions is the issue of sparse rewards [6, 69, 75]. In the context of **RL**, sparse rewards indicate that rewards are not distributed across all visited states within an episode but are awarded only at the end [95]. This reward structure can severely hinder the learning process and complicate the convergence of the learning algorithm. Because the agent receives feedback infrequently, it struggles to recognize which actions are beneficial towards achieving the long-term goal, thus prolonging the training phase and potentially leading to suboptimal policy development.

Another challenge in this domain is the substantial increase in the number of samples needed for accurate reward propagation and estimation of states, particularly over extended horizons [10, 67, 76, 114]. However, the exploration will focus on how a sampling method such as **Monte Carlo Tree Search** (**MCTS**) significantly enhances both the efficiency of policy learning and the performance of the learned policy. [113]. When combined with a guiding heuristic, such as a neural network, **MCTS** has proven capable of remarkable performance in numerous tasks [14]. Drawing inspiration from how AlphaGo Zero [90] employs the same technique, this thesis has crafted a solution for satisfying **LTL_f** objectives in even stochastic environments.

These challenges underscore the need for careful integration of **LTL** within **RL** frameworks, ensuring that **LTL** specifications are not only precise but also conducive to effective learning and policy synthesis.

1.3 Linear Temporal Logic over Finite Traces (LTL_f)

The semantics of LTL are built on the assumption that the system will generate an infinite sequence of states. However, in practice, this assumption often does not hold true. For physical systems like robots, resource limitations are a reality. That is why the current work focuses on a fragment of LTL which accounts for a finite duration for traces by default. By combining RL techniques with LTL_f policy synthesis, this work aims to address the challenge of developing robust policies capable of satisfying LTL specifications in finite-duration trajectory settings.

Time, in particular, serves as a valuable proxy for other limited resources, such as energy and equipment durability, and can be directly related to the battery life of a robot. This makes it a practical choice for modeling constraints imposed on the system. By limiting the maximum duration of a trajectory, the current work treats time as a limited resource in the policy synthesis process.

Most work in this domain traditionally assumes infinite time when devising solutions. However, as will be demonstrated in the results sections of this thesis, such methods exhibit lower success rates when applied within finite trajectory lengths. This discrepancy highlights the importance of integrating realistic constraints like time into the policy synthesis process to ensure that the resulting policies are not only effective in theory but also in actual operational environments.

Using LTL with infinite traces also introduces another issue. Evaluating a finite trace becomes challenging because LTL evaluation inherently assumes infinite traces. Consequently, in practical scenarios, it becomes difficult to calculate metrics such as success rates, necessitating the development of alternative metrics, such as accumulated rewards, to assess the performance of the solution. However, these alternative metrics do not directly convey how well the agent performs. Success rate, measured in percentage, provides a clear and intuitive understanding of performance, making it a more expressive and useful metric in the context of satisfying an LTL task.

1.4 AlphaGo Zero and LTL_f Synthesis

AlphaGo Zero [90] is an artificial intelligence program developed by DeepMind, which achieved significant acclaim by mastering the game of Go. Go is a two-player zero-sum board game, notable for its complex strategic elements and vast number of possible positions. The game is played on a 19x19 grid where players alternately place black or white

pieces on the board with the objective of claiming the most territory and capturing the opponent’s pieces. Go has historically been a challenging domain for [AI](#) due to the high degree of human intuition and in-game search it required, which AlphaGo Zero managed to navigate successfully through advanced [ML](#) techniques including deep neural networks and Monte Carlo Tree Search.

The problem setup for Go and our research share several similarities, including both being episodic and discrete, with outcomes known only at the completion of an episode. This aligns the structural framework of Go with the challenges addressed in synthesizing policies under [LTL](#) specifications. However, notable differences exist between the two (explained in section [4.1.1](#)). While AlphaGo Zero tackles a two-player competitive game, our setup is oriented around a single-player scenario. Even when considering a multi-agent setup, the interaction between the agents is not competitive like Go. This distinction introduces numerous implications, which are discussed in detail in section [4.1](#).

Furthermore, the inherent nature of the tasks within these domains differs significantly. In Go, players strategically place pieces on a board, aiming to surround more territory than the opponent. In contrast, robot navigation involves moving one or more agents across a static map to achieve specified goals. The contrasts between these setups and their implications for policy synthesis and system behavior are covered in section [4.1](#).

Contributions

The primary contribution of this thesis is a novel Monte Carlo-based [RL](#) methodology for generating trajectories that satisfy [LTL_f](#) specifications for systems described as [MDPs](#) within a designated number of timesteps where:

1. The method uses a generative model of the environment to produce [MCTS](#) samples but does not directly use the [MDP](#) transition probabilities in its learning process.
2. The two-player AlphaGo Zero framework is adapted to that of a single agent tasked with satisfying an [LTL_f](#) specification.
3. The [MCTS](#) execution is modified to account for the unique challenges involved in the task of satisfying [LTL_f](#) specifications in an [MDP](#).
4. Optionally, a reactive policy can be extracted from the learned [MCTS](#) statistics which allows for the execution of the policy with significantly less overhead.

Extensive simulations demonstrate that the proposed approach can successfully learn robust policies for scenarios characterized by complex formulas, even in stochastic environments where there is a high risk of specification violations. Crucially, this is achieved

without any prior knowledge of the transition probabilities of the [MDP](#) that defines the environment dynamics. Most importantly where existing state-of-the-art methods fail even in deterministic settings, the developed approach can generate trajectories that satisfy the specification fully in the deterministic setting, and with a high probability (reported as success rates as seen in section [5.2](#)) in stochastic settings.

These contributions underscore the potential of integrating Monte Carlo methods with [RL](#) techniques for enhancing policy synthesis for [LTL](#) specifications.

1.5 Outline of the Thesis

The outline of the thesis is structured as follows:

Chapter [2](#): This chapter reviews the previous work done in the domain of [LTL](#) policy synthesis. It first looks at the origin use cases of [LTL](#), then shifts to discuss traditional [LTL](#) task satisfaction methods. After going over some famous methods, it focuses particularly on works that integrate [LTL](#) and [RL](#), providing a comprehensive overview of the current state of research in this area. Having reviewed the work done in this domain along with their shortcomings, the necessity of the current work is explained in the end and its groundwork is described in the following chapter.

Chapter [3](#): This chapter introduces the necessary background knowledge required to understand the proposed approach. It details the fundamental concepts of [LTL](#), [RL](#), [MCTS](#), among others, and then presents the problem formulation. Once all the required components are defined, the proposed method can be explained.

Chapter [4](#): The proposed method is discussed in detail in this chapter. Beginning with the challenges of [LTL_f](#) objective satisfaction and how they are dealt with. The rest of the chapter explains the overall algorithm and the details of its components, elaborating on how the Monte Carlo-based [RL](#) approach is applied to satisfy [LTL_f](#) specifications.

Chapter [5](#): This chapter presents the results obtained from the extensive simulations of three case studies with multiple environments in both deterministic and stochastic settings. Simulations are conducted on various environment settings in terms of size and complexity to test the performance and robustness of the proposed method.

Chapter [6](#): This chapter starts by reviewing two lines of work for the future that could potentially extend the current work. Then the chapter discusses the implications of the results, interpreting their significance and limitation in various scenarios, and finally concludes the thesis.

Chapter 2

Literature Review

[LTL](#) [77] has been a cornerstone in formal specification languages since its inception, offering a robust framework for defining system behavior over time [21, 23, 30]. Researchers have continuously explored methodologies to satisfy [LTL](#) specifications, reflecting the language’s importance and versatility in various domains. This chapter aims to survey the landscape of these methodologies, highlighting both historical approaches and contemporary advances. The objective is to go over what has been accomplished and to identify promising directions that sparked this thesis, and possible future research.

Initially, this review will explore the diverse applications of [LTL](#), illustrating its utility in specifying complex system behaviors across different domains. Following this, the discussion will pivot to traditional approaches for [LTL](#) task satisfaction, outlining the standard techniques and their limitations. As the chapter progresses, the integration of [MCTS](#) with neural networks will be discussed. This combination has gained popularity in recent years due to its effectiveness in handling complex decision-making tasks and large space states that require balancing exploration and exploitation [112]. Finally, and most critically, this chapter will delve into research that specifically employs [RL](#) for synthesizing policies that satisfy [LTL](#) specifications. This segment will not only cover the technical methodologies but also discuss the nuances and challenges encountered in this specialized intersection of [LTL](#) and [RL](#), and how the current work addresses them. By providing a comprehensive overview of these areas, this chapter sets the foundation for understanding the current state of research and the novel contributions of this thesis in leveraging [RL](#) and [LTL](#) for effective policy synthesis.

2.1 Requirement Specification and Verification via **LTL**

In the year 1977, the foundational paper [77] introduced **LTL** for specifying and verifying the correctness of programs, marking the beginning of **LTL**'s application in system specification. As we move forward in time, [48] provides an extensive overview of using **LTL** and other formal methods for the specification, verification, and analysis of real-time systems, offering practical insights into handling timing constraints. Furthermore, [11] delves into the nuances of real-time systems, arguing that not all Temporal Logics are apt for these systems due to their unique constraints and requirements. It introduces a set of criteria designed to assess the suitability of various temporal logics for the specification, validation, and verification of real-time systems. Other notable works in this domain include [11, 34, 99].

The paper [18], investigates the application of propositional **LTL** as a specification language for planning problems. They argue that **LTL** is well-suited for these purposes due to its rich expressive power and its straightforward model of time and actions, which can accurately capture the dynamics of planning scenarios. The paper demonstrates how different planning problems can be effectively encoded, showing **LTL**'s capability to represent complex planning tasks in a structured and clear manner.

LTL is a popular formal method for system verification. Examples in this domain include air traffic control, autopilot, airplane separation, CPU design, automated warehousing, automotive safety and many more. [84] is a survey that goes over a comprehensive list of aerospace systems where **LTL** is used to define the working conditions of the systems.

In the discussion of reactive systems, which include entities like operating systems, process control programs, and seat reservation systems, [78] offers a comprehensive overview of how temporal logic is applied to their specification and verification. These systems are characterized by their ongoing operations—they are not designed to terminate but instead continuously interact with and respond to their environment. Real-time systems are examples of reactive systems and [2, 53, 71, 73] explore the use of formal methods and **LTL** in particular for system's safety verification. The primary goal of using temporal logic in this context is to ensure that throughout their perpetual operation, these systems behave correctly and maintain the desired interaction with their environment

2.2 Traditional **LTL** Task Satisfaction Methods

Traditional methods for satisfying Linear **LTL** tasks primarily involve formal verification techniques that ensure a system's behavior conforms to specifications.

Model checking [21] is a formal verification technique used to systematically check whether a finite-state model of a system meets certain specifications or properties. It involves a thorough exploration of all possible states of a system to verify whether these states adhere to the given specifications [19]. **LTL** is often employed with model checking to specify properties of systems, particularly those involving sequences of events that unfold over time [22]. **LTL** allows for expressing complex temporal behaviors, such as safety (something bad never happens), liveness (something good eventually happens), and fairness conditions, in a formal and precise manner [91]. As mentioned above, [84] discusses many aerospace systems where **LTL** specifications are analyzed via model checking.

An example work that utilizes an **automata-based approach** for solving **LTL** tasks is [92] which discusses methods for translating **LTL** formulas into efficient Büchi automata. Büchi automata are a type of automaton used for modeling systems that perform infinite sequences of actions [16]. Büchi automata are widely used in formal verification to check whether the behaviors of such systems satisfy specifications expressed in temporal logics like **LTL** [85, 87, 101].

Other works explore **synthesis from LTL specification** directly. As an example [65] presents an approach for synthesizing reactive systems directly from **LTL** specifications using the Spot [29] framework, a library for **LTL** formulas manipulation and automata-based operations. The authors demonstrate how to convert **LTL** formulas into deterministic automata that can then be used to synthesize controllers or software that automatically satisfies the given **LTL** specifications. Another similar example is [80].

Runtime verification is another technique used in the verification of systems, where **LTL** can be used to define the desired requirements. [8, 9, 42] discuss the use of **LTL** for runtime verification, presenting techniques for monitoring running systems to check compliance with **LTL** specifications. They highlight the use of **LTL** as a specification language in runtime verification frameworks and demonstrate how these can be effectively applied to ensure system reliability in real-time.

The work [60], introduces the Temporal Logic of Actions (TLA), an extension of **LTL** that is specifically tailored for specifying and reasoning about concurrent systems. While TLA itself is a broader framework, the methodology it employs is deeply rooted in the principles of **theorem proving** for temporal logic [81]. The paper lays out a comprehensive method for using TLA to specify system behaviors and subsequently prove properties about these behaviors using theorem-proving techniques. Other works like [52] use algorithmic approaches for verifying properties specified in **LTL**.

2.3 MCTS and Neural Networks

The combination of Monte Carlo Tree Search (MCTS) with Neural Networks (NNs), often abbreviated as MCTS+NN, has gained significant popularity in the field of AI, particularly following the groundbreaking success of AlphaGo [89]. MCTS excels in exploring potential future states, while neural networks provide accurate evaluations of those states, improving the overall efficiency and effectiveness of the search process [33]. This integration not only enhances the speed and accuracy of decision-making but also significantly boosts the system’s ability to generalize across different scenarios, making MCTS+NN a highly effective tool in advancing AI applications [72]. Here, a brief discussion will cover some of the work that applies the same principle in solving similar problems.

Two-player games like Go: [31] and their subsequent work [32] managed to apply the MCTS+NN strategy to the game of Hex, a two-player zero-sum game that involves players placing colored pieces on a hexagonal grid to connect opposite board edges. This approach led them to surpass the then state-of-the-art solutions. Similarly, [115] employed MCTS+NN in the game of Dots and Boxes, where players draw lines to form boxes, achieving superior performance. [20] also used this strategy for Othello, noting a slight improvement in performance against the previous best solutions. These examples underscore the effectiveness of MCTS+NN in various two-player games, enhancing strategic depth and decision-making prowess beyond conventional methods. However, it’s also valuable to explore scenarios where the problem does not conform to a two-player game format, highlighting the versatility and potential limitations of the MCTS+NN approach.

The Binpacking Problem: In their study, [61] applied the same MCTS+NN approach to the NP-hard problem of bin packing, specifically aiming to minimize the surface area used in a single bin. This adaptation showcases the versatility of the MCTS+NN combination beyond multiplayer scenarios, as they developed a near-optimal solution for this complex single-player problem. The paper highlights that their method outperforms several alternatives, including a solely neural network-based agent, a supervised learning agent, and a traditional MCTS agent with Monte Carlo rollouts for state-value estimation. Adapting MCTS+NN to such problems demonstrates the challenges and innovations required in tailoring these two components with modified algorithms, parameters and cost functions to fit specific single-player contexts.

Neural Architecture Search (NAS): Authors of [108] introduce AlphaX, a neural architecture search engine that leverages the MCTS+NN framework to optimize network architectures for specific tasks. By exploring the design space through actions such as adding layers and blocks, AlphaX efficiently identifies promising architectures based on

model accuracy. Drawing on methodologies akin to AlphaGo Zero, the MCTS component evaluates each architecture by training and assessing its performance on the designated task. AlphaX demonstrates a significant improvement in sample efficiency—being three times more efficient than random search and 1.5 times more than ϵ -greedy based Q-learning. Additionally, its distributed search capability greatly accelerates the architectural optimization process, showcasing the adaptability and effectiveness of MCTS+NN in automating and enhancing neural network design.

2.4 LTL and Policy Synthesis via RL

Policy synthesis for systems with LTL specifications has been extensively explored in research, especially in recent years. This section focuses on methods that utilize RL to address LTL objectives. RL algorithms have demonstrated their efficacy in solving a broad spectrum of complex problems across domains such as control, robotics, and gaming. In these applications, the behavior of an agent is dictated by a policy, which is essentially a mapping from states to actions. The primary goal in RL is to discover an optimal policy that maximizes the expected cumulative reward.

Model-Free Q-Learning for LTL Specifications: In [12], the authors approach the synthesis of control policies for LTL-specified problems using a model-free Q-learning framework. The strategy involves constructing a product MDP by integrating the environmental MDP with a Limit Deterministic Büchi Automata (LDBA) derived from the LTL formula. Details of these constructs are elaborated in the preliminary chapter 3. Through Q-learning, the optimal Q action values are determined, focusing on maximizing the likelihood of satisfying the LTL specifications over the long term. However, as noted in section 5.2, this approach, while effective in many scenarios, does not always succeed in finding policies that satisfy certain specifications, such as those in section 5.2.1, where the proposed approach in this thesis provides a solution.

Game-Theoretic Approach in Model-Based RL: [13] frames the satisfaction of LTL formulas as a two-player zero-sum game between the agent and a dynamic environment, contrasting with the static or probabilistic environments considered in this work. The methodology employs minimax-Q learning, operating within a product game construction. This construction includes the deterministic Rabin Automaton (DRA) of the LTL formula combined with the environmental MDP. However, in this setting the environment can dynamically react to the agent’s actions, which is not the case in the current work where only a non-reactive (but possibly probabilistic) environment is considered.

Both [12] and [13] base their reward propagation on the Bellman equation, which can result in slow reward distribution if the product system either lacks sufficient states offering positive rewards or if such states are rarely reached, as seen for the sequential delivery case study with multiple unsafe zones in Sec. 5.2.1. Additionally, similar to other studies in this field, both papers assume an infinite time horizon for addressing their tasks, which tends to yield lower success rates in finite scenarios as highlighted by the results in this thesis. Such scenarios demonstrate the limitations of traditional approaches when adapted to tasks with bounded durations, underscoring the contributions of the current work.

Reward Machines for LTL Tasks: Addressing the challenge of sparse rewards in this domain, [45] introduced the concept of **Reward Machines (RMs)**, which was further developed in [46]. These **RMs** form the foundation of their proposed **Q-learning for Reward Machines (QRM)** framework. **RMs** provide a structured way to map states and actions to rewards, making it easier to handle complex specifications and collect rewards for each step in an environment that would otherwise result in sparse rewards.

Despite their effectiveness in mitigating the sparse rewards problem, **RMs** come with certain limitations that have not been addressed in these works. Primarily, the construction of **RMs** is a manual process that requires expert knowledge to build appropriate machines for specific **LTL** tasks. This necessity for expertise can be a barrier to scalability and flexibility in diverse applications. The current thesis work addresses this limitation by utilizing product systems that can be automatically constructed without requiring expert knowledge, enhancing the applicability and accessibility of the approach. Moreover, the **QRM** framework does not naturally generalize to stochastic environments for the same problem it was originally designed to solve in a deterministic setting. This shortcoming highlights the need for approaches that can adapt to varying degrees of environmental uncertainty and dynamics without the need of hand-crafting new reward structures.

Additionally, the original frameworks proposed in [45] and [46] do not inherently take time constraints into account, which can be a problem in real-world scenarios where decisions must be made within finite time frames.

Generalization in LTL Policy Synthesis: The study conducted by [100] represents a significant advancement in the field of **LTL** policy synthesis by leveraging the compositional syntax and semantics of **LTL** to learn policies that can generalize to specifications unseen during training. This is achieved through the innovative use of **Graph Neural Networks (GNNs)**, which encode **LTL** formulas and facilitate the learning process across varied and new (unseen) **LTL** tasks.

Their framework offers a robust approach to handle dynamically generated **LTL** specifications, showcasing the potential of neural architectures in extending the applicability of

policy synthesis methods. However, despite these advances, the study restricts itself to a context-free fragment of **LTL**, which is notably less expressive than **LTL_f** which is used in this work (see section 3.2 for more details). This limitation is evident in their case studies where complex tasks such as the office-world task discussed in Section 5.1 of this thesis, cannot be adequately represented within their system’s capabilities.

Geometric LTL for Finite Time Steps: In addressing the complexities associated with loop-requiring specifications (objectives which necessitate the system to repeatedly visit the same set of states) within finite time steps, [64] introduces a novel task-specification language known as Geometric **LTL**. it integrates the concept of time windows, providing a structured method to manage tasks that traditionally require looping behaviors to be expressed within a finite sequence of actions. Geometric **LTL** is designed to simplify the specification and verification of behaviors that must conclude within specific time frames, making it well-suited for environments where traditional **LTL** might struggle to provide clear guidance due to its infinite horizon assumptions. Despite its promising framework, the application of Geometric **LTL** in the studies conducted by [64] is limited to relatively simple scenarios. The case studies presented in their research involve environments that are significantly less complex than those considered in this thesis, both in terms of environmental complexity (size, number of items, and levels of stochasticity) and the richness of the **LTL** specifications used.

Truncated Linear Temporal Logic: The study by [62] introduces Truncated Linear Temporal Logic (TLTL) as a novel specification language specifically tailored for robotic applications. TLTL is designed to provide a more practical framework for specifying and evaluating the performance of robotic tasks, which often require precise control and timing. This specification language allows for the expression of time-bounded properties, making it highly suitable for scenarios where tasks must be completed within a specific timeframe.

To optimize policies under TLTL specifications, [62] employs **Relative Entropy Policy Search (REPS)**, a method known for its effectiveness in policy refinement and exploration. **REPS** is utilized to maximize a robustness degree introduced by the authors, which quantifies how well an agent fulfills the TLTL specifications. This robustness degree serves as a crucial metric, guiding the policy search towards solutions that not only meet the specified requirements but do so in a manner that enhances the overall reliability and performance of the robotic system.

The efficacy of this approach is demonstrated through evaluations in both simulated and real-world environments, specifically in the context of robot arm manipulation tasks. By assessing the robustness degree, [62] showcases the success of their methodology, highlighting its potential to achieve high precision and reliability in robotic operations. The

integration of TLTL with advanced policy search methods like REPS offers a promising direction for further research and application, particularly in complex robotic systems where timing and precision are paramount.

Consideration of MDP Costs in LTL Task Solutions: In a deviation from the approaches typically discussed, the research presented in [105] introduces a method that incorporates the MDP costs associated with the movement of a robot into the solutions for LTL tasks. This focus on cost optimization aligns with the theme of resource limitation, similar to the assumptions about limited resources integral to the current thesis work. By factoring in the costs of actions within the MDP, their approach strives to achieve a more holistic optimization, balancing task satisfaction with cost efficiency.

However, their methodology requires prior knowledge of a lower bound on the transition probabilities of the underlying MDP. This requirement can restrict the applicability of their approach in environments where such probabilistic bounds are unknown or difficult to estimate. In contrast, the approach developed in this thesis does not depend on such specific information.

Trajectory Transformers for LTL_f -Specified Tasks: recently, [97] introduced the use of trajectory transformers for achieving tasks specified by LTL_f in mini-grid environments. This method employs DFA to manage the issue of sparse rewards, which is a common challenge in RL-based methods dealing with logical specifications.

The use of trajectory transformers represents a significant step forward in the integration of advanced ML techniques with formal methods for policy synthesis. By mapping LTL_f specifications to DFA, the approach ensures that the rewards are distributed in a manner that directly supports the satisfaction of the specified tasks, thereby enhancing learning efficiency and policy effectiveness.

However, while the methodology proposed by [97] is promising, the complexity of the LTL_f specifications and the environmental challenges addressed in their work are relatively modest compared to those examined in this thesis. The mini-grid environments and LTL_f tasks they tested involve simpler dynamics and less stochasticity than those considered here. Such simplifications may limit the direct applicability of their findings to more complex or realistic scenarios, where the environmental unpredictability and specification complexity significantly increase. The work undertaken in this thesis provides a more rigorous test of policy synthesis under conditions that better mirror real-world applications.

Goal-conditioned RL for LTL objectives: In the work [79], the authors propose a framework for optimally solving LTL objectives, even in previously unseen cases. The framework consists of two key phases. First, they introduce the concept of a Task-Augmented MDP, where the agent is assigned a single goal (state) to reach in the envi-

ronment. The agent is trained on simple goals, such as moving from one label to another, covering all possible movements in the environment. Through this training, the agent acquires the basic knowledge necessary to achieve these simple goals. In the second phase, the Büchi automaton of the **LTL** specification is converted into a graph, and Dijkstra’s algorithm is used to find the shortest paths to an accepting state. This path provides the sequence of labels to be visited but not the method to achieve them. By combining the knowledge gained in the first phase with the sequence of labels determined in the second phase, the agent is able to solve the task. Notably, this approach extends to new **LTL** tasks; as long as the agent knows how to navigate the environment to reach simple goals, it can solve unseen tasks by computing the required sequence of labels. A drawback of the approach in [79] is that the number of labels in the task specification leads to an exponential increase in the number of goal value functions the agent must learn during the first phase. In contrast, our approach is not affected in this manner by the number of labels in the specification.

2.5 Proposed Model-Free **RL** Method for **LTL_f** Specifications

Building on the insights gained from previous research and avoiding the limitations observed in the discussed approaches, this thesis proposes a novel model-free **RL** method that is capable of handling any **LTL_f** specification, while not requiring prior knowledge of the **MDP** transition probabilities.

The versatility of the proposed **RL** approach is demonstrated through extensive simulations, which show that it can scale effectively to a diverse range of complex problems. These simulations provide evidence that the method can learn robust policies across various tasks, even in environments characterized by significant stochasticity. This shows not only the adaptability of the method to different situations but also its capability to maintain high levels of policy robustness, ensuring reliable task completion under diverse conditions.

Before delving into the formal problem formulation and exploring how the current work addresses this challenge, it is essential to first establish a solid foundation through a review of relevant preliminaries in the next chapter. This preparatory step will cover the fundamental and theoretical concepts that are critical for understanding and developing the proposed method. By examining these key preliminaries, it is ensured that the methodologies and solutions presented later are well understood and clear for an effective discussion of the problem-solving techniques employed in this thesis.

Chapter 3

Preliminaries

This chapter discusses the foundational elements of our problem setup and subsequently provides a formal definition of the problem formulation. By laying out the core concepts and theoretical frameworks first, a clear understanding of the background necessary for comprehending the complexity of the issues addressed is aimed to be established.

3.1 Markov Decision Processes (MDPs):

The environments discussed throughout this work, more specifically the grid-world scenarios, are modeled as [MDPs](#). In these models, the agent does not directly utilize the transition probabilities or the expected rewards associated with the states of the MDP to make decisions during training or testing, including the online deployment of the policy. An [MDP](#) is formally defined as a tuple $\langle S, s_0, A, P, R \rangle$ where:

- S is a finite set of states,
- s_0 is the initial state (or a probability distribution over states),
- A is a finite set of actions,
- $P : S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, satisfying the property $\sum_{s' \in S} P(s' | s, a) = 1$ for all $s \in S$ and $a \in A$,
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function.

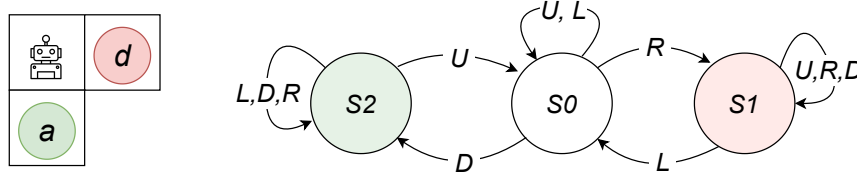


Figure 3.1: A simple 3-cell environment and its MDP. Actions are going up, right, down and left and are shown on the arrows in the MDP.

Fig. 3.1 server as an illustrative example of a 3-cell environment modeled as an MDP. Transition probabilities of this system are implicitly shown as arrows, where each arrow corresponds to a change of state with probability 1. If there is no arrow for an action between two states, the probability of that transition is 0.

MDPs are used to keep track of the location of the agent in the environment, while its inner state with regard to the specification at hand is tracked using another construct called **Limit Deterministic Büchi Automata (LDBA)** (explained in the rest of this chapter). Modeling grid-worlds as MDPs is advantageous because it simplifies the representation of the environment's dynamics, where the outcome of actions in each state follows Markovian properties. This ensures that the future state depends only on the current state and the executed action, not on the sequence of events that preceded it.

Given a set of *atomic propositions* AP , where each member represents a statement about the state of the system (for example being at a tree location in Example 1.1.1), an MDP could be augmented by a *labeling function* $L : L_{ap}(s) \rightarrow \{True, False\}$. This labeling function maps each atomic proposition ap to a boolean value for an input state s of the MDP, which specifies whether or not that atomic proposition holds in the given state. For example, assuming the atomic proposition *wood* in Example 1.1.1 stands for being at a tree location, in that case, $wood = True$ when the agent is at any cell with a tree, and $wood = False$ otherwise. By adding L and AP to the MDP it becomes a *labeled MDP*. The labeling function is essential for evaluating LTL_f specifications over trajectories, i.e., sequences of MDP states. A trajectory of visited (labeled) MDP states can be evaluated via an LTL_f specification by checking the sequence of observed labels (true or false) for each AP against the LTL_f formula by examining the sequence of labels (true or false) for each atomic proposition along the trajectory to determine if the observed behavior satisfies the specified temporal logic requirements. More detail on the evaluation of a trajectory over LTL_f formulae is provided in the next section.

This work does not require the MDP transition probabilities for solving the problem

directly but needs them to run simulations during **MCTS** rollouts (see section 4.3). Unlike common model-based **RL** methods that utilize transition probabilities to update parameters, transition probabilities are only used here to simulate the environment and gather samples. This eliminates the need to know the transition probabilities in advance, offering flexibility when dealing with environments with unknown or complex dynamics. This distinction allows the benefits of sample-based planning without explicitly learning the transition model.

3.2 Linear Temporal Logic over Finite Traces (**LTL_f**):

LTL_f [25] is a variant of the formal language **LTL** [77] used to express temporal constraints for a syste. From a syntactic point of view, **LTL_f** and **LTL** are similar. The only difference is in their semantics and how they are interpreted over a given trajectory. **LTL** assumes the trajectory to be infinite, while **LTL_f** takes a finite trajectory [26].

Syntax of LTL:

$$\begin{aligned} \varphi ::= & \text{true} \mid \text{false} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2 \\ & \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U}\varphi_2 \mid \varphi_1 \mathbf{R}\varphi_2 \end{aligned}$$

Where:

- p is an atomic proposition from a set of atomic propositions AP .
- $\neg\varphi$ is the negation of formula φ .
- $\varphi_1 \wedge \varphi_2$ is the conjunction (and) of formulas φ_1 and φ_2 .
- $\varphi_1 \vee \varphi_2$ is the disjunction (or) of formulas φ_1 and φ_2 .
- $\varphi_1 \rightarrow \varphi_2$ is the implication (if...then) where φ_1 implies φ_2 .
- $\varphi_1 \leftrightarrow \varphi_2$ is the equivalence (if and only if) between φ_1 and φ_2 .
- $\bigcirc\varphi$ is the next operator, which means φ holds in the next state.
- $\Diamond\varphi$ is the eventually operator, which means φ will hold at some future state.

- $\Box\varphi$ is the globally operator, which means φ holds in all future states.
- $\varphi_1\mathcal{U}\varphi_2$ is the until operator, which means φ_1 holds until φ_2 holds.
- $\varphi_1\mathcal{R}\varphi_2$ is the release operator, which means φ_1 releases φ_2 .

3.2.1 LTL_f Semantics

The semantics of LTL_f are defined in terms of interpretations over finite traces, representing sequences of states over a finite set of time instants [25]. These interpretations are modeled as finite words π over the alphabet of 2^P , where P is the set of atomic propositions, where each member represents a statement about the state of the system.

The following notation is used:

- The length of a trace π is denoted as $\text{length}(\pi)$.
- Positions, or instants, on the trace are denoted as $\pi(i)$ for $0 \leq i \leq \text{last}$, where $\text{last} = \text{length}(\pi) - 1$.
- Segments of the trace from position i to j are denoted as $\pi(i, j)$, where $0 \leq i \leq j \leq \text{last}$.

Given an LTL_f interpretation π , the truth of an LTL_f formula φ at an instant i (for $0 \leq i \leq \text{last}$) is defined inductively as follows:

- $\pi, i \models A$ for $A \in P$ iff $A \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$.
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$.
- $\pi, i \models \bigcirc\varphi$ iff $i < \text{last}$ and $\pi, i + 1 \models \varphi$.
- $\pi, i \models \varphi_1\mathcal{U}\varphi_2$ iff there exists j such that $i \leq j \leq \text{last}$, $\pi, j \models \varphi_2$, and for all k where $i \leq k < j$, $\pi, k \models \varphi_1$.

A formula φ is true in a trace π , written $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula φ is satisfiable if there exists an LTL_f interpretation π such that $\pi \models \varphi$. Logical implication and validity follow from these definitions [25].

In this work, the terms *LTL formula*, *specification*, *task*, and *objective* are used interchangeably, but they all refer to the same concept: an LTL formula that defines the requirement specification of the desired behavior.

LTL_f is designed to be evaluated over finite trajectories, while focusing on the logical ordering of events rather than time constraints. To explicitly include time in an **LTL_f** task, a central clock would need to be introduced, incrementing with each time step. This would require defining a new atomic proposition that signals when the clock exceeds a specified time limit. However, incorporating such time-tracking mechanisms is not typical in **LTL_f** task specifications, as **LTL_f** is primarily concerned with event sequences, leaving time constraints to be handled by other parts of the solution.

3.2.2 Evaluating Trajectories of Labeled MDP States

When an agent navigates through the environment, it traverses a sequence of labeled MDP states. At these labeled states, each atomic proposition (events possible in the environment) is realized as a Boolean value. Evaluating these sequences against an LTL_f formula involves a recursive process.

Given an **LTL** formula φ and a trajectory $\sigma = s_0, s_1, \dots, s_{N-1}$ comprising N visited states of an MDP, the sequence of evaluations $e_0, e_1, \dots, e_{N-1}$ is computed recursively using the MDP's labeling function $L : L_{ap}(s) = \text{True/False}$, which maps an atomic proposition ap to a Boolean value for a state s of the **MDP**.

The recursive evaluation is defined as follows:

$$E(\varphi, \sigma, L) := \{e_0, e_1, \dots, e_n\} \text{ where each } e_i \text{ is evaluated based on the following cases:}$$

$$\begin{cases} \text{Return } L_{\varphi}(\sigma_i), & \text{if } \varphi \text{ is a label} \\ \text{Evaluate each operand } Op \text{ in } \varphi \text{ as } E(Op, \sigma, L) \text{ then apply } \varphi, & \text{if } \varphi \text{ contains an operation} \end{cases}$$

Each e_i in the sequence indicates whether φ holds from the i^{th} state onward. For instance, an evaluation sequence $\phi = \{\text{False}, \text{True}, \text{True}\}$ indicates that the trajectory satisfies the LTL_f specification starting from the second state.

To determine if the entire trajectory adheres to the [LTL](#) formula, one needs to inspect the first element in the sequence, $E_0(\varphi, \sigma, L) = e_0$. If e_0 is True, the trajectory fully complies with the formula; if False, it indicates a violation at some point in the trajectory.

It is worth noting that the length of the evaluation sequence may not necessarily match the length of the trajectory. This discrepancy arises because certain temporal operations, such as the Next operation ($\bigcirc$), require information from future states to evaluate the current state.

3.3 Limit Deterministic Büchi Automata ([LDBA](#)):

[Limit Deterministic Büchi Automata \(LDBA\)](#), a variant of deterministic Büchi automata, plays a crucial role in formal verification and model checking, particularly due to its ability to express properties involving repeated patterns of behavior. [LDBA](#) extend standard deterministic Büchi automata by incorporating a concept of limit, which enables them to handle properties that require recurring behavior patterns effectively.

Translating an [LTL](#) formula into an [LDBA](#) can be systematically achieved through techniques outlined in significant works such as [39, 88]. These translations are key to leveraging the strengths of [LDBA](#) in model checking, allowing for a more robust evaluation of system behaviors against the specified LTL properties.

The evaluation of a trajectory against an [LDBA](#) involves checking if the sequence of states produced by the trajectory satisfies the [LDBA](#)'s acceptance condition. This acceptance condition, characterized as a Büchi condition, specifies the infinite sequences (or trajectories) that are accepted by the [LDBA](#), thereby making the verification of compliance with the LTL specifications more feasible.

Formal Definition: An [LDBA](#) can be formally defined as a tuple $(Q, q_0, \Sigma, \delta, F)$, where:

- Q is a finite set of states in the [LDBA](#),
- $q_0 \in Q$ is the initial state,
- Σ represents the alphabet used by the [LDBA](#),
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function dictating the state transitions,
- F is a set of acceptance states.

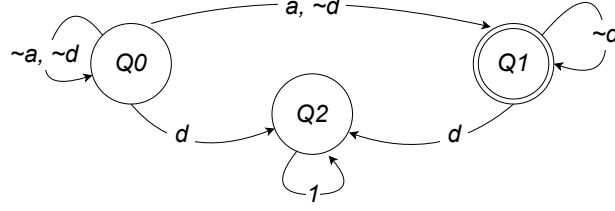


Figure 3.2: The **LDBA** of the **LTTL** formula: $\Box \neg d \wedge \Diamond a$. Double circles indicate accepting states. Arrow labels indicate input alphabets to the **LDBA**.

Derivation from LTL Formulas: The process of deriving an **LDBA** from an **LTL** formula involves constructing a deterministic framework that can capture the infinite behaviors specified by the LTL. This often requires the formula to be decomposed into simpler components that can be directly translated into states and transitions of the LDBA. Each component of the LTL formula corresponds to specific states and transitions in the LDBA, ensuring that every valid model of the formula is a valid trace in the LDBA, and vice versa.

Intuitively, **LDBAs** provide a structured way to evaluate how well a trajectory aligns with the specified **LTL** formula, thereby offering valuable insights into the system’s behavior in its operational environment. The ability to systematically translate LTL formulas into the framework of **LDBA** significantly enhances the applicability and effectiveness of formal methods in real-world scenarios, where ensuring adherence to behavioral specifications is critical. Further details of the **LDBA** structure are available in [88].

Fig. 3.3 server as an illustrative example for the **LTTL_f** formula: $\Box \neg d \wedge \Diamond a$. The formula asks to eventually reach a state with label a while avoiding the label d at all times.

In this work, a tool called Rabinizer 4 [56] is utilized. (<https://rabinizer.model.in.tum.de>), which is a toolkit for generating compact deterministic ω -automata from LTL formulas. Rabinizer 4 supports the translation of LTL into various types of automata, facilitating the automatic conversion of LTL formulas into **LDBA**.

3.4 The Product MDP

The product **MDP** is an essential construct in the synthesis of controllers from **LTTL_f** specifications, representing the Cartesian product of a **LDBA** and an **MDP**. This combination encapsulates states from both the **MDP** and the **LDBA**, thereby integrating the dynamic behavior of the **MDP** with the specification requirements encoded in the **LDBA**.

Given a labeled **MDP** and an LTL_f formula, along with its corresponding **LDBA**, the product **MDP** can be formally constructed as follows:

- $S^* = S \times Q$ - The set of all possible pairs of **MDP** states and **LDBA** states, where S is the set of states in the **MDP** and Q is the set of states in the **LDBA**.
- s_0^* - The initial state of the product **MDP**, typically a combination of the initial states of both the **MDP** and the **LDBA**.
- A^* - The action set of the **MDP**, potentially augmented with epsilon transitions from the **LDBA** to model silent moves that do not affect the state of the **MDP** but may transition the state of the **LDBA**.
- F^* - A set of accepting states within S^* , which are crucial for satisfying the acceptance conditions of the **LDBA**.
- P^* - The transition function mapping each pair of state $s_i^* = (s_i, q_i)$ in S^* to another pair $s_j^* = (s_j, q_j)$ based on the selected action a^* and the labeling function $L(s_i)$.

In the current approach, the exploration is conducted within the space of the product **MDP** rather than solely within the **MDP** state space. As the agent interacts with this augmented environment, it effectively navigates through both the state space of the original **MDP** and the specification constraints imposed by the **LDBA**. This dual exploration ensures that the learned policies not only aim to optimize the expected rewards but also conform to the behavioral norms dictated by the LTL formula. This setup allows for the straightforward extraction of corresponding labeled **MDP** states from a sequence of product **MDP** states, which are essential for evaluation purposes (as explained in section 3.2.2).

Fig. 3.4 server as an illustrative example of the product **MDP** of the **MDP** in Fig. 3.1 and the **LDBA** in Fig. 3.3.

In this work, the implementation of [12] is utilized for constructing the product **MDP** from a specified **MDP** (the map of the environment) and the LTL formula.

Many of the works in the domain of policy synthesis rely on a notion of a product **MDP** to solve their problems [12, 13, 105]. Although having access to the product **MDP** simplifies solving an LTL task, there is a significant challenge when the transition probabilities of the underlying **MDP** are unknown. Without access to these probabilities, finding the optimal policy becomes problematic. The transition dynamics of the environment, such as how the agent moves or how obstacles affect movement, are not directly encoded in the product **MDP**. This means that while algorithms like Dijkstra’s may compute an ”optimal” path

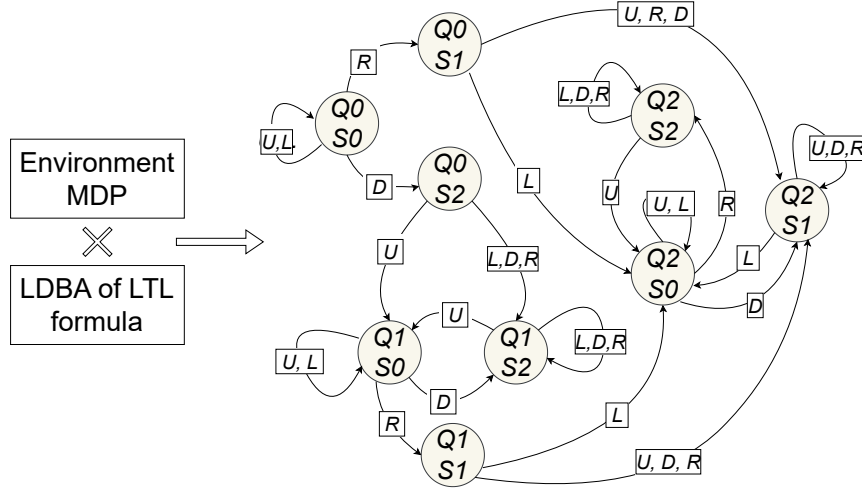


Figure 3.3: Illustrative example of the product **MDP** of the **LDBA** of the **LTL** specification $\Box \neg d \wedge \Diamond a$ corresponding to the example in Fig. 3.3, and the **MDP** in Fig. 3.1

based on the available structure, this path might not be feasible in the actual environment. For instance, without the transition probabilities, the product **MDP** might overlook critical details, such as obstacles, making it impossible to guarantee that the computed solution is truly applicable in practice. Therefore, learning the transition probabilities or estimating them is necessary to properly account for environmental constraints and dynamics.

Estimating transition probabilities in **MDPs** often requires a number of samples that grows exponentially with the size of the state-action space [24], particularly in environments with high stochasticity where there is a tradeoff between exploration and exploitation [93, 94]. As a result, learning or solving **MDPs** by estimating these transition probabilities is not generally feasible in polynomial time [51].

3.5 AlphaGo Zero And The Game of Go

AlphaGo Zero has been a groundbreaking advancement in the field of artificial intelligence. The game of Go, which is a two-player board game where the aim is to control more territory on a 19x19 grid (look at Fig. 3.4) by placing black or white stones, serves as the domain in which AlphaGo Zero has demonstrated its prowess.

From a black box perspective, evaluating a trajectory against a set of **LTL** specifications to obtain a Boolean reward can be viewed as a win/lose scenario, similar to the game of Go.



Figure 3.4: A game of Go being played on a grid board.

The resemblance becomes more profound considering that both involve discrete states and a discrete set of available actions, with moves occurring in episodic sequences. However, a key distinction lies in the absence of a second player in the [LTL](#) policy synthesis problem, necessitating a reformulation of the self-play setup utilized in AlphaGo Zero.

To adapt AlphaGo Zero’s methodology for [LTL](#) policy synthesis, a modified version of the original framework must be employed. This involves tailoring the self-play mechanism to operate within a single-agent context, where the agent’s objective is to satisfy the [LTL](#) specifications rather than competing against an opponent. This adaptation retains the core strengths of AlphaGo Zero, such as its ability to learn effective policies through its use of [Monte Carlo Tree Search \(MCTS\)](#) combined with [Neural Network \(NN\)](#) for state evaluation and action selection. The next chapter discusses the details of how the current work addresses these issues. However, before that, the problem to be solved can now be formally defined. This is essentially the same problem briefly discussed in the introduction (section [1.1](#)). Once the problem formulation is laid out, the proposed method to solve it will be reviewed. By leveraging these techniques, the modified AlphaGo Zero framework

provides a robust foundation for solving [LTL](#) policy synthesis problems.

3.6 Problem Formulation

Having discussed the fundamental concepts involved in the problem setup, the problem can now be formally defined:

Problem 1 *For an environment represented via a labeled [MDP](#) M with its labeling function L , and initial state s_0 , with a priori unknown transition probabilities, an [LTL_f](#) formula φ describing the task specification, and a time horizon T : find a sequence of labeled [MDP](#) states $\sigma = \{s_0, s_1, \dots, s_{T-1}\}$ such that their corresponding observed labels satisfies φ , or:*

$$\text{Generate } \sigma = \{s_0, s_1, \dots, s_{T-1}\} \text{ from } M \text{ such that } E_0(\varphi, \sigma, L) = \text{True} \quad (3.1)$$

Reminder that $E_0(\varphi, \sigma, L)$ was defined in section [3.2.2](#). Problem [1](#) is defined assuming that the environment in question is deterministic since achieving task satisfaction cannot be guaranteed in a stochastic environment. For stochastic environments, the problem formulation requires a modification. This modification would change the objective to the following:

Given all the assumptions in Problem [1](#) the goal is to: find a sequence of labeled [MDP](#) states $\sigma = \{s_0, s_1, \dots, s_{T-1}\}$ such that their corresponding observed labels maximize the probability of satisfying φ .

As discussed in section [3.1](#), in order to solve the above problem, the agent does not require any prior knowledge of the transition probabilities of the underlying [MDP](#) governing the environment, as long as it can interact with and obtain samples from the environment. In other words, the agent interacts with the environment dynamics as a black box, meaning it can run simulations and generate samples from the environment without needing explicit transition probabilities.

The following chapter explains how the proposed approach solves Problem [1](#). At the end of the chapter (section [4.4](#)), it will be shown that once such sequences are generated, a policy that significantly speeds up the sequence generation process can be obtained.

Chapter 4

Methodology

In this chapter, the details of the proposed method are explored. The differences between the game of Go and the current problem setup are discussed first. Following this, the issues arising from these differences are addressed, and an in-depth explanation of the proposed approach is provided.

4.1 From the Game of Go to LTL_f Tasks

Our proposed method is inspired by AlphaGo Zero [90], yet it diverges significantly because fulfilling an LTL_f task fundamentally differs from playing a game of Go. Here, the principal distinctions are outlined, and the adaptation of MCTS for specific needs is described. The details of the MCTS implementation are then discussed in the following sections.

Revisiting States Within a Run: A key difference between Go and the instances of Problem 1 is the uniqueness of state visits within each episode of Go. In Go, each state in a game is unique in that game, preventing repetitive state visits. Contrarily, in tasks like robot navigation, an agent may repeatedly encounter the same states. This tendency can cause issues as the agent might explore these repetitive states without progressing meaningfully within the product MDP state space.

AlphaGo Zero leverages a variant of the PUCT algorithm [83] to balance exploration and exploitation during MCTS search. However, with binary evaluations over finite traces, if MCTS fails to find a satisfactory trajectory after initial samples, revisiting identical states will lead to similar rewards to that of visiting new states that do not satisfy the specification. Therefore the MCTS search might get stuck in redundant and unproductive

explorations. In other words, it would not matter which states the agent explores, new or not, since it is receiving similar rewards for them. Early experiments demonstrated that after a few iterations **MCTS** tended to oscillate among a few repetitive states without gaining new insights, indicating a need for modifications to handle **LTL_f** tasks effectively.

Lack of a Second Player: Unlike Go, which is played within a self-play framework, our setup lacks an adversarial component where learning occurs from both winning and losing scenarios. In Go, if an agent loses, the opposing agent—which is itself in another instance—gains a win, providing a continuous learning opportunity. In our scenario, merely receiving zero rewards (i.e., failing to solve the task) offers limited guidance for achieving the **LTL_f** specifications.

The absence of a second player removes the natural foundation for self-play, which is a powerful tool in **RL**, particularly evident in two-player competitive games. In these settings, competition inherently drives the evolution of strategies, as each player continually adapts to outperform the other. However, in our single-agent setup, this competitive mechanism is absent, making it challenging for the agent to incrementally learn and improve its strategies. There is no ever-improving adversary to provide a benchmark against which the agent can measure and enhance its performance. This lack of competitive self-play means there is no dynamic, escalating standard of challenge, which can weaken the agent’s ability to discover and refine more effective strategies for fulfilling the task. Therefore the primary goal of the agent is to find at least one trajectory that satisfies the task specification.

4.1.1 Adapting **MCTS** for **LTL_f** Objectives

Here, modifications to **MCTS** are discussed to address these unique challenges:

1. **Leaf Node Encounters:** If the agent encounters a new state (leaf node) and has satisfied the specification at least once thus far during training, the network predicts a value. If not, it receives a baseline positive reward to encourage exploration of new states.
2. **Depth Limit Reached:** When the search depth limit is reached, the trajectory is evaluated. If it fulfills the specification, a positive reward, larger than the previous baseline value, is assigned; otherwise, it returns zero. This setup ensures that satisfying trajectories are prioritized and more likely to be revisited, enhancing the effectiveness of subsequent searches.

For the first point, if no satisfying trajectories have been encountered during training, the network lacks exposure to meaningful positive feedback, causing it to output arbitrary, uninformative values. Therefore, until a satisfactory trajectory is found, **MCTS** consistently returns a minor positive reward for new nodes, encouraging the exploration of unvisited states. Once a satisfying path is found, the algorithm consults the network for value predictions, improving search efficiency.

The second point underscores how **MCTS** values complete rollouts. The reward for finding a satisfactory rollout not only ensures that such paths are highly valued but also that they are more frequently sampled, addressing the challenge posed by the lack of competition in our problem setup.

4.1.2 Overall View of the Proposed Method

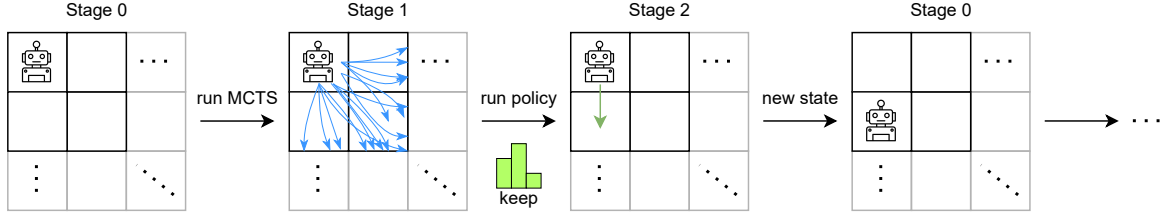


Figure 4.1: A single iteration of stages in Fig. 4.4 depicted for better visualization. At each step, the agent gathers many **MCTS** samples and chooses its next move from the result of these samples. This process continues until either the task is solved or time has run out.

As mentioned in the previous chapter, the **RL** framework in this work integrates Monte-Carlo rollouts (i.e., trajectory samples from a given state) with a neural network to enhance action selection, aiming to increase the likelihood of task satisfaction. Overall, the neural network guides the **MCTS** by pinpointing promising actions, thus reducing the need for extensive sampling. Conversely, **MCTS** helps train the neural network by refining policies for each state and providing the agent with better action choices at each state.

These components and their adaptations for solving Problem 1 are detailed in this chapter, with figures 4.1 and 4.4 depicting the proposed framework and Alg. 1 elaborating on the overall procedure in greater depth while Alg. 2 describes the details of stage ④ in Fig. 4.4 (the **MCTS** component).

In short, in each episode (one full run of the agent in the environment trying to satisfy the specification), the agent begins at a specific state of the grid-world (stage 0 in Fig. 4.4). It then collects many **MCTS** samples from that state (stage 1 in Fig. 4.4) and gets a policy that tells it what to do as its first action. Sampling from this policy, the agent makes its move in the environment (stage 2 in Fig. 4.4) and observes the next state. Having taken one step, the agent repeats the same procedure again until the end of the episode. Alg. 1 describes this procedure.

Algorithm 1 generating a trajectory to satisfy a **LTL_f** formula

Input: Trajectory duration T , The product **MDP**, **LTL_f** formula φ , labeling function L , the network $\text{net}(\theta)$, **MCTS** #samples S , and search_depth
Initialize Q, N, P, W , with zero, and with appropriate dimensions
Initialize σ, Π as empty lists ▷ Observed **MDP** states and their **MCTS** policies
Sample s_0 and append it to σ ▷ s_0^* is initialized based on s_0
for each $t \in \{0, \dots, T\}$ **do**
 $\pi_t \leftarrow \text{MCTS}(\text{net}(\theta), s_t^*, \varphi, Q, N, P, W, S, \text{search_depth})$
 $a_t^* \leftarrow \text{sample from } \pi_t$
 $s_{t+1}^* \leftarrow \text{observe from product MDP transition}$ ▷ Next product **MDP** state
 append s_{t+1} to σ , and π_t to Π
end for
Calculate r_{LTL_f} by evaluating σ against φ

- s_t^* is the product **MDP** state of the t^{th} step, while s_t is the corresponding **MDP** state

The details of the parameters and components in Alg. 1 are now explained, with a focus on how they work, presented in two sections for the neural network and **MCTS**.

4.2 Neural Network for Policy and Value Function Estimation

The actions selected by **MCTS** during its rollouts are informed by a neural network. This network processes an encoded product **MDP** state, $s^* \in S^*$, which it receives as a multi-channelled matrix (explained in section 4.2.1) It then computes through several Conv2D and Dense layers (explained in section 4.2.2) to produce:

1. A probability distribution $\mathbf{p}$ over the action set, serving as the *prior* policy for nodes yet to be explored in [MCTS](#).
2. A real value v , indicating the probability of task satisfaction starting from state s^* .

Contrary to AlphaGo Zero, our network does not utilize a history of visited states for its inputs. The network’s loss function, adapted from [\[90\]](#), for input s^* is given by:

$$L(\text{encoded}(s^*)) = (r_{\text{LTL}_f} - v)^2 - \pi^T \log \mathbf{p} + c\|\theta\|^2 \quad (4.1)$$

In this equation, $v \in (0, +1)$ represents the network’s predicted value, and $\mathbf{p} \in \mathbb{R}^{\dim(A^*)}$ is the suggested policy. $\pi : S^* \rightarrow A^*$ is the improved [MCTS](#) policy, detailed in the subsequent subsection. r_{LTL_f} is the Boolean reward $E_0(\varphi, \sigma, L)$ for the entire episode σ (corresponding to stage ③ in Fig. [4.4](#)), and the final term, $c\|\theta\|^2$, is the L2 regularization for the network parameters θ , where c controls the regularization strength during the training of network weights.

4.2.1 Network Input Format

The input to the network should be the product [MDP](#) state of the desired state. A product [MDP](#) incorporates information about the state of both the [MDP](#) and the [LDBA](#). This information can be represented in a channeled format of the grid world.

Assuming the robot is navigating a grid world of shape $m \times n$, this grid world can be represented as a matrix of shape $m \times n$, where each element encodes the position of the robot, labels of cells (e.g., $a, b, c, \dots$), and the existence of obstacles using numbers. Empty cells can be represented by zeros. A simple example is shown in Fig. [4.2](#), illustrating the grid world and its encoding.

To incorporate [LDBA](#) information into this single matrix, it can be augmented with zero matrices $k - 1$ times, where k is the total number of [LDBA](#) states. The state of the [LDBA](#) can then be represented by placing the non-zero matrix (the encoded [MDP](#) state) in the i -th channel, where i corresponds to the current [LDBA](#) state among all k states. The figure below also depicts a channeled version of a grid world with k [LDBA](#) states, assuming the robot is in the i -th state.

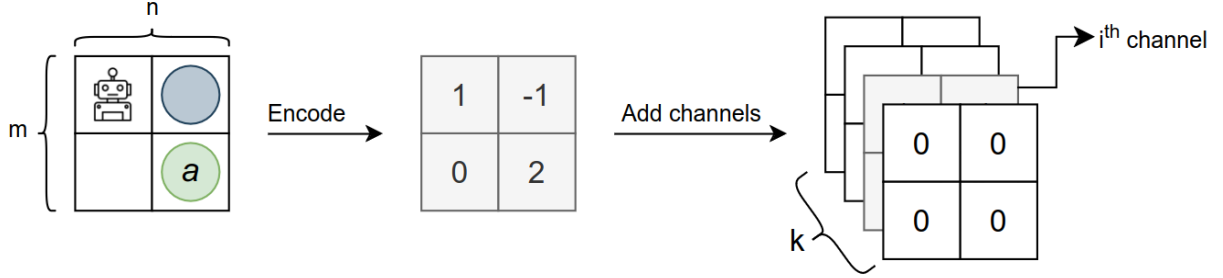


Figure 4.2: Illustrative example of the encoding of an [MDP](#) state for a simple 4-celled environment, and its channeled format for the network

4.2.2 Network Architecture

The architecture of the network in this work is as follows. The input to the network, as discussed above, is a matrix of shape (m, n, k) , where m and n are the number of rows and columns of the grid world, and k is the number of [LDBA](#) states. The two subsequent layers are Conv2D layers with 32 and 8 kernels of shape (2,2) respectively, each with one zero padding, and Rectified Linear Unit (RELU) activation functions. After flattening the resulting $(m, n, 8)$ matrix into a vector of size $(n \times m \times 8)$, two consecutive Dense layers are applied with 32 and 16 outputs respectively, both using RELU activation functions. The final 16-element vector is then processed by:

1. A Dense layer of size l , where l is the number of possible [MDP](#) + [LDBA](#) actions, with a softmax activation function to produce the action probability distribution (the policy).
2. A Dense layer of size one, with a Sigmoid activation function to give the predicted success probability as a real number $\in (0, 1)$.

The loss function for these two outputs is categorical crossentropy for the policy and mean squared error (MSE) for the success probability (based on equation 4.1). The optimizer used for the network is Adam, with Keras’s default parameters for its implementation. See the following plot (Fig. 4.3) for an example of the neural network architecture for a 10×10 grid-world when there are 7 LDBA states.

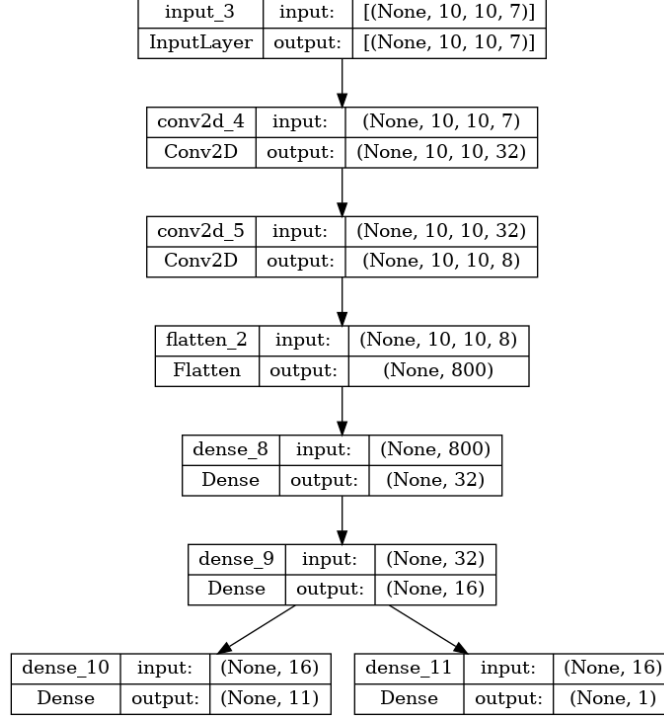


Figure 4.3: Example network architecture for a 10×10 grid-world, with a hypothetical LTL_f specification that results in 7 **LDBA** states

4.3 **MCTS** for Policy Improvement

In this work, the agent and its internal **MCTS** explore the same product **MDP** state space. **MCTS** maintains a tree where nodes represent the product **MDP** states, and edges denote possible transitions to subsequent states.

The results of **MCTS** rollouts from each state s^* are stored in its corresponding node within a set of parameters. These parameters include:

- $N(s^*)$: visit counts of the node's children during the rollouts,
- $W(s^*)$: accumulated backed-up rewards through the node's children,
- $Q(s^*) = W(s^*)/N(s^*)$: average backed-up rewards through the node's children, and
- $P(s^*)$: the *prior* policy for the node (used later on).

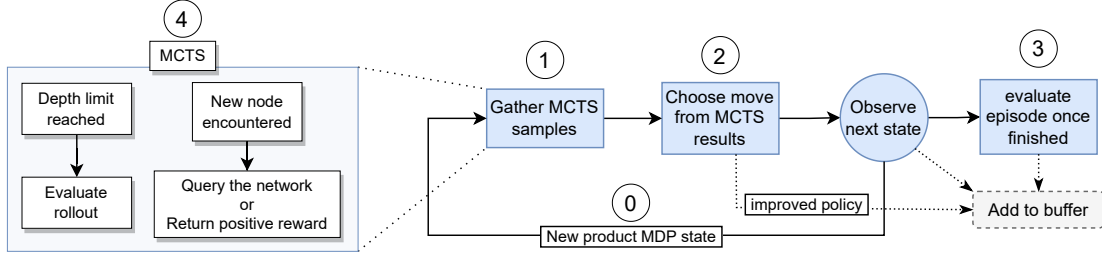


Figure 4.4: The overall procedure of how the proposed method works. At each state, actions are sampled from the improved policies gathered via [MCTS](#), then used to train the network which in turn helps the [MCTS](#) with better rollouts later on.

Whenever [MCTS](#) encounters a new node, which is called a leaf node, it queries the network to obtain this prior policy, possibly along with a value (see Sec. [4.1.1](#)), and updates the rest of the parameters as it gathers more samples.

Assuming a particular state s_t^* (reminder, superscript $*$ means s is a product [MDP](#) state and the subscript t indicates the t^{th} time step), according to Alg. [1](#), the agent starts gathering [MCTS](#) samples. These samples traverse the same product [MDP](#) state space as the one the agent is exploring while actually performing moves in the environment. If the agent were deployed in a real-world scenario, you can think of the [MCTS](#) samples as simulations conducted internally by the robot to help it decide on the best course of action.

As the agent is in state s_t^* , it starts collecting these samples from the same s_t^* , which is the root of the search tree (refer to Fig. [4.5](#)). [MCTS](#) actions are selected based on the following equation [\[83\]](#):

$$\text{Move} \leftarrow \text{Argmax}(U[\text{root}] + Q[\text{root}]) \quad ; \quad U = C \cdot \frac{P[\text{root}]}{1 + N[\text{root}]} \sqrt{N[\text{root}]} \quad (4.2)$$

The different parts of this equation can now be examined to understand how exploration of the state-space works and why modifications are needed. [MCTS](#) chooses its moves at each node in the tree based on the argmax over the $U + Q$ vector. The argmax operation selects the action with the highest $U[\text{root}][a] + Q[\text{root}][a]$. These two vectors adjust the balance between exploration and exploitation. U is responsible for exploration, while Q focuses on reselecting actions that have yielded the best results so far. As observed, Q is the average of the returned values throughout the [MCTS](#).

Examining the equation for U reveals the following insights:

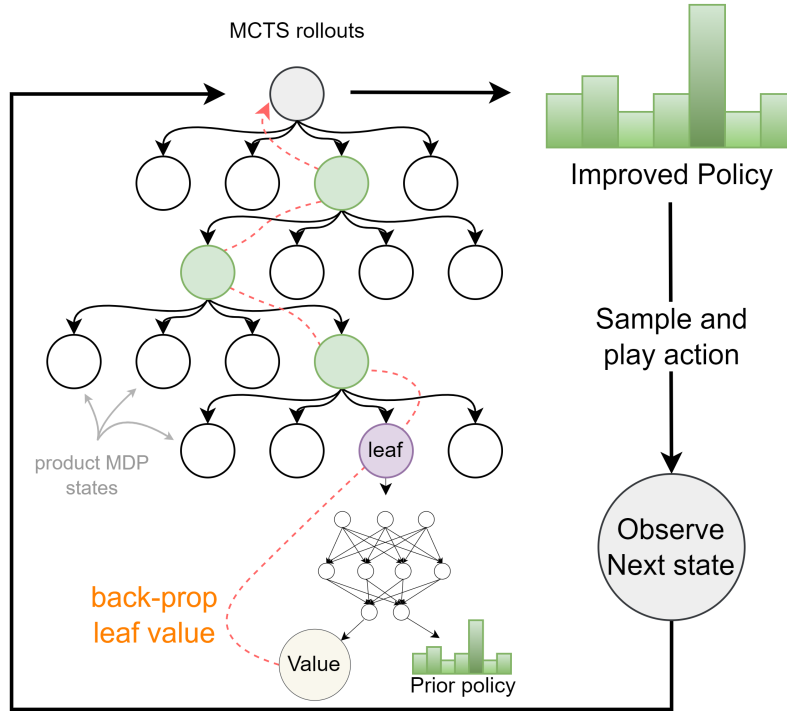


Figure 4.5: Illustration of how **MCTS** works. On the left a single rollout has unfolded, while on the right the result of many such rollouts has resulted in a probability distribution over the next immediate actions, i.e., a policy.

- C is a constant that defines the importance placed on U overall. Generally, a higher constant value results in greater exploration and slower convergence of the number of **MCTS** visits to nodes.
- P is the prior probability for a node, as output by the network when the node was first expanded by **MCTS**.
- N is the number of times the node has been visited throughout **MCTS**.

This framework is designed so that as more samples are gathered, the agent increasingly relies on the Q values (exploitation). This reliance is reflected in the node visit counts N . These visit counts will converge as **MCTS** selects moves with higher expected values, which leads to more focused exploration and exploitation. The converging visit counts are also the same parameters used for extracting the policy for each state.

However, as the next section (4.1) of this chapter explains, the MCTS framework employed is different than the one used in AlphaGo Zero. Here, some of the reward functions tested for the problem setup are discussed.

The following (Alg. 2) describes the MCTS algorithm used in this work for a single rollout on the product MDP space.

Algorithm 2 A single MCTS rollout

Input: N, Q, W, P , Trajectory σ , LTL formula φ , labeling function L , the network $net(\theta)$, list of *visited*, nodes, *search_depth*, constant C , NN_output_active
if $\text{len}(\sigma) \geq \text{search_depth}$ **then**
 return $E_0(\varphi, \sigma, L)$
end if
if *root* not in *visited* **then**
 add *root* to *visited*
 $P[\text{root}] \leftarrow net(\text{root})_{PolicyOutput}$
 if !NN_output_active **then**
 return 0.6 ▷ $0 < \text{positive value} < 1$
 end if
 $value \leftarrow net(\text{root})_{ValueOutput}$
 return *value*
end if
 $U[i] = C \times P[\text{root}][i] \times (\sqrt{N[\text{root}][i]}) / (1 + N[\text{root}][i])$ ▷ for each available action i
 $Move \leftarrow \text{Argmax}(U + Q[\text{root}])$
Observe $root_{new}$ based on *Move*
append new MDP state of $root_{new}$ to σ
 $value \leftarrow \text{MCTS}(root_{new}, \sigma, \dots, \text{search_depth} + 1)$ ▷ recursion
 $N[\text{root}][\text{move}] + = 1$
 $W[\text{root}][\text{move}] + = value$
 $Q[\text{root}][\text{move}] \leftarrow W[\text{root}][\text{move}] / N[\text{root}][\text{move}]$
return *value*

Once a number of samples are collected, the policy is calculated based on Eq. 4.3:

$$\text{policy } \pi = (N[\text{root}]^{1/\tau}) / \sum N[\text{root}]^{1/\tau} \quad (4.3)$$

Where τ is a hyperparameter to adjust how deterministic the resulting policy should be (higher means more deterministic). Different values for τ and C (see Alg. 2) were tested, and final chosen values were $\tau = 1, C = 3$

As described in Alg. 1, at each step of the episode with the agent in state s_t^* (stage ① in Fig. 4.4), S number of MCTS rollouts are performed (stage ①), resulting in a probability distribution over the possible actions (i.e., a policy) π_t for that state. This policy is typically superior to the prior policy suggested by the network for s_t^* , so the agent samples from this policy at the t^{th} step to select an action (stage ②). The chosen action is then stored in a buffer as the target for training the network’s policy output for s_t^* based on Equation 4.1.

4.3.1 Reward Function Engineering

This subsection describes experiments with various reward functions and explains why they did not yield satisfactory results. The section begins with the default AlphaGo Zero framework.

AlphaGo Zero’s default framework: In this framework, each episode unfolds to completion, and the game (or run of the experiment in our case) is evaluated in its entirety. The network is queried whenever MCTS visits a new node during its search. It is important to reiterate that using the same reward propagation framework as AlphaGo Zero does not encourage exploration, as previously discussed.

LDBA rewards: A second reward mechanism that was tested was to use the LDBA rewards. Although LDBA state rewards were not discussed in our setup and when defining the LDBA since they are not used in the proposed method, they were considered at the beginning of this research. The LDBA reward of a state q can be defined as $\text{Reward}(q) = 0$ if $q \notin F$ and a positive number if $q \in F$. By associating the LDBA reward of a product MDP state to it, the problem can be reframed within a standard RL algorithm, where the rewards of desirable states propagate through the product MDP state space. However, this framework does not work well with our problem due to the way LDBA transitions operate.

To transition to an accepting state, the LDBA must follow an ordering of MDP and LDBA ϵ -actions. In particular, the LDBA must first choose the ϵ -action and then visit the appropriate labels in the MDP. The ϵ -actions do not advance the robot’s position in the MDP but only change its internal LDBA state. Therefore, from an external perspective, the order of the LDBA and MDP actions should not matter, but it does from the LDBA perspective. This means there could be trajectories that satisfy the specification and produce a positive reward at the end of the episode, but their history of observed rewards would be all zeros. This confuses the agent because, in MCTS, it is not being rewarded for certain trajectories, yet the network is being trained to favor these trajectories based on the episode’s final outcome.

Combining LTL evaluations with accumulating LDBA rewards: To counter the above issue, I developed a reward mechanism that incorporates both the LDBA and LTL evaluations when assigning rewards to MCTS rollouts and completed episodes. The following reward functions were introduced for this purpose. For each rollout of the MCTS, if the resulting rollout did not trigger a safety violation or end up at an explored state, and observed at least one positive reward in the path, the scalar reward of that MCTS rollout is calculated as follows:

$$\text{Rew}_{\text{MCTS}} = R_{\text{acc}} \left(\frac{1}{R_{\text{mc}}} + R_{\text{mc}} r_{T-1} \right) \quad (4.4)$$

where R_{acc} is the summation of all the rewards observed in that rollout:

$$R_{\text{acc}} = \sum_{n=0}^{T-1} r_i \quad (4.5)$$

R_{mc} is the coefficient used in MCTS rewards.

For evaluating a complete episode, a sequence of rewards was generated for each visited product MDP state as follows:

$$\text{Rew}_{\text{ep}} = R_{\text{ep}} r_{T-1} \vec{1} + [r_0, r_1, \dots, r_{T-1}] \quad (4.6)$$

where R_{ep} is a coefficient for determining the importance of ending up in an accepting state, and $\vec{1}$ is a vector of size T where each element is 1. Rew_{ep} effectively adds a scalar (the product of R_{ep} and r_{T-1}) to all elements in the sequence of observed rewards.

The intuition behind this reward function was to enable the agent to handle infinite interpretations of a specification (in LTL) that would allow it to satisfy specifications requiring loop behaviors (such as indefinitely visiting two states). However, this reward mechanism suffered in these scenarios and resulted in lower success rates compared to the final version of our reward mechanism. In the end, the procedure discussed in Section 4.1.1 outperformed all other tested methods.

4.4 Training the Network and Extracting a Policy

So far in this chapter, the conduct of MCTS rollouts and the collection of results for each state during a run have been explored. At the end of the run, the sequence of visited product MDP states is evaluated using the task specification (as described in Section

3.2.2), and the resulting Boolean r_{LTL_f} is recorded as the outcome or reward of that run. In summary, after each run, two key pieces of information are generated and kept in a buffer:

- A sequence of improved policies.
- The result of the run (r_{LTL_f} in Alg. 1).

The latter is then repeated for all visited product MDP states so that the value target for the network for each state is the result of the game.

During training, generated trajectories (sequences of $\{s_0^*, s_1^*, \dots\}$ in Alg. 1) along with their rewards (r_{LTL_f}) and sequences of MCTS policies (Π in Alg. 1) are stored in a buffer. The network’s weights are then updated on a number of random samples after each run.

Due to the computational intensity of MCTS samples, Alg. 1 can be slow at runtime. To address this, a simple method is proposed to extract a policy from the MCTS parameters after training using equation 4.7.

$$\pi(s^*) = \operatorname{argmax}_a N(s^*, a) \quad (4.7)$$

Training stops once the agent can fully solve the task from all valid initial states or after a maximum number of iterations. Deploying the agent with this policy is computationally much more efficient, enabling us to test our method across a wide range of simulation-based experiments, as presented in the next chapter.

Chapter 5

Simulation Studies

In this section, the proposed approach, specifically the extracted policy from Equation 4.7 following the agent’s training via Alg. 1, is evaluated and compared against a state-of-the-art Q-learning based method known as CSRL [12] across three distinct case studies.

Additionally, the performance of another method, Q-learning with counterfactual experiences for RMs (CRM) [46], is included for comparison within the relevant case study. Initial experiments demonstrated that the extracted policy (Equation 4.7) achieves performance very close to that of Alg. 1 after training, while offering significantly greater computational efficiency.

All experimental evaluations were conducted on a desktop computer featuring an AMD Ryzen 7 6800H processor (3.2 GHz, 8-core, 16-thread), an RTX 3070ti GPU, and 16 GB of RAM. The implementation of the proposed approach and the comparative methods was carried out using Python. The complete source code is available at <https://github.com/CL2-UWaterloo/RL-LTL>.

5.1 Environment Dynamics

The dynamics of the environments are consistent across all case studies. The robot has four MDP actions to choose from: moving up, right, down, or left. The robot may or may not have access to LDBA ϵ -actions, depending on the structure of the LDBA. Taking an ϵ -action only changes the LDBA state of the product MDP, without altering the robot’s location in the environment (i.e., the MDP state).

The environments can be deterministic or stochastic. In the deterministic case, the robot always moves in the desired direction. In the stochastic environment, the robot has a probability p of moving in the intended direction. For the experiments, p was fixed at 0.8, meaning there is an 80% chance the robot will move in the desired direction. There is a 10% chance it will move to the left and a 10% chance it will move to the right of the intended direction. In general, when there is a p chance of performing the intended action, there is a $1 - p$ chance that a random move to the left or right might happen.

Upon collision with a wall (outer layer of the environment) or an obstacle (represented as grey areas), the robot simply remains in its location, rendering the movement ineffective.

As a reminder, the dynamics of the environment carry over to the **MCTS** component. In other words, The **MCTS** component of the algorithm generates samples by simulating possible trajectories within the environment, utilizing the generative model to explore different states and outcomes. However, the learning component of the approach, which involves training the neural network and updating the policy, does not explicitly rely on knowing the exact transition probabilities between states. This characteristic makes the approach semi-model-free within the context of **RL**, as it combines model-based sampling for exploration with model-free learning for policy improvement.

Success rates, defined as the probability of satisfying the task starting from a random initial position, are reported for the evaluations. All reported success rates are measured by running 1000 runs from 10 independent experiments. The robot did not start any runs from cells where it was impossible to satisfy the formula.

Next, the case studies are discussed.

5.1.1 Case 1 - Sequential Delivery with Varying Number of Unsafe Zones

This case study involves a robot navigating an environment with the goal of completing a sequential delivery task. The environment, as depicted in Fig. 5.1a, can contain up to six unsafe zones labeled as d , as shown in Fig. 5.1c. The task requires the robot to satisfy the following **LTL_f** formula:

$$\varphi_{\text{sequential delivery}} = \Box \neg d \wedge (\neg c \mathcal{U} b) \wedge (\neg b \mathcal{U} a) \wedge \Diamond \Box c \quad (5.1)$$

This formula specifies that the robot must:

- Avoid unsafe zones d at all times.

- First visit location a , then location b , and finally location c , strictly in this order.

The agent must adhere to this sequence without visiting these locations in any other order; failure to do so results in an unsuccessful task. As the number of unsafe zones increases, the difficulty of the task escalates due to the higher risk of violating the safety constraints. This makes the sequential delivery task a challenging benchmark for evaluating the robustness of the proposed approach.

In environments with a higher number of unsafe zones, the likelihood of the agent accidentally entering an unsafe zone increases, presenting a significant challenge for the agent to navigate safely while adhering to the required sequence of locations. As demonstrated in Section 5.2, when the number of unsafe zones exceeds a certain threshold, the Q-learning method (CSRL) is unable to find a feasible solution to the problem, highlighting the limitations of traditional approaches under stringent safety constraints.

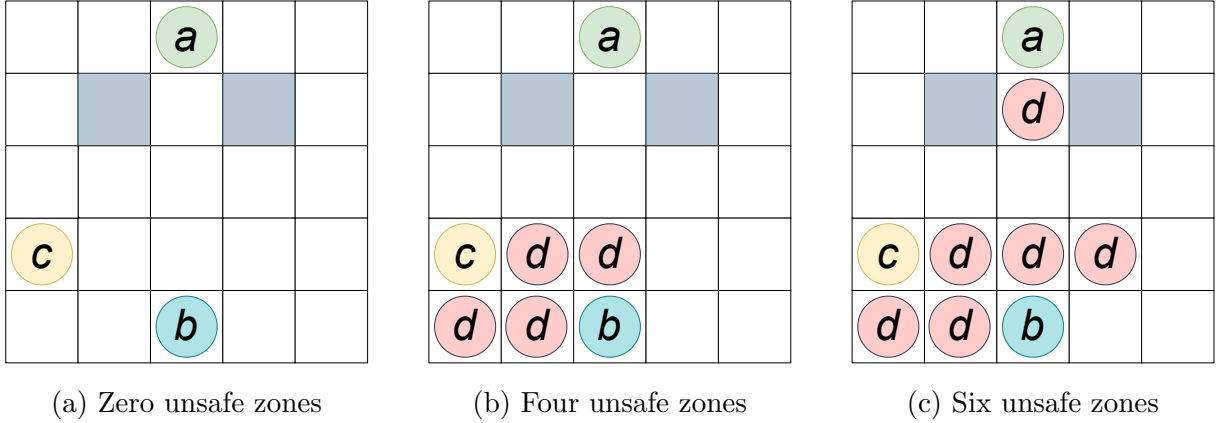


Figure 5.1: Environments for the sequential delivery case (Fig. 5.1a), with up to six unsafe zones (Fig. 5.1c).

5.1.2 Case 2 - Craft-world

The Craft-world environment (see Fig. 5.2), includes ten distinct tasks borrowed from [5] that have been adapted here as LTL_f objectives. Each task in the Craft-world environment involves navigating a grid-world to collect specific resources and achieve predefined goals, represented by LTL_f formulas below. The LTL_f objectives for these tasks are designed to test the agent’s ability to plan and execute complex sequences of actions in a large and

complex environment. For instance, tasks may involve collecting resources in a particular order or avoiding certain areas while gathering items needed to craft tools or complete objectives. In this study, the agent is required to complete these tasks without the special *use* action that was available in the original implementation by [5].

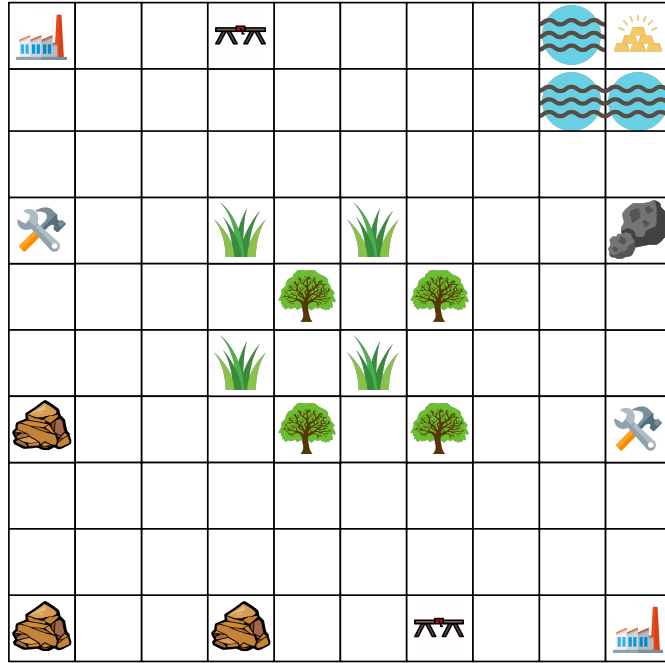


Figure 5.2: The 10×10 craft-world, as implemented for this work. The tasks to carry out for this environment were borrowed from [5].

The following are the ten tasks with their respective LTL_f formulas:

$$\varphi_{\text{make plank}}^{(1)} = \Diamond(\text{tree} \wedge \Diamond \text{toolshed}) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.2)$$

In this task, the agent must first reach a tree and then visit the toolshed to make a plank, while avoiding the river unless a bridge is available.

$$\mathcal{L}_{\text{make stick}}^{(2)} = \Diamond(\text{tree} \wedge \Diamond \text{workbench}) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.3)$$

For this task, the agent needs to collect wood from a tree and then use the workbench to make a stick, ensuring to avoid the river unless a bridge is present.

$$\varphi_{\text{make cloth}}^{(3)} = \Diamond(\text{grass} \wedge \Diamond \text{factory}) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.4)$$

Here, the agent must gather grass and subsequently visit the factory to produce cloth, while avoiding the river unless a bridge is accessible.

$$\varphi_{\text{make rope}}^{(4)} = \Diamond(\text{grass} \wedge \Diamond \text{toolshed}) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.5)$$

In this task, the agent is required to collect grass and then visit the toolshed to make rope, ensuring to avoid the river unless a bridge is available.

$$\begin{aligned} \varphi_{\text{make bridge}}^{(5)} = & \Diamond((\text{tree} \wedge \Diamond(\text{iron} \wedge \Diamond \text{factory})) \vee (\text{iron} \wedge \Diamond(\text{tree} \wedge \Diamond \text{factory}))) \\ & \wedge \Box(\neg \text{river} \vee \text{bridge}) \end{aligned} \quad (5.6)$$

For this task, the agent must either collect wood from a tree and then iron to visit the factory, or collect iron first and then wood to visit the factory to make a bridge, all while avoiding the river unless a bridge is accessible.

$$\varphi_{\text{make bed}}^{(6)} = \Diamond(\text{tree} \wedge \Diamond(\text{toolshed} \wedge \Diamond(\text{grass} \wedge \Diamond \text{workbench}))) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.7)$$

In this task, the agent must collect wood from a tree, visit the toolshed, gather grass, and finally use the workbench to make a bed, while avoiding the river unless a bridge is present.

$$\varphi_{\text{make axe}}^{(7)} = \Diamond(\text{tree} \wedge \Diamond(\text{workbench} \wedge \Diamond(\text{iron} \wedge \Diamond \text{toolshed}))) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.8)$$

For this task, the agent needs to collect wood from a tree, use the workbench, gather iron, and visit the toolshed to make an axe, ensuring to avoid the river unless a bridge is available.

$$\varphi_{\text{make shears}}^{(8)} = \Diamond(\text{tree} \wedge \Diamond(\text{workbench} \wedge \Diamond(\text{iron} \wedge \Diamond \text{workbench}))) \wedge \Box(\neg \text{river} \vee \text{bridge}) \quad (5.9)$$

In this task, the agent must collect wood from a tree, use the workbench, gather iron, and then use the workbench again to make shears, while avoiding the river unless a bridge is present.

$$\begin{aligned} \varphi_{\text{get gold}}^{(9)} = & \Diamond((\text{iron} \wedge \Diamond(\text{wood} \wedge \Diamond(\text{factory} \wedge \Diamond \text{gold}))) \vee \\ & (\text{wood} \wedge \Diamond(\text{iron} \wedge \Diamond(\text{factory} \wedge \Diamond \text{gold})))) \wedge \Box(\neg \text{river} \vee \text{bridge}) \end{aligned} \quad (5.10)$$

For this task, the agent must either collect iron first and then wood to visit the factory and get gold, or collect wood first and then iron to visit the factory and get gold, all while avoiding the river unless a bridge is available.

$$\begin{aligned} \varphi_{\text{get gem}}^{(10)} = & \Diamond(\text{tree} \wedge \Diamond(\text{workbench} \wedge \Diamond(\text{iron} \wedge \Diamond(\text{toolshed} \wedge \Diamond \text{ore})))) \\ & \wedge \Box(\neg \text{river} \vee \text{bridge}) \end{aligned} \quad (5.11)$$

In this task, the agent must collect wood from a tree, use the workbench, gather iron, visit the toolshed, and finally collect ore to get a gem, while avoiding the river unless a bridge is present.

This case study evaluates the effectiveness of the proposed approach in handling multiple, varied tasks within the Craft-world environment, demonstrating the agent's capability to generalize across different LTL_f specifications.

5.1.3 Case 3 - Office World

In this case study from [46], the robot must navigate within the office world environment depicted in Fig. 5.3. The objective is to satisfy the following LTL_f formula:

$$\varphi_{\text{office-world}} = \Box \neg d \wedge \Diamond((m \wedge \bigcirc \Diamond(c \wedge \bigcirc \Diamond o)) \vee (c \wedge \bigcirc \Diamond(m \wedge \bigcirc \Diamond o))) \quad (5.12)$$

In plain English, this formula dictates that the robot must avoid the decoration locations (d) at all times and eventually reach the main office (o) after visiting the coffee location (c) and the mail location (m) in any order. The office-world environment, as implemented in [46], includes additional cells on each axis to simulate walls. The hardest task defined in their study, represented by the formula above (Eq. 5.12), is used for this case study.

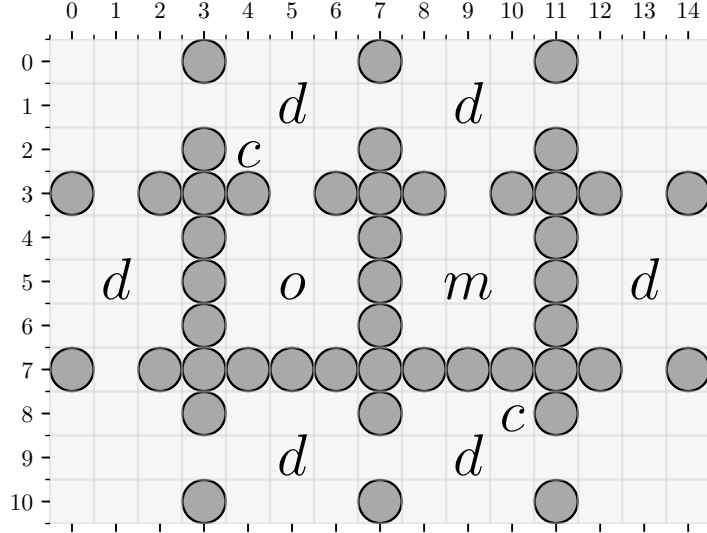


Figure 5.3: Environment for the office-world (see Sec. 5.1.3).

5.2 Simulation Results and Discussion

In this section, the results of the experiments conducted to evaluate the proposed approach are presented and discussed. The performance of the extracted policy is compared to state-of-the-art Q-learning based methods across various case studies. Each subsection details the findings for specific scenarios, highlighting the strengths and limitations of the approaches tested.

5.2.1 Sequential Delivery with Varying Number of Unsafe Zones

In the deterministic setting, the proposed approach successfully solves the task in all seven environments within 30 steps. As a reference, the minimum number of steps required to solve the task from any starting position is 25. The Q-learning baseline (CSRL) manages to solve the task only in the first four environments (up to three unsafe zones) but fails to find a solution in cases with four or more unsafe zones, regardless of the trajectory length.

In the stochastic setting, the average success rates are presented in Fig. 5.4, where the agent’s trajectory length is fixed at 50 steps. In this comparison, CSRL was trained for three million iterations for each environment, allowing for two orders of magnitude more training time than our approach requires to solve the problems.

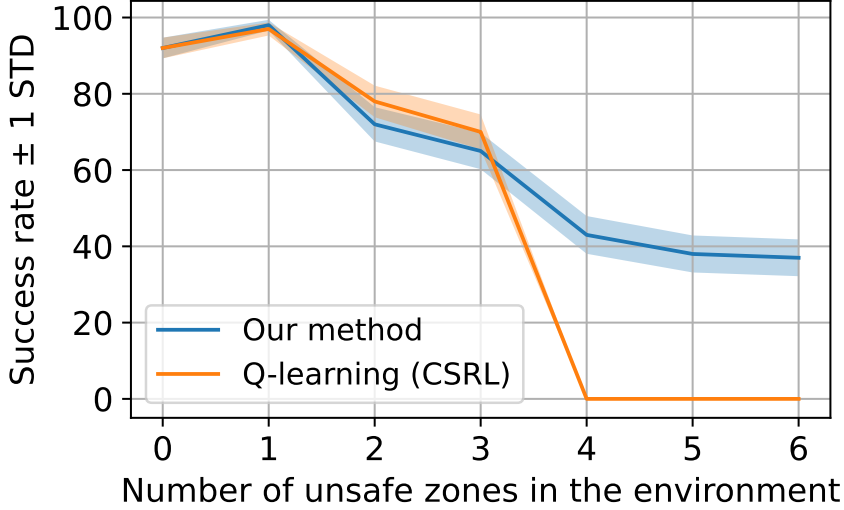


Figure 5.4: Results for the sequential delivery case study in stochastic environments.

The inability of CSRL to solve three of the environments is likely due to the difficulty of reaching the [LDBA](#) accepting states. In the sequential delivery case, the [LDBA](#) generates very sparse rewards. Upon closely examining the [LDBA](#), it becomes evident that there is only one state providing positive rewards. In contrast, due to the high number of unsafe zones, there are numerous death states, which are states from which the agent cannot recover. This imbalance means that while the accepting state is unlikely to be discovered, the death states are much more likely to be visited. As a result, no positive reward is propagated through the [LDBA](#), hindering the Q-learning process as it relies on the propagation of positive rewards from these accepting states. One potential way to address this issue could be to assign negative rewards to these death states, but the effect of such an approach remains to be explored.

On the other hand, [MCTS](#) coupled with a neural network can still find satisfying trajectories in such challenging scenarios. The combination of [MCTS](#) with a neural network helps overcome the sparsity of rewards by effectively exploring the state space and identifying trajectories that satisfy the [LTL_f](#) specifications.

5.2.2 The Craft-World Tasks

In the deterministic setting, the proposed approach successfully solves all ten tasks within 40 steps from any starting position. As a reference, the minimum number of steps required to solve the hardest task from any starting position is 29. The CSRL method cannot be tested on the ninth task (Eq. 5.10) due to its implementation constraints which do not allow the agent to know when a bridge is made. It is also not possible to encode this requirement in the $LT\mathcal{L}_f$ specification in a way that would account for *if* statement that states the bridge is made if its materials are gathered and brought to the correct location. However, it successfully solves the remaining nine tasks within 40 steps.

In the stochastic environments, the proposed approach eventually achieves perfect success rates given sufficient time steps, and CSRL does too for nine of the tasks (aside from the one that it cannot be tested on). The following covers the results obtained from all ten tasks, including the one where CSRL could not be tested.

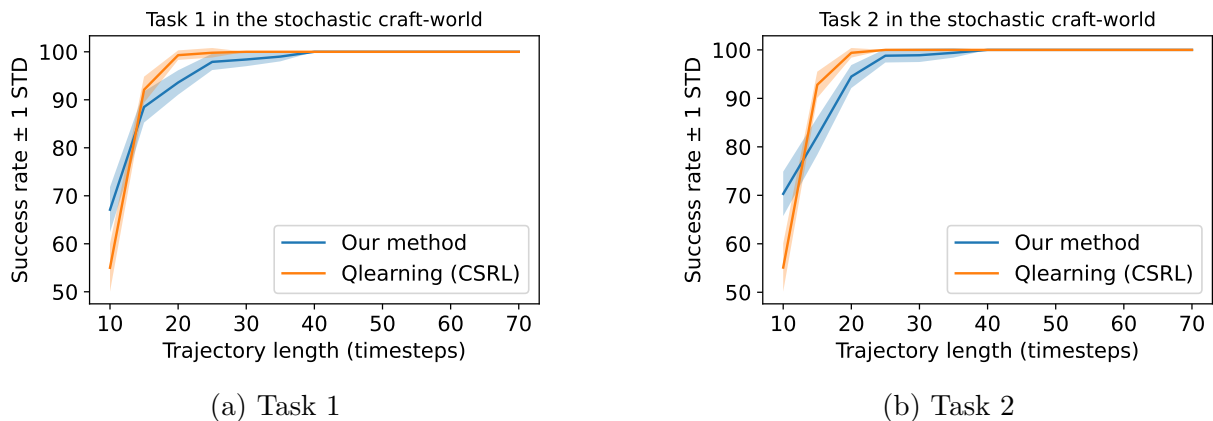
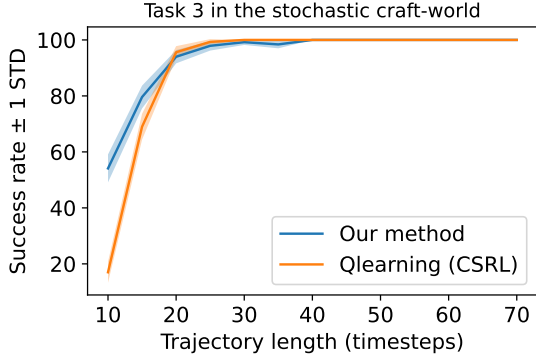
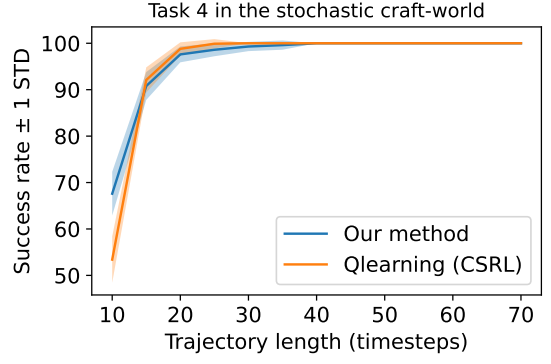


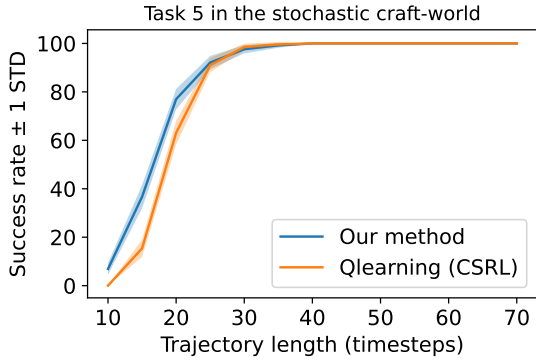
Figure 5.5: Results for the Craft-World Tasks in the Stochastic Setting (Part 1)



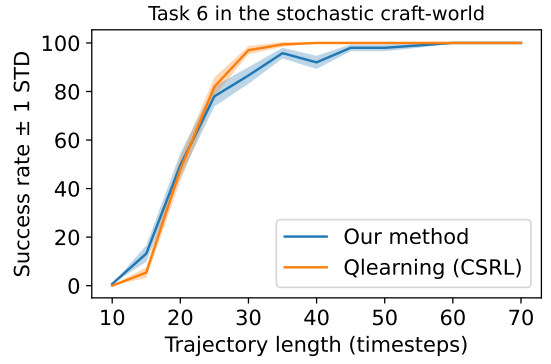
(c) Task 3



(d) Task 4

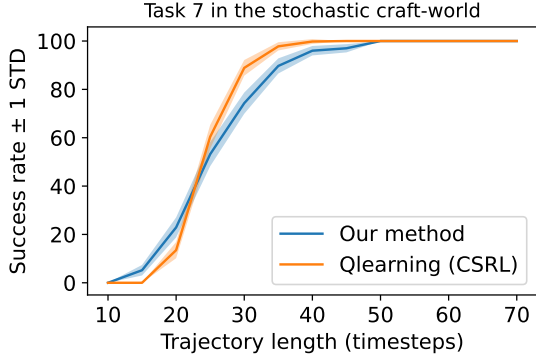


(e) Task 5

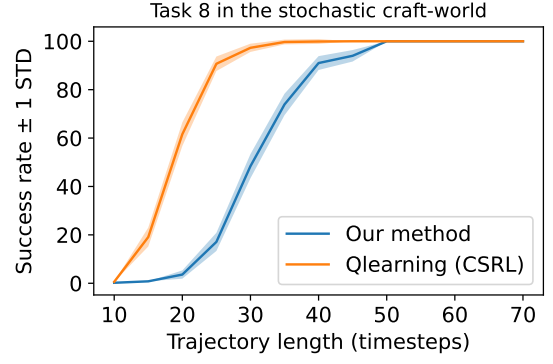


(f) Task 6

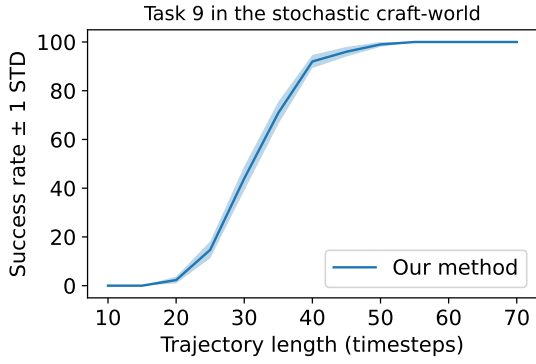
Figure 5.5: Results for the Craft-World Tasks in the Stochastic Setting (Part 3)



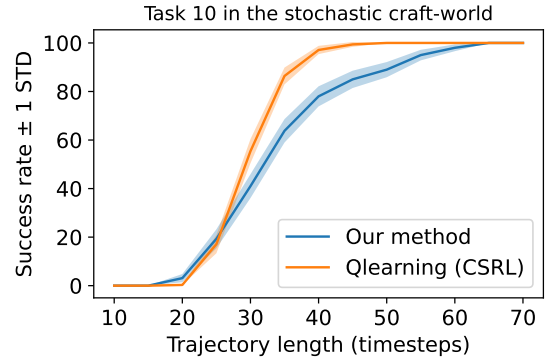
(g) Task 7



(h) Task 8



(i) Task 9



(j) Task 10

Figure 5.5: Results for the Craft-World Tasks in the Stochastic Setting (Part 5)

5.2.3 The Office-World Task

In the deterministic setting, both the proposed approach and CSRL fully solve the task within 70 steps. In the stochastic setting, the proposed approach achieves a higher success rate when the agent is given a short trajectory length to solve the task, while CSRL outperforms the proposed method when allowed longer trajectories (see Fig. 5.6).

As mentioned in chapter 2, CSRL assumes an infinite horizon when learning its policies, which leads to performance degradation when limited to short trajectory lengths. However, when permitted longer trajectories, its policy can eventually solve the task.

CRM [46], the source of this case study, demonstrates task completion in the deterministic setting starting from one initial state. In this evaluation, CRM was tested in a stochastic environment starting from random initial states, with average results reported across 60 independent runs, similar to their methodology.

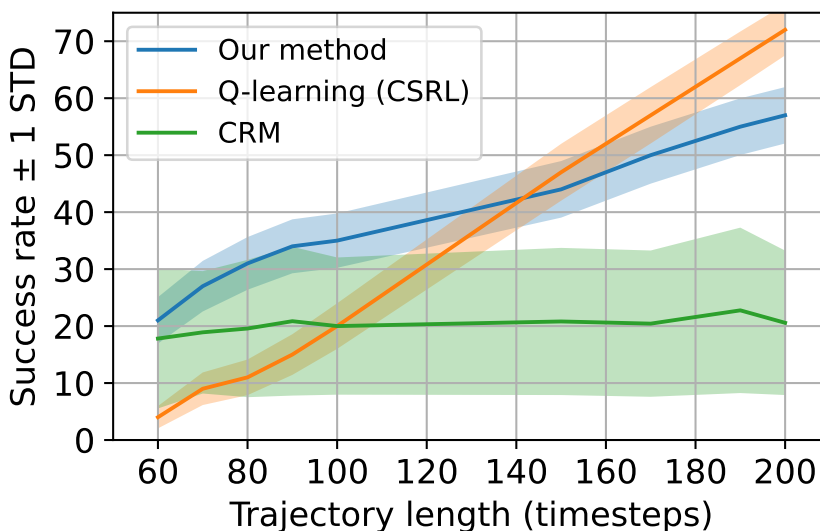


Figure 5.6: Results for the office world in the stochastic environment.

As observed in Fig. 5.6, CRM, with the reward machine designed for the task, does not perform as well in the stochastic environment as it does in the deterministic setting. This performance discrepancy is likely due to the reward machine being tailored for a deterministic environment, which may not provide suitable signals for effective learning in a stochastic setting.

5.3 Time Analysis of the Different Approaches

This section presents a time analysis of the proposed approach compared to other methods across different tasks (averaged across 10 runs). The analysis highlights the training time differences for each method and discusses how our approach performs in relation to CSRL and CRM in various settings. Note that running tests is done very quickly since the policies are learned (recall section 4.4) and is mostly done in a matter of seconds.

Starting with the first case (the sequential delivery case), Table 5.1 shows the comparison of training times between the proposed method and CSRL across different environments with varying numbers of danger zones.

# of Danger Zones	Our Training Time ($\pm$ std)	CSRL Training Time
0	2.1 m $\pm$ 1.1 m	4.5 m
1	3.3 m $\pm$ 1.5 m	4.5 m
2	2.9 m $\pm$ 1.3 m	4.5 m
3	5.7 m $\pm$ 1.9 m	4.5 m
4	6.1 m $\pm$ 1.8 m	>6 h
5	7.4 m $\pm$ 2.3 m	>6 h
6	8.2 m $\pm$ 2.9 m	>6 h

Table 5.1: Training time comparison between our method and CSRL for different numbers of danger zones. In this table, m stands for minutes and h stands for hours.

Table 5.2 provides the training time comparison between our method, CSRL, and CRM for the office-world task, demonstrating the performance of each method in this specific environment.

Method	Training Time ($\pm$ std)
Our Method	118.3 $\pm$ 31.2 m
CSRL	183.5 m
CRM	39.3 $\pm$ 23.6 m

Table 5.2: Training time comparison for the office-world task (m stands for minutes).

Finally, table 5.3 presents a comparison of training times between our method and CSRL across ten craft-world tasks. These results show how our approach scales as task complexity increases.

Craft-World Task	Our Training Time ($\pm$ std)	CSRL Training Time
1	6.2 m $\pm$ 2.1 m	15 m
2	8.6 m $\pm$ 2.3 m	15 m
3	7.1 m $\pm$ 2.8 m	15 m
4	8.8 m $\pm$ 3.1 m	15 m
5	41.2 m $\pm$ 14.5 m	1 h
6	47 m $\pm$ 18.2 m	1 h
7	50.9 m $\pm$ 18.5 m	1 h
8	56.3 m $\pm$ 24.8 m	1 h
9	127.8 m $\pm$ 49.6 m	—
10	116.3 m $\pm$ 45.9 m	1 h

Table 5.3: Training time comparison for craft-world tasks between our method and CSRL. The CSRL training time for the ninth task is not available. In this table, m stands for minutes and h stands for hours.

5.3.1 Summary of Results

The proposed approach excels in solving complex tasks within environments that have a high probability of violating safety specifications, as seen in Fig. 5.1c. In these challenging scenarios, the baseline method struggles to find solutions even in deterministic settings. By treating time, specifically the allowed trajectory length, as a constrained resource, the proposed method consistently achieves higher success rates compared to existing approaches, particularly when tasked with solving problems within a short duration in stochastic environments, such as the one described in Sec. 5.1.3.

Additionally, the method was evaluated on a series of tasks from [5], demonstrating its ability to tackle complex specifications that require intricate levels of planning in stochastic settings. The results show that the proposed approach is not only robust but also versatile, capable of addressing a wide array of challenges across various environments. This highlights its potential for broader applications where traditional methods fall short, reaffirming the effectiveness and reliability of the proposed solution.

Chapter 6

Extensions for the Current Work and Conclusion

In this section, two lines of work that have been undertaken to extend the current research will first be reviewed. Following this, the results obtained will be discussed and summarized, highlighting their implications and significance. Finally, the section will be concluded by reflecting on the overall contributions and future directions of the work.

There are many avenues to extend the current work, potentially involving various aspects such as dealing with dynamic worlds (where labels in the environment move), expanding the action space of the agent, working in continuous state spaces, attempting to solve new tasks in a zero-shot manner, and more. In this chapter, two specific ideas that were investigated will be explored and their results reviewed. Following this, we will discuss the limitations of the current approach and propose potential solutions to address some of them.

The first extension focuses on improving the learning phase for new environments by leveraging training on similar ones. The second extension expands the setup to work with multiple agents. While these ideas are not fully mature, they shed light on additional possibilities that can be built upon the current work.

6.1 Improved Performance for New Environments via Pre-Training

If the problem is examined from a high level, it becomes evident that there are two main components: the task specification and its corresponding [LDBA](#), and the [MDP](#) of the environment. One limitation of the proposed approach is that training on a product [MDP](#) is not transferable to other problems. In other words, if the product [MDP](#) changes, the [MCTS](#) parameters for its states change and are no longer applicable to a new problem setup. This significantly constrains the training process because even a small change in either the [MDP](#) (e.g., relocating a label) or the task specification necessitates restarting the training from scratch.

However, by making certain assumptions, it is possible to enhance the training process for a new problem (i.e., product [MDP](#)). The idea revolves around fixing the task specification and changing the [MDP](#) by relocating the labels in the environment (without altering the size of the grid-world). Let’s explore this approach further.

If the [LTL_f](#) formula remains fixed, the [LDBA](#) associated with it also remains unchanged. Similarly, if the size of the grid-world does not change, the input shape to the network remains consistent. This is because the input format of the network (see Sec. 4.2.1) depends on i) the shape of the environment, and ii) the number of [LDBA](#) states. Consequently, training the network on a specific problem and then altering the grid-world by relocating the labels allows the new product [MDP](#) states to be fed into the network to obtain outputs.

Although the product [MDP](#) has changed, necessitating the re-learning of [MCTS](#) parameters, the network may retain useful information from the previous grid-world. For instance, it may learn about the dangers of unsafe zones or the importance of destination labels. This retained knowledge can help provide better policy priors to the [MCTS](#) earlier in the training process, thereby accelerating the training for the new environment.

To test this idea, the [LTL_f](#) formula was fixed as the sequential delivery task. A set of ten random 10×10 test grid-worlds was implemented to benchmark the effect of training the network on similar worlds. Fig. 6.1 is an example of the ten grid-worlds included in the test set. Additionally, a set of 500 similar random grid-worlds was created for training.

First, to establish a baseline, the agent was tasked with solving each of the test set grid-worlds independently, without any pre-training on similar worlds from the training set. The success rate of the agent was recorded after each iteration of the training on the test set worlds, up to 11 iterations. Each world was trained independently ten times. The average success rates for each iteration of the training were recorded as the success

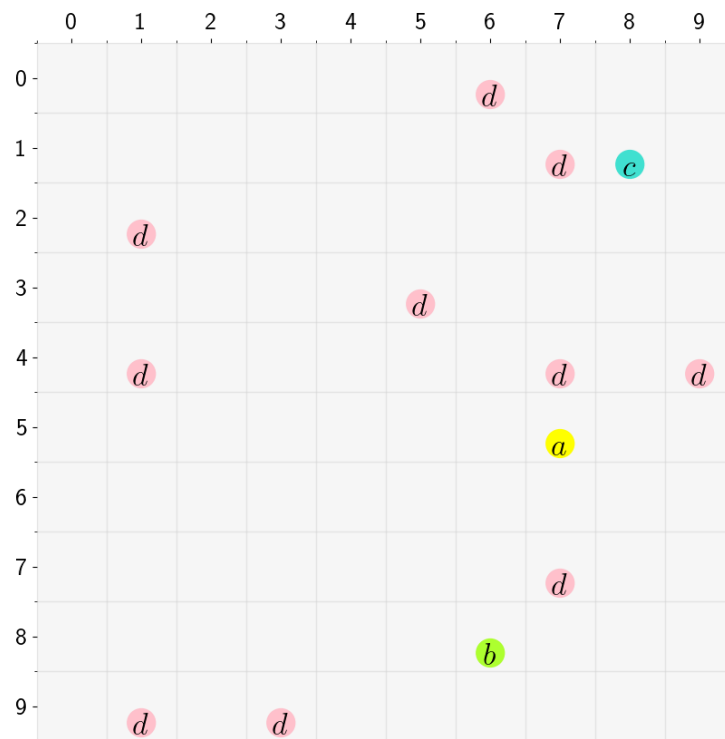


Figure 6.1: An example of the random 10×10 grild-world created.

trend (with training iterations) for each world. Table 6.1 shows these average values. The average of all the ten worlds (last row of the table) was used to measure the improvement in success rates.

A second table was produced for the test set worlds, but this time, the network was pre-trained on a training set of 500 random worlds similar to the test set. The weights of the network were carried over from the training on the 500 training worlds to the test worlds. The success rate per iteration over the test set, using the pre-trained model, is shown in Table 6.2.

To measure the effect of pre-training, two metrics were developed to compare the sequences of success rates with and without pre-training. The first metric is the performance metric, which essentially measures the blue area between the success rate curves (see Fig. 6.2). This metric evaluates whether the success rates were above the baseline after pre-training. A positive value in this metric indicates that pre-training resulted in faster learning and better initial performance compared to the baseline. This value in Fig. 6.2

Table 6.1: Average Success Rates for Each Iteration of Training on Test Set Worlds

Iteration	1	2	3	...	10	11
World 1	0.0652	0.0656	0.2148	...	0.7012	0.7010
World 2	0.3184	0.4340	0.5752	...	0.8084	0.8046
World 3	0.1096	0.2284	0.2440	...	0.7260	0.7372
World 4	0.0000	0.0832	0.2672	...	0.6240	0.6204
World 5	0.0996	0.4336	0.5184	...	0.7436	0.7380
World 6	0.2080	0.4076	0.4384	...	0.7844	0.8122
World 7	0.0000	0.0900	0.2060	...	0.6496	0.6506
World 8	0.0348	0.0608	0.1748	...	0.6232	0.6472
World 9	0.0168	0.0572	0.0656	...	0.5004	0.5464
World 10	0.1096	0.3336	0.4776	...	0.8392	0.8366
Average	0.0964	0.2192	0.3184	...	0.6953	0.7091

is equal to 0.9869. divided by 11 (number of iterations) it suggests we gain $\simeq 9\%$ higher success rates per iteration.

The second metric involves comparing the success rates after the same amount of training iterations. By examining the final values of each success rate sequence (red dotted line in Fig. 6.2), this metric shows how much better the agent performs on average after a fixed training period with and without pre-training. This value is 0.0773, meaning we gain a 7.7% higher success rate if we used the pre-trained model. This comparison helps determine the overall effectiveness of pre-training in improving the agent’s performance over the same training duration.

These metrics provide insights into both the initial acceleration of learning due to pre-training and the final performance improvement achieved through pre-training.

6.2 Scaling to Multiple Agents

Introducing multiple agents to our problem setup can add many layers of complexity and can be approached in various ways. Decisions such as whether the state space of the agents is joint or independent, whether the agents share information, whether they have independent tasks or a common task to achieve together, and whether they are competing or cooperating are all factors that can significantly affect the complexity of the problem. Additionally, the existence of multiple agents imposes constraints on runtime and computational resources. This avenue of research is extensive, and The motivation behind this

Table 6.2: Average Success Rates for Each Iteration with Pre-trained Model

Iteration	1	2	3	...	10	11
World 1	0.2952	0.2995	0.3460	...	0.8953	0.8460
World 2	0.3380	0.3980	0.6019	...	0.9616	0.9809
World 3	0.3122	0.3401	0.3103	...	0.8164	0.9226
World 4	0.2800	0.2823	0.2943	...	0.7532	0.7480
World 5	0.2794	0.5079	0.4799	...	0.8036	0.6958
World 6	0.3161	0.4006	0.5084	...	0.7384	0.8074
World 7	0.2800	0.2930	0.3073	...	0.7222	0.6282
World 8	0.2876	0.2765	0.2663	...	0.7318	0.6659
World 9	0.2798	0.2872	0.2800	...	0.5606	0.5733
World 10	0.2956	0.3463	0.6360	...	0.7889	1.0000
Average	0.2964	0.3431	0.4030	...	0.7772	0.7868

section is to explore the possibility of applying the same technique at a larger scale and understand the challenges and potential solutions in a multi-agent setup.

In this section, a basic version of the problem with multiple agents will be considered. The setup will be described, followed by a discussion of the results obtained using the proposed approach. So far, the focus has been on a single-agent setup, where the agent is given a task to solve in a labeled [MDP](#). This setup can be extended to multiple agents who are attempting to solve the same task, with the added constraint that they must avoid collisions with each other.

This setup inherently limits the types of tasks that can be designed for the agents. Specifically, tasks requiring agents to remain at a certain location indefinitely cannot be satisfied if agents are prohibited from occupying the same position simultaneously due to collision avoidance constraints. To extend the current approach to multiple agents, the agents can be initialized in the environment. According to [Alg. 1](#), each agent would run many [MCTS](#) samples to help decide its next move. However, this would require the agent to have some understanding of the behavior of other agents to accurately simulate the environment.

Adding other moving agents essentially transforms the environment into a dynamic one. As discussed at the beginning of this chapter, the current approach cannot handle dynamic environments because changes in the [MDP](#) also change the product [MDP](#). However, it is possible to separate the agents' locations from the [MDP](#) and treat them as separate entities. These entities affect the behavior of the agent purely by enforcing a negative reward if a

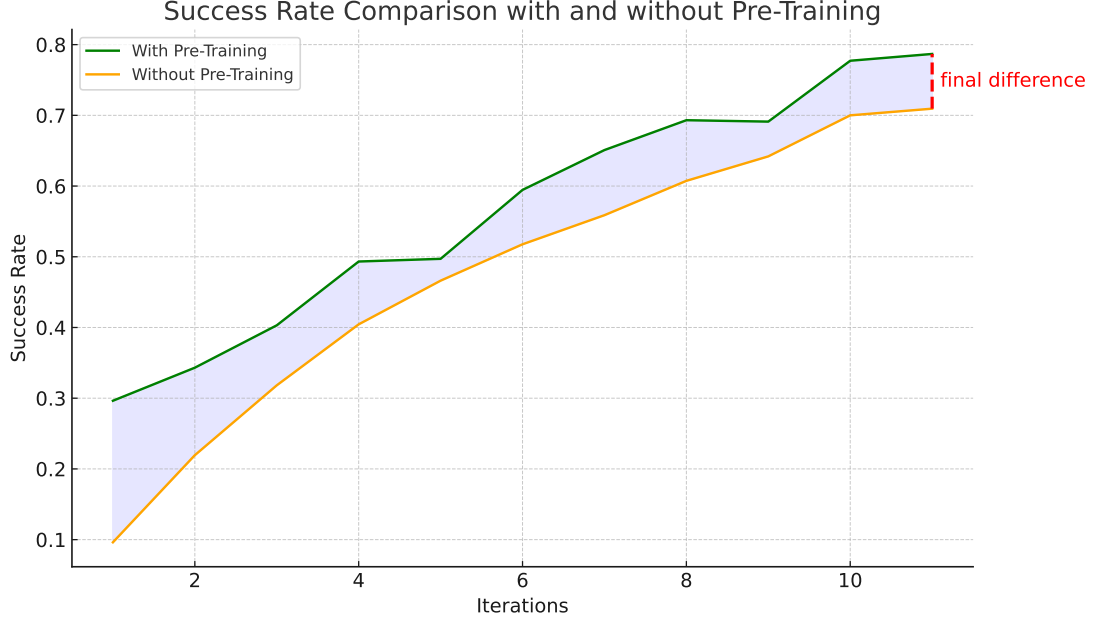


Figure 6.2: Compariosn of average success rates for the ten test worlds.

collision occurs. Therefore, [MCTS](#) samples can be run either by having all agents share the same joint policy, which they can use for other agents in their [MCTS](#) simulations, or by having independent policies, where each agent has an internal belief about the behavior of other agents.

The first approach has the advantage of accuracy, as each agent has access to the latest state of other agents' policies. However, it also has drawbacks, such as difficulty in training agents separately in a decentralized manner and potential overhead on the communication channel, which may not be feasible if the communication channel is limited. The second approach avoids these drawbacks but may degrade the overall system's quality due to each agent's internal estimates.

In the proposed setup for this section, the agent's belief about other agents is simplified. Since all [MCTS](#) samples are gathered for the next immediate actions, the agent can assume that all other agents are static while planning its next move. This assumption is inaccurate for future timesteps, as other agents will move, but it is reasonable for the next immediate action if agents take turns selecting their next moves. On the other hand, the assumption

of sequential decision-making over agents lengthens the decision-making process, as each agent must wait its turn to run its simulations and make its move. This is not the ideal setup. A more appealing setup is where all agents run their [MCTS](#) simulations in parallel, but this introduces challenges in modeling other agents' actions when each agent is running its [MCTS](#).

Therefore, each agent has its own internal state of the environment, similar to the single-agent setup, but the overall state of all agents is a product of all their states. There is no need to track the progress of the overall state directly; keeping track of individual agent states is sufficient and simpler to implement. The task is considered solved if all agents satisfy it. As a consequence of our setup, the state-space of the problem grows by an order of $O(k^n)$ where n is the number of agents and k is the multiplication of the number of [LDBA](#) states and the grid-world dimensions.

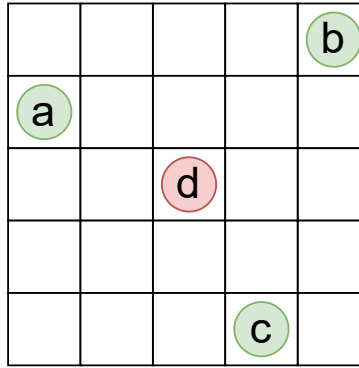


Figure 6.3: Environment for the multi-agent setup.

Let's quickly review the characteristics of the multi-agent setup:

- There can be n number of agents simultaneously in the [MDP](#).
- All agents are solving the same [LTL_f](#) task.
- There is an additional constraint preventing agents from colliding (i.e., being at the same location simultaneously).
- Agents take turns choosing their moves and assume other agents are static labels in the environment during [MCTS](#) sampling.

- If agents collide, a negative reward is immediately produced, and the [MCTS](#)/run terminates.
- A positive reward is only given to a run of the environment if all agents satisfy the task.

To test the above setup, a simple grid-world was created (Fig. 6.3). The agents were given a simple task of eventually visiting a , b , and c (in any order) while avoiding hitting each other and the danger zones d at all times. This task can be written as an [LTL_f](#) task as follows Where n is the number of agents, and ' A_i at x ' means the event that agent i is at label x :

$$\begin{aligned} \varphi_{\text{multi-agent task}} = \\ \bigwedge_{i=1}^n ((\Box \neg A_i \text{ at } d) \wedge (\Diamond A_i \text{ at } a) \wedge (\Diamond A_i \text{ at } b) \wedge (\Diamond A_i \text{ at } c)) \bigwedge_{\substack{i,j \\ i \neq j}} \Box \neg (A_i \text{ at } A_j) \end{aligned} \quad (6.1)$$

With the above setup, it is no longer possible to extract a fixed policy like in Eq. 4.7 and the only way to produce a trajectory for all the agents is to run Algorithm 1. Algorithm 1 is slow on its own, but it becomes even slower when used for multiple agents since the number of agents has increased and the state space of the problem has expanded. This makes it difficult to train and test the agents, especially when there are more than two agents in the environment.

Figure 6.4 shows the results for running the above scenario with 2, 3, and 4 agents over different time lengths. This setup was only tested in the deterministic environment. For each configuration with a fixed number of agents and a fixed trajectory length, 10 sets of 10 runs (for a total of 100 runs) were conducted, and their success rates were averaged and recorded. The locations of the agents were chosen randomly for each run.

As seen in Fig. 6.4, as the number of agents increases, the success rate drastically decreases. The reason for this is most likely because navigating the environment with multiple agents makes it difficult to avoid collisions. Expanding the size of the environment could help mitigate this issue, but it also increases the state space, making it more challenging to explore and solve. Another observation was that the main reasons for task failure were either not completing the task within the given time (agents got stuck in loops that did not accomplish the task) or colliding with each other. The danger zone was almost always avoided by all agents. This suggests, as expected, that dealing with fixed labels in the environment is easier for the proposed algorithm, while handling moving labels (agents) requires further modifications to address this problem. This becomes more pronounced considering that solving this environment almost perfectly in both deterministic and stochastic setups is straightforward for one agent with the current algorithm.

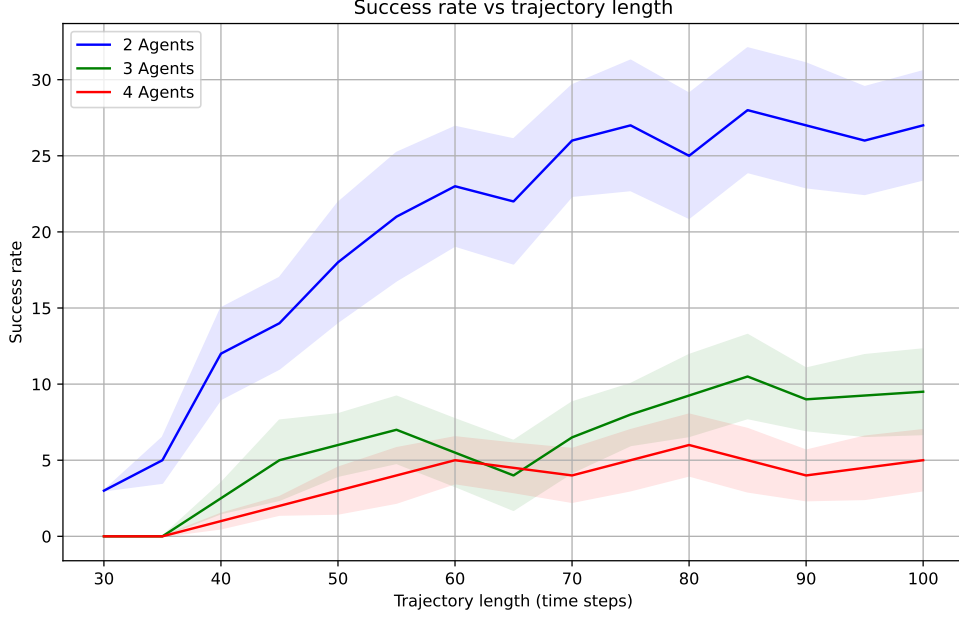


Figure 6.4: Success rates for the multi-agent setups with different number of agents.

6.3 Discussion and Conclusion

The proposed approach effectively utilizes the combination of a neural network and [MCTS](#) to generate trajectories that satisfy [\$LTL_f\$](#) specifications in discrete state and action systems. The achievement of perfect success rates in all deterministic scenarios with limited trajectory lengths demonstrates that this method can confidently and quickly identify promising moves to satisfy complex [\$LTL_f\$](#) specifications in various environments. This is particularly valuable in time-constrained environments, where efficient sampling is critical.

Moreover, the higher success rates observed in comparison to state-of-the-art approaches, particularly in environments with numerous unsafe zones (case [5.2.1](#)) and stochasticity in robot movements, underscore the robustness of the proposed method. The ability to maintain performance in challenging environments highlights the approach’s potential for real-world applications, where unpredictability and risk are often present.

However, certain trends in the results warrant further discussion. For instance, some of the plots (e.g., Fig. [5.6](#) and [6.4](#)) exhibit a plateau in success rates after a certain trajectory

length. This phenomenon suggests that while the proposed approach excels in scenarios with time constraints, it may encounter limitations in situations where time is not a critical factor. Specifically, the method may struggle to find the optimal policy (in terms of the length of the found trajectory) when the problem demands exhaustive exploration and long trajectories to achieve the desired outcome.

These observations indicate that while the proposed approach performs well in many cases, its design may inherently favor scenarios where decision-making within limited time frames is crucial. This trade-off, though advantageous in many applications, points to potential areas for improvement, particularly in refining the method for environments where time is abundant and the optimality of the policy is paramount.

6.3.1 Limitations and future work

While the simulation studies illustrate the applicability and performance of this approach across a range of problems, the proposed method does have limitations. primarily, unlike [12], the current approach does not provide theoretical guarantees for maximizing the probability of satisfying the [LTL](#) specification. Additionally, this approach lacks completeness guarantees, meaning it may fail to find a trajectory that satisfies a specification, even when one exists.

Despite the absence of theoretical soundness guarantees, rigorous simulation evaluations demonstrate practical success in scenarios where even state-of-the-art methods fail. This approach is well-suited for static environments where the locations of items in the environment do not change over time (with the exception of the agent). Testing this approach with multiple agents reveals its potential to solve the same [LTL](#) specifications in systems with more agents. However, the increase in state-space and the addition of dynamic elements make the problem more challenging. While the proposed approach can still address these cases, further modifications are necessary for optimal performance.

One significant challenge in dealing with larger problems is the exponential growth of the state space, particularly when working with the product [MDP](#). As the size of the [MDP](#) and the number of states in the [LDBA](#) increase, the current approach may struggle to scale efficiently. This is especially true when the specification results in an enormous [LDBA](#), leading to a dramatic increase in the state space to be explored. To address this issue, one potential solution would be to replace the tables used for storing [MCTS](#) statistics with function approximators, which can estimate values for each state dynamically. Another approach could be inspired by AlphaGo Zero, where only nearby states of the problem are tracked, instead of the entire state space, while employing a receding horizon strategy to

update and discard state statistics as needed. Nevertheless, it should be emphasized that the current method does face limitations when applied to problems of larger scale.

Additionally, one of the key findings of this research is that the agent struggles when tasked with satisfying specifications that require navigating through several sequential steps before reaching the final goal. Specifications that involve checking a sequence of conditions, particularly those requiring around five or more tasks (based on experience) to be completed in order, proved challenging. In such cases, the agent had difficulty identifying the correct sequence of actions without intermediate rewards. This type of task is especially hard for the current setup, as it lacks intermediate guidance. Introducing some form of intermediate reward system could significantly aid the agent in discovering the correct sequence of actions required to satisfy the specification.

The overall solution described in this work operates in a simulated environment, which assumes perfect inputs and outputs. In the real world, noise, imperfect sensor inputs, and actuator inaccuracies are common challenges. These factors can significantly impact the performance and reliability of the approach. Therefore, deploying this solution in real-world settings would require additional efforts to account for and mitigate these issues, ensuring that the system performs to the same standard. Experimenting with stochastic environments, as has already been done, represents a step forward toward real-world implementation, addressing some of the uncertainties present in practical scenarios. Future work could focus on integrating robust control techniques to handle noise and uncertainties in sensor inputs and actuators, as well as testing the approach in real-world environments to assess its performance under practical conditions.

6.3.2 Conclusion

In this work, an [RL](#)-based method was proposed to solve [LTL](#) specifications in a system with finite states and actions, modeled as an [MDP](#) with a priori unknown transition probabilities. Our approach first employs a neural network combined with [MCTS](#) to generate runs that satisfy the task. A policy is then extracted from the [MCTS](#) parameters to enable online deployment and testing.

Extensive simulations demonstrate that this approach can successfully learn policies to satisfy complex [LTL_f](#) specifications within a maximum number of steps (e.g., corresponding to robot battery life or a time limit). This method learns robust policies for the stochastic cases studied, where either a state-of-the-art approach could not satisfy the specification of interest or performs better when constrained by a limited trajectory duration.

References

- [1] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015.
- [2] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425. IEEE, 1990.
- [3] Glenn Ammons, Rastislav Bodík, and James R Larus. Mining specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.
- [4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.
- [5] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International conference on machine learning*, pages 166–175. PMLR, 2017.
- [6] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.
- [7] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.
- [8] Franz Baader, Andreas Bauer, and Marcel Lippmann. Runtime verification using a temporal description logic. In *International Symposium on Frontiers of Combining Systems*, pages 149–164. Springer, 2009.

- [9] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):1–64, 2011.
- [10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
- [11] Pierfrancesco Bellini, Riccardo Mattolini, and Paolo Nesi. Temporal logics for real-time system specification. *ACM Computing Surveys (CSUR)*, 32(1):12–42, 2000.
- [12] Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10349–10355. IEEE, 2020.
- [13] Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Model-free reinforcement learning for stochastic games with linear temporal logic objectives. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10649–10655. IEEE, 2021.
- [14] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
- [15] Barrett R Bryant, Jeff Gray, Marjan Mernik, Peter J Clarke, Robert B France, and Gabor Karsai. Challenges and directions in formalizing the semantics of modeling languages. 2011.
- [16] J Richard Büchi. On a decision method in restricted second order arithmetic, logic, methodology and philosophy of science (proc. 1960 internat. congr.), 1962.
- [17] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.
- [18] Serenella Cerrito and Marta Cialdea Mayer. Using linear temporal logic to model and solve planning problems. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 141–152. Springer, 1998.

- [19] William Chan, Richard J Anderson, Paul Beame, Steve Burns, Francesmary Modugno, David Notkin, and Jon D Reese. Model checking large software specifications. *IEEE Transactions on software Engineering*, 24(7):498–520, 1998.
- [20] Nai-Yuan Chang, Chih-Hung Chen, Shun-Shii Lin, and Surag Nair. The big win strategy on multi-value network: An improvement over alphazero approach for 6x6 othello. In *Proceedings of the 2018 international conference on machine learning and machine intelligence*, pages 78–81, 2018.
- [21] Edmund M Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*, pages 54–56. Springer, 1997.
- [22] Edmund M Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
- [23] Edmund M Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.
- [24] Clarissa Costen, Marc Rigter, Bruno Lacerda, and Nick Hawes. Planning with hidden parameter polynomial mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11963–11971, 2023.
- [25] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI’13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.
- [26] Giuseppe De Giacomo, Moshe Y Vardi, et al. Synthesis for LTL and LDL on finite traces. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 1558–1564. AAAI Press, 2015.
- [27] Xu Chu Ding, Calin Belta, and Christos G Cassandras. Receding horizon surveillance with temporal logic specifications. In *49th IEEE Conference on Decision and Control (CDC)*, pages 256–261. IEEE, 2010.
- [28] Xuchu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, 2014.

- [29] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What’s new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV’22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022.
- [30] E Allen Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.
- [31] Chao Gao, Ryan Hayward, and Martin Müller. Move prediction using deep convolutional neural networks in hex. *IEEE Transactions on Games*, 10(4):336–343, 2017.
- [32] Chao Gao, Martin Müller, and Ryan Hayward. Three-head neural network architecture for monte carlo tree search. In *IJCAI*, pages 3762–3768, 2018.
- [33] Audrey Gaymann and Francesco Montomoli. Deep neural network and monte carlo tree search applied to fluid-structure topology optimization. *Scientific reports*, 9(1):15916, 2019.
- [34] Dimitra Giannakopoulou. *Model checking for concurrent software architectures*. PhD thesis, University of London, 1999.
- [35] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey 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. IEEE, 2017.
- [36] Meng Guo and Dimos V Dimarogonas. Multi-agent plan reconfiguration under local ltl specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.
- [37] Meng Guo and Michael M Zavlanos. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control*, 63(12):4051–4066, 2018.
- [38] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.

- [39] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. *arXiv preprint arXiv:1311.2928*, 2013.
- [40] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.
- [41] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *Formal Modeling and Analysis of Timed Systems: 18th International Conference, FORMATS 2020, Vienna, Austria, September 1–3, 2020, Proceedings 18*, pages 1–22. Springer, 2020.
- [42] Klaus Havelund and Doron Peled. Runtime verification: from propositional to first-order temporal logic. In *Runtime Verification: 18th International Conference, RV 2018, Limassol, Cyprus, November 10–13, 2018, Proceedings 18*, pages 90–112. Springer, 2018.
- [43] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.
- [44] Gerard J Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.
- [45] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.
- [46] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.
- [47] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-logic-based reward shaping for continuing reinforcement learning tasks. In *Proceedings of the AAAI Conference on artificial Intelligence*, volume 35, pages 7995–8003, 2021.
- [48] Mathai Joseph, Alan Burns, Andy Welling, Krithi Ramamritham, Jozef Hooman, Steve Schneider, Zhiming Liu, and Henk Schepers. *Real-Time Systems: specification, verification, and analysis*, volume 62. prentice Hall Englewood Cliffs, 1996.

- [49] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [50] Yiannis Kantaros and Michael M Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control*, 64(5):1916–1931, 2018.
- [51] Michael Kearns and Satinder Singh. Finite-sample convergence rates for q-learning and indirect algorithms. *Advances in neural information processing systems*, 11, 1998.
- [52] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic verification of linear temporal logic specifications. In *Automata, Languages and Programming: 25th International Colloquium, ICALP’98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*, pages 1–16. Springer, 1998.
- [53] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220, 2009.
- [54] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.
- [55] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:211–236, 2018.
- [56] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: from ltl to your favourite deterministic automaton. In *International Conference on Computer Aided Verification*, pages 567–577. Springer, 2018.
- [57] Moez Krichen. Formal methods and validation techniques for ensuring automotive systems security. *Information*, 14(12):666, 2023.
- [58] Marta Kwiatkowska and David Parker. Automated verification and strategy synthesis for probabilistic systems. In *Automated Technology for Verification and Analysis: 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, pages 5–22. Springer, 2013.

- [59] Morteza Lahijanian, Sean B Andersson, and Calin Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, 2011.
- [60] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.
- [61] Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Torbjorn S Dahl, Amine Kerkeni, and Karim Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672*, 2018.
- [62] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3839. IEEE, 2017.
- [63] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [64] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*, 2017.
- [65] Thibaud Michaud and Maximilien Colange. Reactive synthesis from ltl specification with spot. In *Proceedings of the 7th Workshop on Synthesis, SYNT@ CAV*, volume 5, 2018.
- [66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [68] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deep-stack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

- [69] Mirco Mutti, Mattia Mancassola, and Marcello Restelli. Unsupervised reinforcement learning in multiple environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7850–7858, 2022.
- [70] Rani Nelken and Nissim Francez. Automatic translation of natural language system specifications into temporal logic. In *Computer Aided Verification: 8th International Conference, CAV’96 New Brunswick, NJ, USA, July 31–August 3, 1996 Proceedings 8*, pages 360–371. Springer, 1996.
- [71] Evgeny Novikov and Ilja Zakharov. Verification of operating system monolithic kernels without extensions. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV 8*, pages 230–248. Springer, 2018.
- [72] Santiago Ontanón. Informed monte carlo tree search for real-time strategy games. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.
- [73] Jonathan S Ostroff. *Temporal logic for real time systems*. John Wiley & Sons, Inc., 1989.
- [74] Ivan Papusha, Jie Fu, Ufuk Topcu, and Richard M Murray. Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 434–440. IEEE, 2016.
- [75] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.
- [76] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J Lim. Guided reinforcement learning with learned skills. *arXiv preprint arXiv:2107.10253*, 2021.
- [77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. iee, 1977.
- [78] Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. *Current Trends in Concurrency: Overviews and Tutorials*, pages 510–584, 2005.

- [79] Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. *Advances in Neural Information Processing Systems*, 36, 2024.
- [80] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248, 2015.
- [81] Mark Reynolds and Clare Dixon. Theorem-proving for discrete temporal logic. In *Foundations of Artificial Intelligence*, volume 1, pages 279–313. Elsevier, 2005.
- [82] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degraeve, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.
- [83] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.
- [84] Kristin Y Rozier. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2):163–203, 2011.
- [85] Alexander Schimpf, Stephan Merz, and Jan-Georg Smaus. Construction of büchi automata for ltl model checking verified in isabelle/hol. In *International Conference on Theorem Proving in Higher Order Logics*, pages 424–439. Springer, 2009.
- [86] Philippe Schnoebelen. The complexity of temporal logic model checking. *Advances in modal logic*, 4(393-436):35, 2002.
- [87] Roberto Sebastiani and Stefano Tonetta. “more deterministic” vs. “smaller” büchi automata for efficient ltl model checking. In *Correct Hardware Design and Verification Methods: 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L’Aquila, Italy, October 21-24, 2003. Proceedings 12*, pages 126–140. Springer, 2003.
- [88] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic büchi automata for linear temporal logic. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 312–332. Springer, 2016.

- [89] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [90] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [91] A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.
- [92] Fabio Somenzi and Roderick Bloem. Efficient büchi automata from ltl formulae. In *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*, pages 248–263. Springer, 2000.
- [93] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006.
- [94] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.
- [95] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.
- [96] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.
- [97] Daiying Tian, Hao Fang, Qingkai Yang, Haoyong Yu, Wenyu Liang, and Yan Wu. Reinforcement learning under temporal logic constraints as a sequence modeling problem. *Robotics and Autonomous Systems*, 161:104351, 2023.
- [98] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018.

- [99] Yih-Kuen Tsay. Compositional verification in linear-time temporal logic. In *Foundations of Software Science and Computation Structures: Third International Conference, FOSSACS 2000 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2000 Berlin, Germany, March 25–April 2, 2000 Proceedings 3*, pages 344–358. Springer, 2000.
- [100] Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.
- [101] Moshe Y Vardi. Automata-theoretic model checking revisited. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 137–150. Springer, 2007.
- [102] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.
- [103] Steen Vester. On the complexity of model-checking branching and alternating-time temporal logics in one-counter systems. In *Automated Technology for Verification and Analysis: 13th International Symposium, ATVA 2015, Shanghai, China, October 12–15, 2015, Proceedings 13*, pages 361–377. Springer, 2015.
- [104] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. Model checking programs. *Automated software engineering*, 10:203–232, 2003.
- [105] Cameron Voloshin, Hoang Le, Swarat Chaudhuri, and Yisong Yue. Policy optimization with linear temporal logic constraints. *Advances in Neural Information Processing Systems*, 35:17690–17702, 2022.
- [106] Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 35137–35150, 2023.
- [107] Chuankeng Wang, Yinan Li, Stephen L Smith, and Jun Liu. Continuous motion planning with temporal logic specifications using deep neural networks. *arXiv preprint arXiv:2004.02610*, 2020.
- [108] Linnan Wang, Yiyang Zhao, Yu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search.

- In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9983–9991, 2020.
- [109] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
 - [110] Eric M Wolff, Ufuk Topcu, and Richard M Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on decision and control (CDC)*, pages 3372–3379. IEEE, 2012.
 - [111] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Automatic synthesis of robust embedded control software. In *2010 AAAI Spring Symposium Series*, 2010.
 - [112] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.
 - [113] Timothy Yee, Viliam Lisý, Michael H Bowling, and S Kambhampati. Monte carlo tree search in continuous action spaces with execution uncertainty. In *IJCAI*, pages 690–697, 2016.
 - [114] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.
 - [115] Yimeng Zhuang, Shuqin Li, Tom Vincent Peters, and Chenguang Zhang. Improving monte-carlo tree search for dots-and-boxes with a novel board representation and artificial neural networks. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 314–321. IEEE, 2015.