Title

Learning to Solve Long-Horizon Tasks with Formal Logic and Structured Feedback

ISBN

9798189271854

Author

Shah, Ameesh

Publication Date

2026-05-01

Peer reviewed|Thesis/dissertation

Learning to Solve Long-Horizon Tasks with Formal Logic and Structured Feedback

By

Ameesh Shah

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sanjit A. Seshia, Chair
Professor Shankar Sastry
Associate Professor Koushil Sreenath
Associate Professor Dorsa Sadigh

Spring 2026

Learning to Solve Long-Horizon Tasks with Formal Logic and Structured Feedback

Abstract

Learning to Solve Long-Horizon Tasks with Formal Logic and Structured Feedback

by

Ameesh Shah

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sanjit A. Seshia, Chair

Recent advancements in policy learning have enabled impressive results in robotics and cyber-physical systems (CPS), ranging from dexterous manipulation to bipedal locomotion. Despite these successes, learned robot policies still struggle to integrate high-level reasoning and feedback with low-level control, limiting their efficacy in complex, long-horizon tasks. This dissertation introduces methodologies that enable learned policies to leverage the same skills that humans use to reliably achieve complex goals in the real world: devising high-level plans, learning from partial successes, creating cooperative strategies, and dynamically adjusting behavior through trial-and-error.

The central insight across the methods contained in this thesis is to use *formal, symbolic structure* to express both complex tasks and their feedback mechanisms. In doing so, policies can learn high-level behavior from precise, structured plans while maintaining the expressivity of their underlying (deep) parameterization to learn low-level control. We explore this insight across a number of settings. We first demonstrate how exploiting the underlying structure of tasks specified via formal logic allows reinforcement learning agents to learn complex, long-horizon tasks in both the single and multi-agent setting. The precise, compositional nature of formal logic enables a dense, incremental reward structure in otherwise prohibitively sparse reward signals. We then take a step back and explore how formal task specifications can be learned from data and provide a novel method for learning these specifications from user-provided preferences. Last, we discuss policy learning in the era of foundation models, and resolve the single-shot nature of existing robot foundation models (RFMs) by devising a structured feedback mechanism that allows RFMs to iteratively learn complex tasks through trial-and-error.

As a whole, this dissertation advocates for a principled integration of symbolic structure into end-to-end policy learning techniques to enable complex, long-horizon behaviors. We conclude by discussing how a foundation model-centric field can still benefit from structured task specifications, and the role formal logic can play in an increasingly neural landscape.

To Sangita

Contents

List of Figures

List of Tables

Acknowledgments

They say that it takes a village to raise a child, and I believe this adage holds for helping someone finish a Ph.D. as well. I have many, many people I'd like to thank for helping me throughout my graduate school journey and who made this experience as enriching, colorful, and fulfilling as it has been.

First and foremost, I'd like to thank my advisor, Sanjit Seshia, who gave me the opportunity to come study at Berkeley under his tutelage and has been an immense source of support and encouragement throughout all of my ups and downs. Sanjit has always encouraged me to be thorough and exploratory when trying something new, and to not shoot my own ideas down before I even have a chance to evaluate it. I'd also like to thank my committee, Shankar Sastry, Koushil Sreenath, and Dorsa Sadigh, who have all offered me great guidance throughout different phases of graduate school, and who each separately advocated for me and my work. I'd also like to thank Swarat Chaudhuri, who has been a role model and guiding source for me from the very beginning of my research career when I was an undergraduate student at Rice, and Sergey Levine, who helped me bring my research into real-world robot learning and was an important influence on me towards the tail end of my Ph.D.

I have had a number of valuable collaborations during my Ph.D., and without these people none of my research would have been made possible. I'd like to first thank My Heroes of Academia, Adwait Godbole and Federico Mora, who were my biggest influences at Berkeley even though we all worked on different things. I'd like to also thank members of my lab group, especially Nik Lauffer and Beyazit Yalcinkaya, who I always could come to for engaging research discussions and learn from. Beyond the Learn and Verify Group, I've had some wonderful collaborators at Berkeley. I'd like to particularly thank William Chen for being my robotics sherpa, and Micah Carroll for always helping a lending hand, especially early on in my Ph.D. Beyond Berkeley, I've been fortunate to work with some brilliant people. I'd like to thank Cameron Voloshin, Chenxi Yang, and Abhinav Verma for pushing me as a researcher and helping expand my horizons. And to everyone else who worked with me and helped me during my research - even though I don't have room to list all of your names, I'm extremely grateful for your support.

Beyond lab, it was the people in Berkeley who really made this experience special. These moments include but are certainly not limited to: idle days in Cory Hall spent with Pei-Wei Chen, Adwait, and Federico, Core Blast Saturday mornings with Dibya Ghosh, Katie Kang, and Colin Li, taking laps around Berkeley Way West with Alejandro Escontrela, Cat Glossop, Chung Min Kim, Kush Hari, Amy Lu and more, morning workouts with Kevin Black, Paul Zhou, and Sarah McClure, roommate adventures with Jessy Lin, Sarah Wooders, Justin Kerr, and Aniket Tolpadi, CorePower with Laura Smith and Kevin Zakka, and crashing the Law School with Rachel Gaines, Krishna Desai, and Angelina Paul.

Although I spent most of my time in Berkeley during my graduate school career, my community extended well past campus, and the friends across the Bay Area and beyond are an absolutely immeasurable source of love and support. Teju Manchenella was my first

roommate in the Bay and made my transition to a new home as smooth as can be. Spencer Chang, who has been my brother for more than ten years, was someone I could always come to for advice and support and who I've been fortunate to grow up with. Rohan Sheth, who is not just family, but one of my best friends, has been an emotional rock for me. And for the many people of the Bay Area and my other hometowns, who have positively influenced me throughout my time at Berkeley: Hasan Seede, Natalie Bolton, Dominique Huang, Austin Bae, Andy Corbato, Michelle Dai, Alex Du, Mai Vu, Chris Sabbagh, Samyu Yagati, Kush Agrawal, Darren Corprew, Jiwon Park, and many, many more. I'd like to especially acknowledge Andy Zhang, Cici Xie, May Zhong, and Andrew Ligeralde, who were like a little family to me and always helped me stay afloat. And to Andrew Ligeralde, Takuma Makihara, and Alex Hwang - my biggest thanks go to the three of you. Alex has been my inspiration since the day I stepped foot on to Rice's campus and is the reason I started doing research. A serendipitous late-lunch conversation in the Jones Commons with Alex and Andrew is what first gave me the idea of pursuing a Ph.D., and Andrew has been the steadiest presence in not only being an incredible friend, but always grounding my perspective in how to do research. Takuma has walked his own research journey alongside me every step of the way and I've always cherished his companionship.

Lastly, I'd like to thank my incredible family for lifting me up since the very beginning. To my dad - there's no way any of this would have been possible without you. Thank you for every ounce of love, support, and affirmation you've given me. To my wonderful extended family, especially Rohan, my Ami Masi, and my Anjali Masi, but also all of my wonderful cousins, aunts, and uncles: thank you for supporting me through my formative school years in Houston and Berkeley. And to my mom: thank you for being my hero, my north star, and who I dedicate my life to. I hope to continue honoring your memory.

Chapter 1

Introduction

The fields of robotics and cyber-physical systems (CPS) have enjoyed tremendous progress in recent years due to the rapid advancement of artificial intelligence (AI) and machine learning (ML) methodologies (Xiao et al. 2025). Nowadays, we have robots that can follow open-ended natural language instructions in household settings (Physical Intelligence et al. 2025; Gemini Robotics Team et al. 2025), perform dexterous manipulation tasks like opening cabinets or using hammers and pliers (Bakker et al. 2025), and execute complex control and reasoning in domains like loco-navigation or full-body humanoid control (Radosavovic et al. 2024; Ouyang et al. 2024).

These exciting new results are due in large part to the development of highly scaled data-driven robot learning techniques. As computational hardware systems have grown able to store and process internet-scale datasets, our learning algorithms have scaled alongside them, leading to unprecedented general-purpose capabilities in the form of *foundation models* (Zhou et al. 2025) for visual-semantic comprehension and generation. Seeing this success, robotics practitioners sought to map these developments onto their field. These thrusts have included collecting massive teleoperated demonstration datasets for large-scale supervised learning (Black et al. 2024), leveraging simulation to easily collect large amounts of on-policy data for reinforcement learning (Radosavovic et al. 2024), or leveraging the huge amount of unstructured demonstration data available online (e.g. YouTube tutorials) to passively learn robotic skills (Bharadhwaj et al. 2023).

Despite the excitement around our fields' rapid movement, state-of-the-art robot policies have yet to be deployed in many meaningful commercial or industrial settings. When observing learned robot policies deployed in the real world, this comes as no surprise. Learned robot behavior generally appears nonsmooth and awkward and takes a (very) long time to accomplish simple tasks. Perhaps most pressingly, current robot policies are extremely *brittle*: they are unable to learn complicated behavior that requires multiple steps, and they struggle to generalize their learned behavior to different task or environment settings.

In this thesis, we will study this brittleness problem, particularly in the context of learning *long-horizon* task policies. The term "long-horizon" will be intentionally loosely defined, but in general refers to tasks that are typically semantically broken down into sub-tasks by

humans. For example, picking up a water bottle is not considered 'long-horizon' because humans don't typically reason about such a behavior through sub-tasks (such as 'move your hand to the water bottle', then 'grasp the water bottle', etc.) On the other hand, a task like cleaning up a dinner table will be considered long-horizon, because humans usually perform this step by step (e.g., first put the dishes in the sink, then wipe down the dinner table, then wash the dishes, and so on.) Reasoning about and accomplishing long-horizon tasks is a hallmark of human intelligence: as we learn and grow from infancy to adolescence, we quickly gain the fundamental principles of long-horizon task solving, such as making plans, coordinating with other humans, and learning from past failures in order to successfully accomplish a task.

Unfortunately, long-horizon problem solving skills that are second nature for humans remain well beyond the reach of our current robot learning systems. Why is this the case? To better understand what kinds of challenges prohibit effective and efficient long-horizon policy learning, let us explore a motivating example.

1.1 Motivating Example

Consider a warehouse that packs, organizes, and ships packages of varying sizes and types. In the warehouse, both autonomous robots and humans work together to accomplish a variety of tasks, including packaging goods, moving different boxes to storage locations within the warehouse, bringing shipments to outbound loading docks, and maintaining machinery. Beyond the fundamental control and dexterous capabilities required to perform these tasks, what kind of high-level reasoning do we require from our autonomous robots? We outline four fundamental capabilities, commonplace in human behavior but missing from state-of-the-art autonomous systems, as follows:

Capability 1: Learning from incremental feedback

Human beings rarely treat their tasks and goals as all-or-nothing. If a computer science undergraduate student has to write a basic operating system kernel, and they've implemented every necessity except for forking and memory stack overflow handling, they would probably not accept a zero as a grade on the assignment, even though the kernel is incomplete. Similarly, in our example, suppose a robot is learning to prepare a good for delivery. If the robot successfully packaged the good but failed to bring it to the correct loading dock, the robot should know that the task is 'partially complete' and understand which parts of its behavior should be positively and negatively reinforced. Perhaps most importantly, our robot should *remember* its partial progress through a task so it does not repeat redundant behavior.

The methodology of 'trial-and-error learning' in machine learning is most commonly represented by **Reinforcement Learning** (RL), where a policy interacts with its environment for some finite amount of time, receives a reward for its actions, updates its policy as a

result of the reward it received, and repeats (Sutton et al. 2018). The primary challenge for a machine learning practitioner in instantiating an RL agent is that of *reward design*: namely, in what states of the environment should the robot be given a positive reward, when should it be given a negative reward, and how much reward should be given? Reward design is often treated as an art rather than a science, and RL practitioners typically resort to the simplest design: they define a task for their robot (e.g., "prepare the good for delivery"), then give a reward of +1 if the robot successfully accomplishes the task, and zero otherwise.

A simple reward design, while easy to define for the robot, leads to a litany of issues in policy learning. Recall our partially complete example of a robot's behavior from earlier: under this reward scheme, the robot would receive no positive feedback from its behavior, even though it managed to accomplish part of the task correctly. Even if the robot accomplishes 99% of its task, it will still receive a reward of zero! Similarly, without explicitly defining partial progress of a task, policies must form an implicit plan of their own through a task, which proves to make the learning problem significantly more difficult (Jothimurugan et al. 2019).

Capability 2: Balancing completion with cost

In most cases, simply getting a task done is not enough. If our robot has to prepare a delicate vase for delivery by packaging it and moving it to the correct loading dock, it wouldn't be very useful if the robot knocks the package against the wall 400 times on the way to the loading dock and breaks the vase inside the box. Humans are constantly balancing 'completion' criteria (am I getting the task done?) with 'cost' criteria (am I doing the task in the safest, most efficient or smoothest way possible?).

When training robot policies to accomplish tasks through RL, such cost criteria often must be made explicit. Rewards offered in reinforcement learning are *discounted* over time, which means that a policy that gets a positive reward will receive a higher amount of that reward the faster it gets to that rewarding state. Discounting incentivizes efficiency, but efficient policies are not always what we want when deploying robots in the real world. If the most efficient path for our robot to get to the correct loading dock requires moving through multiple heavy machinery sites, we'd probably prefer our robot to take a smoother, less interfering path. In order to incentivize our robot to take the more preferred path, we can adjust our reward function to discourage moving through heavy machinery sites, i.e., by providing a negative reward any time the robot enters one of these sites. However, in making this adjustment, we may be hampering the policy's ability to learn. If our robot reaches a few heavy machinery sites early on in its learning process, it will be discouraged from moving around altogether and the end result will be a robot that just stays put so it doesn't incur any negative reward from moving around. How do we balance our cost criterion so that our learned robot policies still accomplish their original tasks?

Capability 3: Coordinating with other agents

Sufficiently complex tasks require humans to cooperate and coordinate with one another. In policy learning, the field of **Multi-Agent Reinforcement Learning** (MARL) studies how multiple autonomous agents can simultaneously learn cooperation in order to achieve tasks. In our warehouse, robots may have to work together to deliver a set of packages to different loading docks within a certain amount of allotted time. Such a task will require agents breaking down the delivery task into sub-tasks, assigning sub-tasks to each agent based on their environmental context and individual capabilities, and communicating when sub-tasks are complete if the task contains ordering constraints.

The aforementioned challenges of cooperation raise a number of potential pitfalls when learning policies for multiple cooperative agents. If some agents have stronger initial capabilities than others, less capable agents in the team may be disincentivized from acting in risk of interfering with the more capable agents. How do agents agree on a common communication protocol when working on separate sub-tasks? And in the worst case, if a task is not properly broken down for a team of agents, the overall cooperative behavior may not result in accomplishing the original task.

Capability 4: Autonomously recognizing and learning from past mistakes

Until now, we have discussed reinforcement learning as the primary means of policy learning for our robots. Recent approaches to robot learning have eschewed RL in favor of foundation-model-centric learning, where massive supervised datasets of robot behavior are instead learned from to distill generalist robot policies (Xiao et al. 2025). In doing so, robotics practitioners are able to leverage the language-conditioned internet-scale prior of foundation models (Radford et al. 2019) and deploy robots that can act in open-ended environments and respond to requests in natural language.

A major limitation with these robot foundation models (RFMs) is that they do not have a means of dynamically learning from their own past behavior in order to refine their policies for future attempts. Suppose a robot in our warehouse is using an RFM for end-to-end planning and control and is tasked with putting different goods into boxes that are appropriately sized for each. If our robot makes a mistake and puts a good into a box that is too small or too big, it has no recourse on how to identify their behavior as a failure, understand why it made a mistake, and learn from that mistake to avoid making it in the future.

1.2 Thesis Overview

In this thesis, we present solution concepts to each of the challenges outlined above. Central to our contributions is the intuition that using formal *symbolic representations* of task spec-

ifications and feedback mechanisms can help us facilitate long-horizon task learning. More specifically, symbolic representations provide a principled underlying structure that can be exploited by extensions to common policy learning approaches. This structure enables (1) shaping of reward functions that incentivize incremental progress through long-horizon tasks, (2) separate satisfaction conditions from cost conditions to ensure policy learning is both correct *and* optimal, (3) break down team-based tasks into individual subtasks to facilitate coordination, and (4) structure feedback to allow for principled assessment and learning from past mistakes.

When we discuss symbolic task representations in this thesis, we will most often be referring to *formal logical specifications* defined with propositional Boolean logic. We will briefly introduce propositional (Boolean) logic, which we will use as a means of explicitly defining complex, long-horizon tasks.

Formal Logic for Long-horizon Task Specification

Mathematical logic provides rigorous notation with precise semantics by which we can outline desirable or undesirable outcomes, which can be used to define tasks or properties for a computational system's behavior that we want it to abide by. Although there are many different formal logics that such *formal specifications* can be defined in, most logics are built upon the same fundamental building blocks of propositional logic, which we briefly discuss as follows:

An *atomic proposition* is a variable that takes on a Boolean truth value. We define an *alphabet* Σ as all possible combinations over a finite set of atomic propositions (AP); that is, $\Sigma = 2^{\text{AP}}$. For example, if $\text{AP} = \{a, b\}$, then $\Sigma = \{\{a, b\}, \{b\}, \{a\}, \{\}\}$. In our problem settings, atomic propositions will represent states or events in the environment that are of particular interest for our autonomous policies. Full (long-horizon) tasks will compose multiple atomic propositions (i.e., individual events) to specify tasks. For example, suppose we have three events: A, which indicates successfully packaging a good, B, which indicates successfully delivering the good to the loading dock, and C, which indicates if the package collides with any walls. We can use a formal language, such as Linear Temporal Logic (Pnueli 1977), to specify a task of "Do A, then B, while avoiding C."

A critical aspect of formal logic is that their Boolean nature provides a *precise* notion of task satisfaction. If a robot attempts a task and produces a trajectory of its behavior, we can formally 'check' that behavior against our formal task specification and receive a Boolean value that indicates whether or not we achieved the task. Receiving Boolean feedback on our behavior is crucial for ensuring that our behavior is fully correct, and is used to *formally verify* computational systems (Baier et al. 2008). Although there is a robust body of work on how we can formally verify hand-designed computational systems, much less work exists on how formal logic can be applied to learning-based systems. Formal logic was not designed with complex black-box learning systems in mind, and as a result, we must bridge the gaps between the precise, discrete semantics of formal logic and the continuous optimization of

machine learning to encode Boolean goals into a form suitable for policy learning. In this these, we will explore multiple avenues of building such bridges.

1.3 Contributions

In its totality, this thesis provides a suite of techniques that enables policy learning for complex, long-horizon tasks by bridging together formal logic, foundation models, and reinforcement learning. The high level principles of these works espouse the usage of symbolic structure to help policies learn incrementally, coordinate with teammates, and understand how to repair their own strategies when they make mistakes in the real world.

1. We create a theoretically principled optimization objective for reinforcement learning that balances correctness-critical task satisfaction conditions with cost conditions that outline how 'best' to accomplish a task (Chapter 3). In order to do so, we adopt Linear Temporal Logic (LTL)(Pnueli 1977) as our means of task specification, and leverage prior work to transform LTL specifications into *proxy* rewards that are readily optimized by RL approaches. In order to balance LTL tasks with external cost conditions, we pose the multi-objective problem as an instance of constrained optimization, requiring that a learned policy satisfy the original LTL task with maximum probability. Our constrained objective formulation allows us to analytically derive a lower bound on the 'ratio' between the LTL proxy reward and external reward to ensure that the resulting Lagrangian dual version of the original constrained objective preserves both correctness and cost-optimality when optimized by an RL policy.

2. We develop **Cycle Experience Replay** (CyclER), a method of reward shaping for LTL proxy reward functions, that alleviates the reward sparsity issue that plagues temporal logic-guided RL (Chapter 3). Following our earlier example, incremental progress for RL-based policy learning is difficult when rewards are *sparse*; that is, when a reward is only given after the entire task has been accomplished. CyclER exploits the underlying symbolic structure of LTL task specifications to find accepting *paths* and *cycles* that provide step-by-step plans for learning to solve complex, long-horizon tasks. CyclER reasons about all possible step-by-step plans that can feasibly achieve an LTL task, and counterfactually shapes past experience collected by a policy to give the maximum reward possible based on how much progress an agent for all possible accepting paths and cycles. Our experimental results demonstrate that CyclER-based reward shaping for LTL tasks in RL settings greatly improve the learnability of long-horizon tasks in complex control environments and loco-navigation tasks.

3. We present Learning Optimal Task Decompositions (LOTaD), a technique that learns multi-agent policies by decomposing a symbolic task specified by a deterministic finite automaton (Chapter 4). These automata, also referred to by Reward Machines (Toro Icarte et al. 2022) in the literature, can be decomposed such that a single *monolithic*

task can be broken down into sub-tasks that can then be assigned to each agent within a team. Importantly, prior literature has developed a formal notion of Reward Machine decomposition (Neary et al. 2021) that preserves correctness of the broken down monolithic task: in other words, accomplishing each of the decomposed tasks ensures that the overall task is accomplished as well. Such notions of correctness are critical when cooperating on complex, long horizon tasks, as discussed in our motivating example.

4. We visit the question of *where* formal task specifications come from in the first place. Learning task specifications from labeled examples and demonstrations is a rich space of ongoing research. We contribute Membership Respecting Preferences (MemRePs) (Chapter 5), a formalizing notion of preference-based ordering for demonstration-based specification learning. MemRePs introduce a new medium of specification learning: whereas prior literature would synthesize specifications strictly from user-provided positive or negative labeled demonstrations, MemRePs allow a user to identify pairwise preferences, and seamlessly integrates such preferences within the existing labeled example-based framework. Importantly, the MemRePs formalization is *concept-class agnostic*; to demonstrate this, we present instantiations of MemRePs in multiple concept class experimental settings and show that the combination of labeled examples and pairwise preferences can reduce the total number of labeled queries necessary to learn specifications.

5. Lastly, we turn our attention to robot foundation models and present Learning from Inference-Time ExecutioN (LITEN), a methodology for RFMs to learn their affordances from repeated interactions with their real-world environment (Chapter 6). Current RFMs, namely Vision-Language-Action models (Brohan et al. 2022), are *single-shot*: this means that if a language-conditioned policy fails to carry out an instructed task in the real world, it has no recourse to learn from its mistake and improve its performance in future attempts. To address this, the LITEN approach decouples reasoning from action, and introduces a separate foundation model to act as a "planner", only relying on the VLA as a low-level "controller." Through repeated trials, the VLM planner suggests sequences of sub-tasks for the VLA to execute, and based on those interactions, LITEN's VLM planner systematically and structurally analyzes the outcomes of previous interactions. This structured feedback mechanism generates meaningful takeaways, in natural language, that can then be used by the VLM planner to inform future attempts at solving the same task. Our experimental results show that LITEN demonstrates strong improvement over time, in comparison to existing self-refinement procedures for foundation models that are not attuned to the specific domain of robotics.

The primary aim for this body of work is to demonstrate the utility of symbolic structure when learning to solve complex, long-horizon tasks within modern policy learning frameworks. Although much of machine learning is predicated on minimizing the amount of rigid inductive bias when learning solutions, structure can provide great benefit, particularly

when tasks become sufficiently complex and exceed the limitations of our current suite of approaches.

Chapter 2

Background and Preliminaries

We begin by briefly introducing the fundamental concepts that underpin the contributions of this thesis. Specifically, we will outline two key topics: (1) task specification logics, which enable the precise definition of objectives for computational systems, and (2) policy learning techniques, which enable data-driven training of decision-making models.

2.1 Logical Task Specifications

As discussed in Chapter 1, the fundamental building blocks of propositional logic can be combined with higher-order operators in order to express more complicated specifications, such as temporally extended objectives. We operate over a set of atomic propositions AP, which represent specific properties or events in the environment that can be evaluated to a Boolean truth value (e.g., "the robot is holding the object"). We define an alphabet $\Sigma = 2^{\text{AP}}$, representing all possible truth assignments to these propositions at any given discrete timestep. A *trajectory* is a temporal sequence of truth assignments to our alphabet, which abstracts a robot's behavior into a sequence of discrete timesteps that capture which propositional variables of interest are true at a given point in time. We now present two formalisms used as specification languages in this thesis to identify desirable and undesirable trajectories: Deterministic Finite Automata and Linear Temporal Logic.

Deterministic Finite Automata and Reward Machines

A Deterministic Finite Automaton (DFA) is a finite state machine that accepts or rejects finite sequences of symbols (trajectories). Formally, a DFA is defined as a tuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where:

- Q is a finite set of states,
- $\Sigma = 2^{\text{AP}}$ is the finite alphabet,
- $\delta : Q \times \Sigma \to Q$ is the deterministic transition function,

- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of accepting states.

DFAs are functionally equivalent to regular expressions and serve as their operational form. For candidate trajectories of Boolean variables representing a robot's history of behavior, the DFA transitions between states based on the conjunction of variables observed at each timestep. A trajectory is deemed successful if it terminates in an accepting state $q \in F$, and unsuccessful if it terminates in a rejecting (or sink) state from which F is unreachable. For example, if a robot must pick up a box (event a) and then place it on a conveyor belt (event b), the DFA will transition from q_0 to an intermediate state upon observing $\{a\}$, and finally to an accepting state in F upon observing $\{b\}$.

Inspired by finite state machines, **Reward Machines (RMs)** (Toro Icarte et al. 2022) have emerged in concrete machine learning contexts to specify structured objectives. An RM operates similarly to a DFA but extends the framework by defining explicit reward functions over transitions or states. Formally, an RM can be defined as a tuple $\mathcal{R} = \langle U, u_0, \Sigma, \delta_u, \delta_r \rangle$, where U is a finite set of logical states, $u_0 \in U$ is the initial state, $\delta_u : U \times \Sigma \to U$ is the logical transition function, and $\delta_r : U \times U \to \mathbb{R}$ defines the reward received when transitioning between logical states.

This formulation makes RMs highly expressive as objective representations. Despite this expressivity, they are typically utilized in a manner akin to DFAs, often issuing a reward of $+1$ upon transitioning to a goal state, a negative reward if an undesirable clause is violated, and 0 otherwise. While DFAs and RMs are desirable for their semantic simplicity, they are inherently limited in their temporality; specifically, indefinite or infinite-horizon objectives cannot be specified with these standard formalisms. To address this, we turn to a more generalizable language.

Linear Temporal Logic

Linear Temporal Logic (LTL) (Pnueli 1977) provides a robust mathematical framework for specifying temporally extended objectives over a linear axis of time. Unlike DFAs, which are restricted to finite trajectories, LTL can express indefinite or infinite-horizon objectives that require a policy to act indefinitely to satisfy a condition.

The syntax of LTL is built upon atomic propositions $p \in \text{AP}$, standard Boolean connectives (such as negation $\neg$, conjunction $\wedge$, and disjunction $\vee$), and temporal operators. The minimal grammar for an LTL formula ϕ is given by:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \bigcirc\phi \mid \phi \mathcal{U} \phi$$

The temporal operators allow us to reason about how truth values change over the sequence of discrete timesteps in a trajectory. The primary operators and their intuitive meanings are as follows:

- **Next ($\bigcirc$):** Evaluates to true if the inner formula holds at the immediate *next* timestep of the trajectory. For example, $\bigcirc p$ asserts that proposition p will be true exactly one step from now.
- **Eventually ($\Diamond$):** Derived from the Until operator ($\Diamond\phi \equiv \text{True}\mathcal{U}\phi$), this evaluates to true if the formula holds at *some* current or future timestep. For example, $\Diamond$goal asserts that the robot will reach its goal state at some point, without specifying exactly when.
- **Globally ($\Box$):** Derived using negation and Eventually ($\Box\phi \equiv \neg\Diamond\neg\phi$), this evaluates to true if the formula holds at *all* current and future timesteps. For example, $\Box$safe asserts that the robot must maintain a safe state at every step of its trajectory.
- **Until ($\mathcal{U}$):** Evaluates to true if the first formula holds continuously *until* the second formula becomes true. For example, carry_box$\mathcal{U}$at_belt requires the robot to continuously carry a box until it arrives at the conveyor belt.

By composing these operators, LTL allows us to mathematically express the fundamental classes of desired computational behavior. *Safety* properties encode the intuition that "bad things never happen," commonly expressed using the Globally operator (e.g., $\Box\neg$collision). Conversely, *liveness* properties encode the intuition that "good things eventually happen," typically expressed using the Eventually operator (e.g., $\Diamond$task_complete). LTL excels at combining these concepts into highly complex, long-horizon specifications. For example, we can specify a reach-avoid task, where a robot must eventually reach a target while always avoiding a hazard ($\Diamond$target$\wedge\Box\neg$hazard), or a recurrent surveillance task, where a robot must repeatedly visit a specific room forever ($\Box\Diamond$visit_room).

To operationalize LTL formulae and make them checkable against these (potentially infinite) trajectories, they are standardly converted into automata. In reinforcement learning contexts, a common translation target is the **Limit Deterministic Büchi Automaton (LDBA)**. An LDBA shares the same structural tuple as a DFA, $\mathcal{A}_{LDBA} = \langle Q, \Sigma, \delta, q_0, F\rangle$, but operates over infinite words. The semantics of LDBAs differ primarily in their acceptance condition: an infinite trajectory satisfies the LDBA if it visits at least one accepting state $q \in F$ *infinitely often*. In practice, verifying this within a learning loop typically involves identifying a recurrent "loop" of behavior and ensuring that this loop successfully intersects with an accepting state while avoiding rejecting sink states. While many variants of LTL exist in the literature, we will primarily focus on this standard formalization, introducing quantitative semantics and specific shaping extensions later in Chapter 3.

2.2 Policy Learning Methods

We now introduce popular machine learning methodologies for training policies at a general level of abstraction. A *policy* fundamentally embodies an action-perception loop: an agent observes the state of the world, uses that observation to decide on an action, executes the

action, receives a new observation resulting from the environment's dynamics, and repeats the process. To learn dynamic policies, end-to-end learning predominantly follows one of two paradigms: Reinforcement Learning (RL), where a policy learns via trial-and-error to optimize a provided reward signal, or Imitation Learning (IL), where an agent learns to replicate expert demonstrations.

Reinforcement Learning

Reinforcement Learning (RL) problems are formalized as Markov Decision Processes (MDPs). An MDP is defined by the tuple $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$, where S is the state space, A is the action space, $P(s_{t+1} \mid s_t, a_t)$ represents the transition probability dynamics, $R(s_t, a_t, s_{t+1})$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. The behavior of an agent is defined by its policy $\pi(a_t \mid s_t)$, which maps states to a probability distribution over actions.

The fundamental mathematical objective of RL is to find an optimal policy π^* that maximizes the expected cumulative discounted reward, often referred to as the return. Formally, we seek to find a policy that maximizes the objective function $J(\pi)$:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right]$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ represents a trajectory generated by following policy π in the environment.

To optimize this objective, RL methods typically evaluate the quality of states or state-action pairs. The state-value function $V^\pi(s)$ measures the expected return starting from state s and following π thereafter. Similarly, the action-value function $Q^\pi(s, a)$ measures the expected return of taking action a in state s and subsequently following π. When the environment dynamics P and reward function R are explicitly known, and the state and action spaces are discrete, we can find optimal policies using exact Dynamic Programming (DP) algorithms like Value Iteration or Policy Iteration. These methods rely on the *Bellman backup*, a recursive operation that computes the value of a state based on the immediate reward and the discounted expected value of the subsequent state. By iteratively applying Bellman backups over the entire state space, DP converges to the optimal values (V^* or Q^*). The agent can then derive an optimal policy by simply acting greedily with respect to these maximized values.

However, in the complex robotic domains studied in this thesis, these tabular DP methods become fundamentally intractable. Real-world robotic tasks feature continuous, high-dimensional state and action spaces (e.g., continuous joint torques or high-dimensional camera observations). This leads to the "curse of dimensionality," where evaluating every possible state-action pair is impossible. Furthermore, the true environmental dynamics P are typically unknown to the robot.

To overcome these limitations, we turn to **Deep Reinforcement Learning (Deep RL)**, which leverages deep neural networks as universal function approximators to estimate

policies and value functions without requiring tabular iteration or prior knowledge of the transition dynamics. Modern Deep RL predominantly utilizes *Temporal Difference (TD) learning* and the *Actor-Critic* architecture to learn continuous control policies. TD learning is a model-free approach that updates value estimates based on other learned value estimates (bootstrapping) using transitions (s, a, r, s') sampled from environmental interaction.

In the Actor-Critic paradigm, the learning process is divided between two interacting neural networks:

- **The Critic:** Approximates the value function (e.g., $Q_\phi(s, a)$ parameterized by weights ϕ). It is trained to minimize the *critic loss*, which is typically the TD error: the squared difference between its current value prediction and a target value derived from the Bellman equation (e.g., $r + \gamma \max_{a'} Q_\phi(s', a')$).
- **The Actor:** Approximates the policy ($\pi_\theta(a \mid s)$ parameterized by weights θ). It is updated to minimize the *actor loss*, adjusting its parameters to increase the probability of selecting actions that the Critic evaluates as highly rewarding (i.e., actions that yield high Q-values or positive advantages).

By iteratively looping this action-perception process—collecting experiential transitions in the environment, updating the Critic to better evaluate the policy, and updating the Actor to choose better actions based on the Critic's gradient feedback—algorithms like Proximal Policy Optimization (PPO) Schulman et al. 2017 and Soft Actor-Critic (SAC) Haarnoja et al. 2018 can successfully learn robust behaviors in complex, unknown environments.

Imitation Learning and Robot Foundation Models

While Deep RL provides a mathematical framework for learning optimal behaviors, it is notoriously sample-inefficient. Training a policy entirely from scratch in the real world typically requires thousands, if not millions, of environmental interactions. This becomes highly impractical without either a high-fidelity simulator or a human continuously manually resetting the environment. Furthermore, random exploration during the early stages of RL poses severe safety risks, as an untrained robot could easily damage itself or its surroundings. To circumvent these issues, it is often advantageous to initialize a robot with a "policy prior" that provides a foundational understanding of task execution and safe operation before engaging in autonomous trial-and-error.

To acquire this prior, we turn to Imitation Learning (IL), where expert data (e.g., human teleoperation) is provided, and the policy learns to map observations directly to expert actions. For the purposes of this thesis, we focus primarily on **Behavioral Cloning (BC)**, which reduces imitation learning to a standard supervised learning problem. Given a dataset of expert demonstrations defined as $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$, the objective of BC is to find policy parameters θ that minimize the discrepancy between the policy's predicted action and the ground-truth expert action. Formally, we minimize the expected loss:

$$\mathcal{L}_{BC}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[L(\pi_\theta(s), a) \right]$$

where L is an appropriate supervised loss function, such as Mean Squared Error (MSE) for continuous action spaces or Cross-Entropy for discrete action spaces.

Recently, the advent of large foundation models—particularly Vision-Language Models (VLMs)—has revolutionized how we establish these policy priors. VLMs encapsulate a strong, language-conditioned understanding of visual semantics derived from internet-scale pre-training. By leveraging this semantic prior, researchers can apply large-scale supervised learning (BC) over massive datasets of multi-task robot teleoperation to create Robot Foundation Models (RFMs).

The most prominent of these RFMs are Vision-Language-Action (VLA) models, which adapt VLM architectures to output physical actions. At any given timestep, the input to a VLA typically consists of a natural language instruction defining the task, visual observations from the robot's cameras, and the robot's current proprioceptive state (e.g., joint configurations). The backbone of these models relies on the Transformer architecture, whose self-attention mechanisms are highly effective at processing these sequential, multimodal inputs over time. The VLA then autoregressively generates an output sequence that includes action tokens—mapping to a predefined action space such as end-effector poses or gripper states—thereby completing the action-perception loop.

While highly scaled BC serves as the foundational pre-training step to distill these generalist, language-conditioned policies, pure BC can be susceptible to compounding errors when deployed in the real world. As a result, post-training methods—such as targeted RL fine-tuning—are increasingly being explored to further refine policy robustness, correct out-of-distribution failures, and adapt these models to strict environmental constraints.

In the following chapters, we will reiterate many of these fundamentals and provide additional technical detail in order to tightly bridge the gap between formal logical specifications and modern, high-capacity policy learning.

Chapter 3

LTL-Constrained Policy Optimization with Cycle Experience Replay

A central challenge in reinforcement learning is that of designing reward functions for long-horizon tasks. Naïvely defined reward functions often fail to capture the incremental nature of multi-step tasks, leading to sparsity and poor policy learning. As a result, significant research effort has explored *Linear Temporal Logic* (LTL) as an alternative to Markovian reward functions for specifying objectives for reinforcement learning (RL) agents (Sadigh et al. 2014; Hasanbeig et al. 2018; Camacho et al. 2019; Wang et al. 2020a; Vaezipoor et al. 2021; Alur et al. 2022; De Giacomo et al. 2020; Voloshin et al. 2023). LTL provides a flexible language for defining objectives, or *specifications*, that are often not reducible to scalar Markovian rewards (Abel et al. 2021). Unlike typical Markovian reward functions that are defined only over observations in the environment, objectives defined in LTL are composable, easily transferred across environments, and offer a precise notion of satisfaction.

LTL specifications and Markovian reward functions have been used separately in a variety of RL settings, but few works consider both rewards *and* specifications in the same setting (Voloshin et al. 2022). The combination of the two is important: an LTL specification can define the meaning of achieving a task, and a reward function can be optimized to find the best way of achieving that task. Consider our previous example of robot motion planning in a warehouse: an LTL specification can describe the waypoints a robot should reach and obstacles it should avoid while navigating to a loading dock, and a reward function can optimize for factors like energy consumption, stability of motion, and so forth.

This section considers the problem setting of RL-based reward optimization under an LTL constraint. This constrained learning problem can be naturally formulated in an unconstrained form through a Lagrange-style relaxation (Le et al. 2019; Achiam et al. 2017) where the LTL constraint is represented by a *proxy* reward function. Some versions of this proxy have been introduced in prior literature (Hasanbeig et al. 2020; Voloshin et al. 2023; Camacho et al. 2019). However, these proxy rewards are sparse and difficult to optimize. As a result, learned policies in practice often end up ignoring the LTL constraint entirely and focus only on optimizing the reward function. A few existing works attempt to circumvent

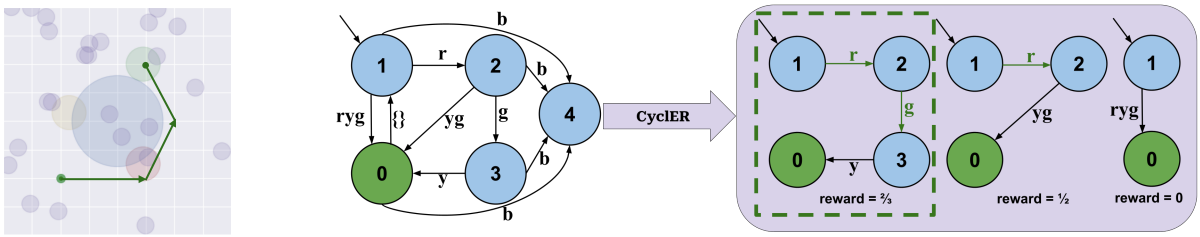


Figure 3.1: Left: The FlatWorld MDP and an example trajectory. Right: An LDBA for the LTL formula $\varphi = G(F(r)\&F(g)\&F(y))\&G(\neg b)$ (some edges omitted for readability); the accepting state 0 is coded in green. The CyclER method considers all accepting paths within an LDBA and selects the most reward-ful path for the trajectory to shape LTL reward. Unlike previous approaches (Camacho et al. 2019; Voloshin et al. 2023), CyclER offers dense reward, even without visiting the accepting state.

this sparsity issue by considering alternative temporal logics that provide a denser reward signal (Krasowski et al. 2023; Lavaei et al. 2020; Gundana et al. 2021) or by limiting their approach to discrete state spaces that can be solved exactly (Voloshin et al. 2022).

In this section, we introducing a novel reward-shaping proxy for LTL called *Cycle Experience Replay* (CyclER) that alleviates the reward-sparsity issue of previous LTL proxy rewards, allowing us to scale to continuous state and action spaces with an objective readily optimizable by deep RL (DRL) approaches. We briefly summarize the CyclER approach as follows. Given the (known) automaton structure representing an LTL objective, CyclER computes all possible infinite paths, or cycles, through the automaton that define accepting behaviors for our task. Then, when an agent collects a finite episode of experience in the world, CyclER counterfactually reasons over that experience by considering how much progress it made through each cycle. The cycle that the agent progressed through the furthest is then used to shape LTL reward. We provide a visualization of this approach in Figure 3.1.

Under certain assumptions, CyclER maintains theoretical guarantees on LTL optimality that are competitive with state-of-the-art LTL proxy rewards (Voloshin et al. 2023). A key advantage of CyclER is how it readily incorporates quantitative semantics (QS), a popular technique for reward design in temporal logic that has yet to be extended to infinite-horizon LTL tasks. Our empirical results demonstrate CyclER's effectiveness during policy learning.

In summary, this chapter makes the following contributions. We present a problem formulation for LTL-constrained policy optimization in the presence of continuous spaces and function approximators for use in deep RL. We propose a technique for this problem setting, CyclER, that alleviates the proxy reward sparsity issue and provides guarantees that ensure approximate optimality of LTL satisfaction. Third, we introduce a new way to leverage quantitative semantics in LTL reward shaping. Lastly, we present promising experimental results using CyclER in LTL-constrained optimization settings, outperforming

existing approaches.

3.1 Setting the Problem

Preliminaries

This subsection will build on our concepts of atomic propositions, alphabets, and state machines defined in chapter 2.

Labeled MDPs We formulate our environment as a labelled Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T^{\mathcal{M}}, d_0, \gamma, r, L^{\mathcal{M}})$, containing a state space $\mathcal{S}$, an action space $\mathcal{A}$, an *unknown* transition function, $T^{\mathcal{M}} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, an initial state distribution $d_0 \in \Delta(\mathcal{S})$, a discount factor $0 < \gamma < 1$, a reward function $r : \mathcal{S} \times \mathcal{A} \to [R_{\min}, R_{\max}]$, and a labelling function $L^{\mathcal{M}} : \mathcal{S} \to \Sigma$. The labelling function returns which atomic propositions in our set AP are true for a given MDP state.

As a running example, consider the "FlatWorld" MDP in Figure 3.1 (left), where an agent (represented by the green dot) can move around the environment and visit different colored regions of interest. The agent is given a reward of 1 every time it visits one of the small purple regions. Our alphabet in FlatWorld is defined as $\text{AP} = \{r, g, b, y\}$, corresponding to whether the agent is in the red, green, blue, or yellow region at any point in time.

Linear Temporal Logic (LTL) Linear Temporal Logic (Pnueli 1977) is a specification language that composes atomic propositions with logical and temporal operators to precisely define tasks. We use the symbol φ to refer to an LTL task specification, also called an LTL formula.

In LTL, specifications are constructed using both logical connectives: not ($\neg$), and ($\&$), and implies ($\rightarrow$); and temporal operators: next (X), repeatedly/always/globally (G), eventually (F), and until (U). For more detail on the exact semantics of LTL operators, see (Baier et al. 2008).

As an example, let us demonstrate task specifications in the "FlatWorld" environment from Figure 3.1 (left). LTL can easily define some simple objectives, such as safety $G(\neg b)$ (avoid the blue region), reachability $F(g)$ (reach the green region), or progress $F(y)\&X(F(r))$ (reach the yellow, then the red region). We can also combine operators to bring together these objectives into more complex specifications, such as $G(F(r)\&F(y)\&F(g))\&G(\neg b)$, which instructs an agent to oscillate amongst the red, yellow, and green regions indefinitely while avoiding the blue region.

In order to determine the logical satisfaction of an LTL specification, we can transform it into a specialized automaton called a *Limit Deterministic Büchi Automaton (LDBA)*. See (Sickert et al. 2016; Hahn et al. 2013; Křetínskỳ et al. 2018) for details on how LTL specifications can be transformed into semantically equivalent LDBA.

More precisely, a (de-generalized) LDBA is a tuple $\mathbb{B} = (\mathcal{B}, \Sigma, T^{\mathbb{B}}, \mathcal{B}^*, \mathcal{E}, b_{-1})$ with a set of states $\mathcal{B}$, the alphabet Σ of predicates ν that defines deterministic transitions in the automaton, a transition function $T^{\mathbb{B}} : \mathcal{B} \times (2^{\Sigma} \cup \mathcal{E}) \to \mathcal{B}$, a set of accepting states $\mathcal{B}^*$, and an initial state b_{-1}. An LDBA has separate deterministic and nondeterministic components $\mathcal{B} = \mathcal{B}_D \cup \mathcal{B}_N$, such that $\mathcal{B}^* \subseteq \mathcal{B}_D$, and for $b \in \mathcal{B}_D$, $x \in \Sigma$ then $T^{\mathbb{B}}(b, x) \subseteq \mathcal{B}_D$. $\mathcal{E}$ is a set of "jump" actions, also known as epsilon-transitions, for $b \in \mathcal{B}_N$ that transitions to $\mathcal{B}_D$ without evaluating any atomic propositions. A path $\xi = (b_0, b_1, \dots)$ is a sequence of states in $\mathcal{B}$ reached through successive transitions under $T^{\mathbb{B}}$, not including the initial state b_{-1}, as the labeling function $L^{\mathcal{M}}(s_0)$ transitions the state of $\mathbb{B}$ to b_0 to be consistent with the initial state of the MDP.

Definition 3.1.1 (Acceptance of ξ)**.** *We accept a path $\xi = (b_0, b_1, \dots)$ if an accepting state of the Büchi automaton is visited infinitely often by ξ.*

Problem Statement

We would like to learn a policy that produces satisfactory (accepting) trajectories with respect to a given LTL formula φ while maximizing r, the reward function from the MDP. Before we define our formal problem statement, we introduce more notation:

Definition 3.1.2 (Product MDP)**.** *A **product MDP** synchronizes the MDP with an LDBA. Specifically, let $\mathcal{M}^{\varphi}$ be an MDP with state space $\mathcal{S}^{\varphi} = \mathcal{S} \times \mathcal{B}$. Policies over our product MDP space can be defined as $\pi : \mathcal{S}^{\varphi} \to \Delta(\mathcal{A}^{\varphi})$, where our new set of actions combine $\mathcal{A}^{\varphi}((s, b)) = \mathcal{A}(s) \cup \mathcal{E}$, to include the jump transitions in $\mathbb{B}$ as possible actions. We define the space of all possible policies as Π. The new probabilistic transition relation of our product MDP is defined as:*

$$T(s, b, a, s', b') = \begin{cases} T^{\mathcal{M}}(s, a, s') & a \in \mathcal{A}(s), b' = T^{\mathbb{B}}(b, L(s')) \\ 1 & a \in \mathcal{E}, b' = T^{\mathbb{B}}(b, a), s = s' \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

A policy generates trajectories $\tau = ((s_0, b_0, a_0), (s_1, b_1, a_1), \dots)$ in the product MDP. Define $\mathcal{R}(\tau) \equiv \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ as the total reward along a trajectory τ.

Definition 3.1.3 (Trajectory acceptance)**.** *A trajectory is said to be **accepting** with respect to φ ($\tau \models \varphi$, or "φ accepts τ") if there exists some $b \in \mathcal{B}^*$ that is visited infinitely often.*

Definition 3.1.4 (Policy satisfaction)**.** *A policy $\pi \in \Pi$ satisfies φ with some probability $\mathbb{P}[\pi \models \varphi] = \mathbb{E}_{\tau \sim \mathcal{M}_{\pi}^{\varphi}}[\mathbf{1}_{\tau \models \varphi}]$. Here, $\mathbf{1}$ is an indicator variable that checks whether or not a trajectory τ is accepted by φ, and $\mathcal{M}_{\pi}^{\varphi}$ is the distribution of trajectories induced by policy π in a product MDP $\mathcal{M}^{\varphi}$.*

Definition 3.1.5 (Probability-optimal policies)**.** *We will denote* Π^* *as the set of policies that maximize the probability of satisfaction with respect to* φ*; that is, the policies that have the highest probability of producing an accepted trajectory:* $\Pi^* = \{\pi \in \Pi | \mathbb{P}[\pi \models \varphi] = \max_{\pi' \in \Pi} \mathbb{P}[\pi' \models \varphi]\}$.

Our aim is to find a policy in the probability-optimal set Π^* that collects the largest expected cumulative discounted reward. We state this constrained objective formally as follows:

$$\pi^* \in \text{argmax}_{\pi \in \Pi^*} \mathbb{E}_{\tau \sim \mathcal{M}_\pi^\varphi} \big[\mathcal{R}(\tau)\big] \tag{3.2}$$

For notational convenience, we will refer to the MDP value function as $\mathcal{R}_\pi \equiv \mathbb{E}_{\tau \sim \mathcal{M}_\pi^\varphi} \big[\mathcal{R}(\tau)\big]$.

In certain cases, the probability-optimal set of policies Π^* may be empty; consequently, a solution to 3.2 may not exist. In section 3.2, we introduce a proxy objective with a similar potential of non-existence and discuss how our ultimate optimization objective behaves in this setting.

To align with our intended applications towards deep RL, we consider stochastic, memoryless policies over the product MDP. Such policies are capable of capturing probability-optimal policies for a given LTL specification if an optimal policy exists (Bozkurt et al. 2020; Voloshin et al. 2022).

3.2 LTL-Constrained Policy Optimization: Theory

Finding a policy within Π^* is, in general, not tractable: an LTL constraint φ is defined over infinite-length trajectories but policy rollouts in practice produce only finite-length trajectories (Yang et al. 2022). We adopt *eventual discounting* (Voloshin et al. 2023), a common approach in the existing literature which aims to optimize a proxy value function that approximates the satisfaction of φ. Eventual discounting is defined as:

$$V_\pi = \mathop{\mathbb{E}}_{\tau \sim \mathcal{M}_\pi^\varphi} \left[\sum_{t=0}^{\infty} \Gamma_t r_{\text{LTL}}(b_t) \right], \qquad \Gamma_t = \gamma_\varphi^{j_t}, \quad r_{\text{LTL}}(b_t) = \begin{cases} 1 & \text{if } (b_t \in \mathcal{B}^*) \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where $j_t = \sum_{k=0}^{t} r_{\text{LTL}}(b_k)$ counts how many times the set $\mathbb{B}^*$ has been visited (up to and including the current timestep). Notably, eventual discounting does not discount based on the amount of time between visits to an accepting state. A policy that maximizes this eventual discounting reward is approximately probability-optimal with respect to φ when γ_φ, the discounting factor associated with φ, is selected properly (see Theorem 4.2 in (Voloshin et al. 2023) for an exact bound).

As a result of eventual discounting, we can replace Π^* in objective 3.2 with the set of policies that maximize V_π. Let $V_{\max} = \max_{\pi \in \Pi} V_\pi$ be the maximal value.

$$\pi^* = \mathop{\text{argmax}}_{\pi \in \{\pi \in \Pi | V_\pi = V_{\max}\}} \big[\mathcal{R}_\pi\big] \tag{3.4}$$

We now form the Lagrangian dual of objective 3.4 as $\pi^* = \min_\lambda \text{argmax}_{\pi\in\Pi}\big[\mathcal{R}_\pi + \lambda(V_\pi - V_{\max})\big]$. In theorem 3.2.2 we show that because we only care about constraint-maximizing policies, there exists $\lambda^* \in \mathbb{R}$ such that solving the inner maximization of the Lagrangian dual must be constraint optimal for any fixed $\lambda > \lambda^*$. Intuitively, the higher λ is, the more our learned policy will account for V_π during optimization until the constraint must be satisfied. At that point, because we are already achieving the maximum possible V_π, any additional lift will only come from maximizing over the MDP value $\mathcal{R}$, even if we continue to increase λ. With this observation, we can form an unconstrained objective function from objective 3.4 to be the following:

$$\pi^* = \arg\max_{\pi\in\Pi}\big[\mathcal{R}_\pi + \lambda V_\pi\big] \tag{3.5}$$

where we have dropped the dependence on $V_{\max}$ since it is a constant and fixed $\lambda > \lambda^*$. We show that under certain assumptions, an exact value for λ^* can be found to ensure that a policy that maximizes eq. 3.5 will certainly maximize V_π.

Assumption 3.2.1. *There exists a positive nonzero gap $\epsilon > 0$ between the value V_π of policies in $\pi \in \Pi^*$ and the highest-value policies that are not; that is, $V_{\max} - \max_{\pi\in(\Pi\setminus\Pi^*)}(V_\pi) > \epsilon$.*

Theorem 3.2.2. *Under Assumption 3.2.1, for any choice of $\lambda > \frac{R_{\max}-R_{\min}}{\epsilon(1-\gamma)}$, the solution to objective 3.5 must be a solution to objective 3.4.*

Proof for Theorem 3.2.2

Proof. Consider two policies: (1) $\pi \in \Pi \setminus \Pi^*$, which does not achieve $V_{\max}$, (2) $\tilde{\pi} \in \Pi^*$, achieving $V_{\max}$. Let $R_{\max}$ and $R_{\min}$ be upper and lower bounds on the maximum and minimum achievable single-step reward in $\mathcal{M}$, respectively. Evaluating objective 3.5 for both of these policies satisfies the following series of inequalities:

$$\mathcal{R}_{\tilde{\pi}} + \lambda V_{\tilde{\pi}} \overset{(a)}{\geq} \frac{R_{\min}}{1-\gamma} + \lambda(V_\pi + \epsilon) \overset{(b)}{\geq} \frac{R_{\max}}{1-\gamma} + \lambda V_\pi \overset{(c)}{\geq} \mathcal{R}_\pi + \lambda V_\pi$$

where (a) follows from assumption 3.2.1 and bounding the worst-case MDP value, (b) follows from selecting $\lambda > \frac{R_{\max}-R_{\min}}{\epsilon(1-\gamma)} (\equiv \lambda^*)$, (c) follows since the highest MDP value achievable by π must be upper bounded by the best-case MDP value.

As a consequence of $(a-c)$ we see that policies achieving $V_{\max}$ are preferred by objective 3.5. Consider $\pi^* \in \Pi^*$, the solution to objective 3.5. Thus, since $\pi^* \in \Pi^*$, then π^* must also achieve $V_{\pi^*} = V_{\tilde{\pi}} = V_{\max}$. Therefore, in comparing objective 3.5 for both π^* and $\tilde{\pi}$ it follows immediately that $\mathcal{R}_{\pi^*} \geq \mathcal{R}_{\tilde{\pi}}$ since π^* is optimal for objective 3.5. Since the choice of $\tilde{\pi}$ is arbitrary, we have shown that π^* is also a solution to objective 3.4. □

We note that Assumption 3.2.1 can be found in previous literature (Voloshin et al. 2023) and serves as a sufficient but not necessary condition for our results.

On the existence of Assumption 3.2.1

If we are restricted to stationary policies and the space of policies Π is finite, then assumption 3.2.1 will always hold. A finite space of policies can be enumerated over, and we can take the difference between the optimal and next-best policies to find ϵ. As an example, consider a toy MDP with a continuous, 1-dimensional state space $[0, 1]$ and continuous, 1-dimensional action space $[0, 1]$, where the transition function determines the next state as the agent's action, i.e. $T^{\mathcal{M}}(s, a, s') = a$. Suppose we are given a task specification $G(F(1))$. Under this specification, the agent will receive a reward of 1 every time it outputs 1, and 0 otherwise.

Consider a finite-sized policy class Π of just two deterministic policies: π_0 that only outputs 0, and π_1 that only outputs 1. Here, assumption 3.2.1 holds even in this continuous space. However, assumption 3.2.1 is not limited to just finite-sized policy classes. Consider an infinite-sized Π, where one policy in Π, called π^*, always outputs 1, and all other policies are Gaussian policies with $\sigma = 0.0001$ and μ uniformly sampled from the interval $[0, 0.0001]$. Here, even when Π is infinite and contains stochastic policies in a continuous space, assumption B.1 holds.

Although the aforementioned examples are toyish, they demonstrate that although assumption 3.2.1 is always true when Π is finite, this is not the only case. Further characterization for when the assumption holds is left for future work.

On the existence of a solution to 3.4

In our definition of Π^* in 3.1.5 and the formulation of our objective in 3.2, it is possible that no solution to 3.2 exists in settings where Π^* is empty. This non-existence issue reoccurs in objective 3.4, where it is possible that no policies exist that achieve $V_{\max}$. In all of these cases, non-existence is a result of an infinite-sized Π where a sequence of policies exists in Π that come increasingly arbitrarily close to achieving $V_{\max}$ (or, in the case of 3.2, to achieving the optimal probability of satisfying φ), without ever reaching the maximum value.

In these cases, we clarify what a policy that optimizes our true objective 3.5 is actually achieving. Recall that we ultimately aim to optimize a proxy objective of $\mathcal{R}_\pi + \lambda V_\pi$, under a sufficiently large λ. If the set Π^* is empty, then by definition, there does not exist a "gap" between policies in Π^* and policies outside of it. In other words, the value of ϵ is 0, and Assumption 3.2.1 does not hold. As a result, as λ tends to infinity, the sequence of policies that is increasingly optimal with respect to $\mathcal{R}_\pi + \lambda V_\pi$ will always prioritize increasing V_π over $\mathcal{R}_\pi$. In other words, if assumption 3.2.1 does not hold, optimizing 3.5 will continually try to improve the proxy reward for LTL satisfaction and will ignore resulting changes to the MDP reward $\mathcal{R}_\pi$.

It is theoretically possible that subsequent policies along the aforementioned sequence have arbitrarily different $\mathcal{R}_\pi$, which is undesirable. However, in practice, this does not tend to be the case, and policies that learn to optimize objective 3.5 exhibit behavior that is apparently both LTL-satisfying and performant in MDP reward, as evidenced by our

experiments in Section 3.4. Generally, values for $\mathcal{R}_\pi$ are not highly unstable along the sequence of policies that are increasingly optimal with respect to V_π, and allowing λ to serve as a hyperparameter effectively enables a tradeoff to value $\mathcal{R}_\pi$ against V_π (see Section 3.4).

As mentioned earlier, the probability-optimal set of policies with respect to φ may be empty. The same is true for our updated definition of Π^* that contain policies that achieve $V_{\max}$. In the case of this non-existence, Assumption 3.2.1 does not hold, and a policy that optimizes 3.5 will prioritize improving V_π at the potential expense of $\mathcal{R}_\pi$. We provide an extended discussion of this consequence in Section 3.2.

Empirical Considerations. Since the conditions for Assumption 3.2.1 are often unknown, there may not be a verifiable way to know that that our learned policy is maximizing V_π. Because of this, we will treat λ as a tunable hyperparameter that allows a user to trade off the relative importance of empirically satisfying the LTL constraint. There are a number of strategies one can use to find an appropriate λ: for example, one can iteratively increase λ until a desired LTL reward is achieved. In our experiments, we show an example of this trade off, and notice that the trade off lessens in severity once λ exceeds a value that enables learning LTL-satisfying policies (table 3.6).

3.3 Cycle Experience Replay (CyclER)

To distinguish between the MDP's reward function and the eventual-discounting proxy reward in 3.3, we write the MDP reward function $r(s, a)$ as $r_{\text{MDP}}(s, a)$. In Deep RL settings, we maximize objective 3.5 using the reward function $r_{\text{DUAL}}(s_t, b_t, a_t) = \gamma^t r_{\text{MDP}}(s_t, a_t) + \Gamma_t \lambda r_{\text{LTL}}(b_t)$.

However, optimizing objective 3.5 is challenging due to the sparsity of r_{LTL}. r_{LTL} is nonzero only when an accepting state in $\mathbb{B}$ is visited, which may require a long, precise sequence of actions.

Consider the FlatWorld MDP and LDBA in Figure 3.1. The MDP's reward function incentivizes visiting the small purple regions in the world. Under r_{LTL}, a policy will receive no reward until it completes the entire task of avoiding blue and visiting the red, yellow, and green regions through random exploration. If r_{MDP} is dense, a policy may fall into an unsatisfactory 'local optimum' by optimizing for r_{MDP} it receives early during learning, and ignore r_{LTL} entirely. In Figure 3.2, we see that a policy trained on r_{DUAL} makes such an error.

We seek to address this shortcoming by *automatically* shaping r_{LTL} so that a more dense reward for φ is available during training. Below, we present our approach, which exploits the known structure of the LDBA $\mathbb{B}$ and cycles within $\mathbb{B}$ that visit accepting states.

Rewarding Accepting Cycles in $\mathbb{B}$

By definition of LTL satisfaction (def. 3.1.3), a trajectory must repeatedly visit an accepting state b^* in an LDBA. In the context of the automaton itself, that means that an accepting trajectory will traverse an *accepting path* from the initial state to an accepting state, and then repeatedly traverse *accepting cycles* within $\mathbb{B}$ that continually visit accepting states.

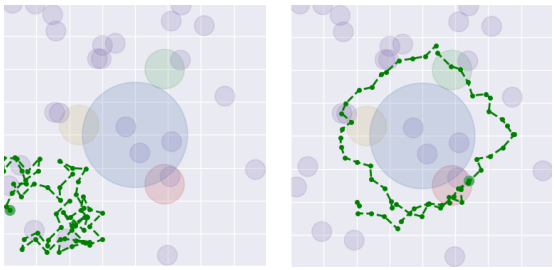


Figure 3.2: Trajectories from unshaped r_{DUAL} (left) and CyclER r_{DUAL} (right) for the formula $G(F(r)\&F(g)\&F(y))$ & $G(\neg b)$.

Definition 3.3.1 (Accepting Initial Path (AIP))**.** *An accepting initial path in $\mathbb{B}$ is a set of valid transitions (b_i, ν, b_j) (i.e., the predicate ν that transitions b_i to b_j) in $\mathbb{B}$ that starts at the initial state b_{-1} and ends at an accepting state $b_k^* \in \mathcal{B}^*$.*

Definition 3.3.2 (Accepting Cycle (AC))**.** *An accepting cycle in $\mathbb{B}$ is a set of valid transitions (b_i, ν, b_j) in $\mathbb{B}$ that start and end at accepting states $b_k^*, b_l^* \in \mathcal{B}^*$.*[1]

Our key insight is that we can use accepting paths and cycles in $\mathbb{B}$ to shape r_{LTL}. Instead of only providing reward when an accepting state in $\mathbb{B}$ is visited (as per previous approaches e.g. (Voloshin et al. 2023)), we reward progress within an accepting path or cycle. In our example from fig 3.1, if we reward each transition in the initial path $\{1, 2, 3, 0\}$ and the cycle with the same states, the agent would receive rewards for visiting the red region, then yellow, then green, then for returning to red, and so on.

Multiple accepting paths and cycles may exist in $\mathbb{B}$. The path and cycle that is used to shape r_{LTL} cannot be picked arbitrarily, since they may be infeasible under the dynamics of the MDP. For example, the cycle $\{1, 2, 0\}$ in Figure 3.1 cannot effectively shape r_{LTL} because it is impossible to be both in the yellow and green regions at the same time.

Reward Shaping with CyclER

CyclER is a reward function that automatically selects paths and cycles to shape r_{LTL} based on collected experience.

Definition 3.3.3 (Minimal AIP (MAIP))**.** *A minimal accepting initial path for accepting state b_k^* is an AIP that does not contain a subcycle for any node b_i in the path where $b_i \neq b_k^*$.*

Definition 3.3.4 (Minimal AC (MAC))**.** *A minimal accepting cycle c for accepting states b_k^* and b_l^* is an AC that does not contain a subcycle for any node b_i in the cycle where $b_i \notin \{b_k^*, b_l^*\}$.*

[1] Our usage of the word "cycle" is not a cycle in the traditional sense of graph search, but instead refers to paths that connect two accepting states in $\mathbb{B}$ (allowing for "cyclical" acceptance).

Algorithm 1: Find Minimal Accepting Initial Paths and Cycles (**FindMAIPsAndMACs**)

Input: LDBA $\mathbb{B}$, accepting states set $\mathcal{B}^*$
Initialize $\mathcal{C}$ to an empty set;
Initialize $\mathcal{P}$ to an empty set;
DFS(b_{-1}, {}, $\mathcal{P}$);
foreach *accepting state* $b^* \in \mathcal{B}^*$ **do**
    Initialize *visited* to an empty set;
    Initialize C to an empty set;
    DFS(b^*, {}, C);
    Add C to $\mathcal{C}$;
return $\mathcal{C}$, $\mathcal{P}$

Algorithm 2: DFS (Helper for Alg. 1)

Input: Starting node b, Path p, set S
Add node b to *visited*;
foreach *Outgoing transition* (b, ν, b') *from* b **do**
    if $b' \in \mathcal{B}^*$ **then**
        Add the transition (b, ν, b') to p;
        Add p to S;
    else
        if $b' \notin$ *visited* **then**
            Add the transition (b, ν, b') to p;
            DFS(b', p, S);
Remove node b from *visited*;

We provide CyclER with all MAIPs and MACs in an LDBA using Depth-First Search with backtracking. In algorithms 1 and 2, we include the psuedocode for finding minimal accepting initial paths and minimal accepting cycles in a given $\mathbb{B}$, which constitute the sets $\mathcal{P}$ and $\mathcal{C}$ respectively for usage in algorithm 3.

We let $\mathcal{P}$ and $\mathcal{C}$ define the set of MAIPs and MACs, respectively.

We also maintain a frontier e of visited transitions in $\mathbb{B}$ at each timestep in a trajectory that ensures reward will only be given once per transition until an accepting state is visited. To understand the need for our we show via example that the existence of non-accepting cycles in $\mathbb{B}$ may allow for trajectories that infinitely take transitions in a MAC or MAIP without ever visiting an accepting state.

Consider the accepting cycle $\{3, 1, 2\}$ in the partial automaton in Figure 3.3. Although this cycle is a MAC, there does exist a separate cycle starting and ending at state 1 (i.e. the cycle $\{1, 2, 0\}$.) If we give reward every time a transition in the cycle $\{3, 1, 2\}$ is taken, a policy may be able to collect infinite reward without ever visiting an accepting state. For example, in Figure 3.3, a path $\{1, 2, 0, 1, 2, 0 \dots\}$ would infinitely take transitions in a MAC,

and therefore collect infinite reward without ever visiting the accepting state 3. Our visited frontier e will ensure that rewards will only be given once per transition until an accepting state is visited.

In particular, we set $e[(b_i, \nu, b_i)] = 1$ when a transition (b_i, ν, b_i) is taken and reset all $e \equiv 0$ when $b_j \in \mathcal{B}^*$. Policies that use CyclER observe s, b, and e as their current state.

Now we describe the CyclER reward computation. We first collect a full trajectory τ induced by $\mathcal{M}_\pi^\varphi$ for a given policy π. Then, at each timestep t from 0 to $|\tau| - 1$, we compute $r_\mathcal{C}$ for every path in $\mathcal{P}$ if we have not yet visited an accepting state, or every cycle in $\mathcal{C}$ if we have. We will abuse notation slightly and use c to refer to elements in either $\mathcal{P}$ or $\mathcal{C}$:

$$r_\mathcal{C}(s, b, s', b', e, c) = \begin{cases} \frac{1}{|c|} & \text{if } (b, L^\mathcal{M}(s'), b') \in c \text{ and } e[b, L^\mathcal{M}(s'), b'] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

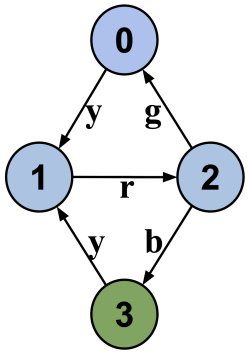


Figure 3.3: A partial Büchi automaton that necessitates a visited frontier.

This function rewards every transition taken in a given c. In other words, when an agent "gets closer" to an accepting state by progressing along a path or cycle, we reward that progress. Importantly, we do not reward transitions taken in a given c more than once per visit to an accepting state. In other words, a trajectory that repeatedly takes transitions in c but never reaches an accepting state b^* will only receive reward for the first instance of each transition taken in c. To account for c of varying length, rewards are normalized by the length $|c|$.

Algorithm 3: Cycle Experience Replay (**CyclER**)

Input: Trajectory τ, $\mathcal{B}^*$, cycles $\mathcal{C}$, paths $\mathcal{P}$, Labelling function $L^{\mathcal{M}}$
Initialize matrix $R_{\mathcal{C}}$ size $\max(|\mathcal{C}|, |\mathcal{P}|) \times (|\tau| - 1)$;
Initialize r_{CyclER} to an array of size $(|\tau| - 1)$;
Initialize $j = 0$;
foreach $t = 0, \ldots, |\tau| - 1$ **do**
  if $j = 0$ **then**
    foreach *Path* $p_i \in \mathcal{P}$ **do**
      $R_{\mathcal{C}}[i, t] = r_{\mathcal{C}}(b_t, s_{t+1}, b_{t+1}, e_t, p_i)$
  else
    foreach *Cycle* $c_i \in \mathcal{C}$ **do**
      $R_{\mathcal{C}}[i, t] = r_{\mathcal{C}}(b_t, s_{t+1}, b_{t+1}, e_t, c_i)$
  Set $e[b_t, L^{\mathcal{M}}(s_{t+1}), b_{t+1}] = 1$;
  if $b_{t+1} \in \mathcal{B}^*$ *or* $t + 1 = |\tau|$ **then**
    Select $i = \text{argmax}_{i \in |C|}(\sum_{j'=j}^{t} R_{\mathcal{C}}[i, j'])$;
    foreach t' *from* j *to* $t + 1$ **do**
      $r_{\text{CyclER}}[t'] = R_{\mathcal{C}}[i, t']$
    $j = t + 1$;
    Set all $e \equiv 0$;
return r_{CyclER};

If we visit an accepting state b^* or reach the end of a trajectory, we retroactively assign rewards to the timesteps that preceded this point, up to the most recent accepting state visit (if one exists). Assigned rewards correspond to the cycle with the highest total reward for that partial trajectory. Put simply, CyclER picks the 'best' cycle for a partial trajectory and uses it to shape reward. Even if a trajectory does not manage to visit an accepting state, CyclER will still provide reward if it was able to take *any* transition along *any* MAIP. The CyclER algorithm is formally outlined in Algorithm 3.

Let us walk through an example of using CyclER to shape reward by referring to the specification φ and trajectory in the FlatWorld MDP from Figure 3.1. Here, the agent produces a trajectory of length 4, indicated by the end of each line segment, starting from its initial state. The corresponding LDBA path for this trajectory ξ is $(1, 2, 2, 3)$. Under the unshaped (non-CyclER) proxy reward defined in 3.3, no reward would be gained from this trajectory, as the accepting state of the LDBA is not visited. Instead, let us consider how CyclER shapes the reward for each transition in this trajectory. There are three MAIPs in this LDBA: $\{(1, ryg, 0)\}$, $\{(1, r, 2), (2, yg, 0)\}$, and $\{(1, r, 2), (2, g, 3), (3, y, 0)\}$. For each of these MAIPs, CyclER will retroactively assign rewards to each transition within the trajectory. Under the MAIP $\{(1, ryg, 0)\}$, no transitions will receive reward, as the predicate ryg is not achieved. For $\{(1, r, 2), (2, yg, 0)\}$, the transition to the second timestep of the trajectory will receive a reward of $\frac{1}{2}$ for visiting the red region. Lastly, for the MAIP $\{(1, r, 2), (2, g, 3), (3, y, 0)\}$, both the transitions to the second and fourth timesteps will receive a reward of $\frac{1}{3}$, for visiting the red and green regions, respectively. Therefore, CyclER will choose to shape the trajectory's rewards using the MAIP $\{(1, r, 2), (2, g, 3), (3, y, 0)\}$, as it results in the highest cumulative

reward assigned to the entire trajectory.

We denote the rewards returned from Algorithm 3 as r_{CyclER}. r_{CyclER} can be used in place of the unshaped r_{LTL} in function 3.3 to provide a more dense reward for τ.

Theorem 3.3.1 (Informal)**.** *By replacing r_{LTL} with r_{CyclER} in 3.3, the solution to problem 3.5 remains (approximately) probability optimal in satisfying the LTL formula φ.*

Proof for Theorem 3.3.1

We start with some notation. Let r^{τ}_{CyclER} represent the reward function for a trajectory τ that are returned by the execution of Alg. 3. Let $T(\tau)$ be the set of timesteps when an accepting state in $\mathbb{B}$ is visited for a trajectory. We write the value function for CyclER, letting Γ_t be the same function as defined in function 3.3:

Assumption 3.3.2. *Suppose $T_{\max} = \max_{\pi\in\Pi} \mathbb{E}_{\tau\sim\mathcal{M}^P_\pi}\left[T(\tau)\middle|\tau \not\models \varphi\right] < M$, there is a uniform bound on the last time a bad (non-accepting) trajectory visits an accepting state across all bad trajectories induced by any policy.*

Lemma 3.3.3. *Under Assumption 3.3.2, for any $\pi \in \Pi$ and $\epsilon > 0$ we have*

$$|(1-\gamma)V^{cyc}_{\pi} - \mathbb{P}[\pi \models \varphi]| \leq \epsilon$$

when $\gamma \geq (1-\epsilon)^{\frac{1}{M+1}}$ is chosen appropriately.

Proof. We follow the proof style of Lemma 4.1 from (Voloshin et al. 2023). Let $\mathbb{P}[\pi \models \varphi] = p$ be the probability that π satisfies the LTL specification φ. Recall the value function

$$V^{\text{cyc}}_{\pi} = \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}^{\varphi}_{\pi}}\left[\sum_{t=0}^{\infty}\Gamma_t r^{\tau}_{\text{CyclER}}[t]\right]$$

Let $T(\tau)_{(i)}$ be the (random) i-th visit to an accepting state in $\mathbb{B}$. Let $T(\tau)_{(0)}, T(\tau)_{(-1)}$ refer to the first and last visit, respectively.

Because $r^{\tau}_{\text{CyclER}}[t] = \frac{1}{|c|}$ only when a transition in a cycle is taken (and only once per that transition) and the distance between successive visits to an accepting state is $|c|$ then at most γ^i reward is accumulated between successive visits to an accepting state. In other words $\mathbb{E}_{\tau\sim\mathcal{M}^{\varphi}_{\pi}}\left[\sum_{t=T_{(i)}+1}^{T_{(i+1)}}\Gamma_t r^{\tau}_{\text{CyclER}}[t]\right] = \gamma^i$ and therefore $V^{\text{cyc}}_{\pi} \leq \frac{1}{1-\gamma}$.

Further, every trajectory τ is decomposable into (1) the partial trajectory up to the first visit (ie. at time $T(\tau)_{(0)}$), the partial trajectory between the first and last visit (ie. between time $T(\tau)_{(0)}$ and $T(\tau)_{(-1)}$), and (3) the remainder of the trajectory. For trajectories that satisfy the LTL specification, $T(\tau)_{(-1)} = \infty$, otherwise $T(\tau)_{(-1)} \leq M$, finite and bounded (by Assumption 3.3.2). For ease of notation, we omit the dependence of T on τ (i.e. we write $T(\tau)_{(0)}$ as $T_{(0)}$). By linearity of expectation, we can rewrite our previous equation as:

$$V_\pi^{\text{cyc}} = \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{T_{(0)}} \Gamma_t r^\tau_{\text{CyclER}}[t]\right] + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(0)}+1}^{T_{(-1)}} \Gamma_t r^\tau_{\text{CyclER}}[t]\right] + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(-1)}+1}^{\infty} \Gamma_t r^\tau_{\text{CyclER}}[t]\right].$$

When a path τ is accepting, by definition, $\mathbb{E}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{T_{(0)}} \Gamma_t r^\tau_{\text{CyclER}}[t] \middle| \tau \models \varphi\right] = 1$ because every accepting initial path will achieve a reward of 1.

By considering accepting trajectories of LTL formula φ, then $T_{(-1)} = \infty$:

$$V_\pi^{\text{cyc}} \geq \left(1 + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(0)}+1}^{\infty} \Gamma_t r^\tau_{\text{CyclER}}[t] \middle| \tau \models \varphi\right]\right) \mathbb{P}[\pi \models \varphi] = \frac{p}{1-\gamma}$$

where the inequality follows from having dropped any value from non-satisfying trajectories. On the other hand, by the law of total expectation, for a lower bound we have:

$$\begin{aligned} V_\pi^{\text{cyc}} &= p \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{\infty} \Gamma_t r^\tau_{\text{CyclER}}[t] \middle| \tau \models \varphi\right] + (1-p) \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{\infty} \Gamma_t r^\tau_{\text{CyclER}}[t] \middle| \tau \not\models \varphi\right] \\ &\leq p\frac{1}{1-\gamma} + (1-p)\frac{1-\gamma^{M+1}}{1-\gamma} \end{aligned}$$

where the first term comes from the upper bound on $V_\pi^{\text{cyc}} \leq \frac{1}{1-\gamma}$ and the second term comes from bounding $T_{(0)}$ with a uniform upper bound M by Assumption 3.3.2

Combining the upper and lower bound together and subtracting off p from both sides, we have

$$0 \leq (1-\gamma)V_\pi^{\text{cyc}} - p \leq 1 - \gamma^{M+1}$$

Select $\gamma \geq (1-\epsilon)^{\frac{1}{M+1}}$ which implies that

$$|(1-\gamma)V_\pi^{\text{cyc}} - p| \leq \epsilon$$

□

Lemma 3.3.4. *Let $p^* = \max_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi]$. Under Assumption 3.3.2, then any policy π optimizing V_π^{cyc} (ie. achieving $V_{\max}^{cyc}\}$ maintains $|V_\pi^{cyc} - p^*| \leq \epsilon$.*

Proof. This follows by an identical argument as in Theorem 4.2 in (Voloshin et al. 2023), by using Lemma 3.3.3: V^{cyc}-optimizing policy π^*_{cyc} must satisfy $|(1-\gamma)V^{\text{cyc}}_{\max} - p^*| \leq \epsilon$ when γ is selected as in Lemma 3.3.3.

□

CyclER with Quantitative Semantics

Researchers have defined *Quantitative Semantics* (QS) for a number of temporal logics in order to quantify the measure of satisfaction for a given specification. These semantics, originally developed to monitor how close hybrid control systems are to violating properties reliant on continuous-value sensor data (Maler et al. 2004), have since been used in reinforcement learning to learn policies that satisfy formulae in a variety of specification languages, including Signal Temporal Logic (STL) (Li et al. 2017; Balakrishnan et al. 2019). However, existing methods of QS for reward shaping fail to effectively extend to indefinite-horizon LTL tasks. In what follows, we will introduce QS for LTL, discuss why naïvely using QS as reward is ill-fitted to deep RL for LTL, and introduce our own approach, extending the CyclER reward shaping method to effectively incorporate QS.

To use QS, we associate each atomic predicate $x \in \text{AP}$ with a *robustness measure* $f_x : \mathcal{S} \to \mathbb{R}$ that quantifies how close x is to being satisfied at state s. x evaluates to true at a given state iff $f_x(s) \geq c_x$, where c_x is a constant threshold. We can compose variables in AP with the logical and temporal operators of LTL by introducing QS for each operator, which follows the standard semantics defined for languages like TLTL (Li et al. 2017) and STL (Maler et al. 2004; Fainekos et al. 2009) and is provided in figure 3.5. An LTL formula φ is true if the quantitative evaluation of $\rho_\varphi > 0$ and false otherwise. We also define a maximum and minimum achievable value for ρ in a given MDP as $\rho_{\max}$ and $\rho_{\min}$, respectively.

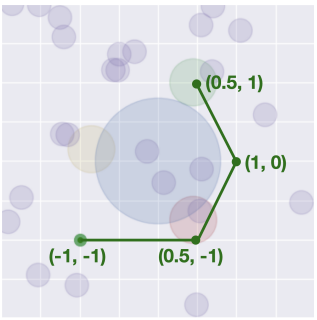


Figure 3.4: A toy trajectory in the Flatworld MDP.

To better understand quantitative evaluation for a given LTL formula, consider a trajectory (which we will denote as ξ) from the Flatworld MDP, shown in figure 3.4 and the formula $\varphi_{\text{toy}} = F(r)\&G(\neg b)$. We can define robustness measures for our atomic propositional variables as $f_x(s) = \text{distance}(s, x_{\text{center}}) < x_{\text{radius}}$, requiring that the agent be within a region for its variable to evaluate to true.

Let's quantitatively evaluate φ_{toy} on the trajectory ξ. For the expression $F(r)$, we find the maximum quantitative evaluation for the variable r in our trajectory. Since our trajectory visits the red region at point (0.5, -1), the evaluation for r at that point is some positive value $c_r > 0$, so the expression $F(r)$ will evaluate to true. For the expression $G(\neg b)$, we negate the quantitative evaluation and find the minimum value for the negated evaluation of b. At the point (1, 0), the agent is closest to the blue region, but since it does not enter it, the minimum value is some positive value $c_b > 0$. The quantitative evaluation for φ_{toy} on ξ is therefore a positive value $\min(c_r, c_b)$, so φ_{toy} evaluates to true for ξ.

Suppose we were to naively use the quantitative evaluation of a given LTL specification "off-the-shelf" as a reward function for an RL agent. Since the quantitative evaluation of a trajectory at any given point in time requires evaluating the future states of the trajectory, we can only assign a reward for entire trajectories. For the toy specification considered in our example, the reward would be the smaller of (1) the maximum distance the agent

$$
\begin{aligned}
\rho(s_{t:t+k}, \top) &= \rho_{\max}, \\
\rho(s_{t:t+k}, f_x(s_t) < c_x) &= c_x - f_x(s_t), \\
\rho(s_{t:t+k}, \neg\varphi) &= -\rho(s_{t:t+k}, \varphi) \\
\rho(s_{t:t+k}, \varphi \implies \psi) &= \max(-\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \varphi \& \psi) &= \min(\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \varphi \| \psi) &= \max(-\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, G(\varphi)) &= \min_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \varphi)) \\
\rho(s_{t:t+k}, F(\varphi)) &= \max_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \varphi)) \\
\rho(s_{t:t+k}, X(\varphi)) &= \rho(s_{t+1:t+k}, \varphi)(k > 0) \\
\rho(s_{t:t+k}, (\varphi U \psi)) &= \max_{t' \in [t, t+k)} (\min(\rho(s_{t':t+k}, \psi), \min_{t'' \in [t, t')} (\rho(s_{t'':t'}, \varphi))))
\end{aligned}
$$

Figure 3.5: Quantitative Semantics for Temporal Logics, used in Metric and Signal Temporal Logics.

reaches from the blue region and (2) the minimum distance the agent reaches from the red region during the trajectory. Although this seems reasonable as a reward for our example, the quantitative evaluation of an LTL formula becomes increasingly more obscure as the specification increases in complexity. For example, for the specification defined in Figure 3.1, which instructs the agent to indefinitely oscillate amongst the red, green and yellow regions while avoiding blue, ξ would have a lower quantitative evaluation than a trajectory that does not visit any of the three regions but just barely enters the blue region, even though the latter violates the specification without making any qualitative progress. Moreover, there would be no meaningful difference in quantitative evaluation between ξ and a trajectory that visits red, green and yellow while avoiding blue, or between a trajectory that completes the task twice, or thrice, and so on.

The quantitative evaluation of a trajectory as a reward signal for LTL fails because there are no well-defined terminal conditions for evaluating a finite trace under an infinite-horizon specification (e.g., φ from Figure 3.1), which means that value produced is often useless when used as a reward signal. This is made worse due to the fact that quantitative evaluation of an LTL formula assigns a single value for an entire trajectory, making credit assignment difficult. We propose an alternative reward that avoids these issues, extending the CyclER approach to handle quantitative semantics by rewarding "progress" made by traversing individual transitions in $\mathbb{B}$.

The high-level intuition for our method is as follows: recall that for a given trajectory,

CyclER computes hypothetical rewards for each cycle in $\mathbb{B}$, where reward is given if the next transition is taken within that cycle during the trajectory. Our key insight is that we can straightforwardly extend this paradigm to use QS by instead rewarding quantitative progress made towards taking the next transition within an individual cycle. Each transition in $\mathbb{B}$ corresponds to a non-temporal predicate of atomic propositions. Crucially, we can quantitatively evaluate these predicates on individual states, rather than trajectories. When an agent moves to a new state in $\mathcal{M}$, we can quantify how close the agent is to satisfying the transition predicate (and therefore taking the transition in the cycle), and compare it to how close the agent was to satisfying the transition predicate in the previous state. If there is a positive difference between these two values, we reward that progress made towards satisfying the transition predicate.

We present our reward function in function 3.7. In the context of Alg. 3, the reward function defined in 3.7 will replace $r_{\mathcal{C}}$. Intuitively, we can interpret the reward function defined in 3.7 as identical to $r_{\mathcal{C}}$, but rewarding quantitative progress towards taking a transition in a cycle rather than only offering reward once that transition is taken. We normalize progress using $\rho_{\max}$ and $\rho_{\min}$ to ensure that the scale of rewards from function 3.7 remain consistent.

Let us once again return to the example trajectory in Figure 3.4, with the LTL specification and LDBA from Figure 3.1 as our objective, and consider the cycle $\{1, 2, 3, 0\}$. We begin in state 1 of $\mathbb{B}$, where the transition we aim to take has the corresponding predicate r. When we transition from (-1, -1) to (0.5, -1), we satisfy r, and receive positive reward from CyclER because we are closer to r than we were in the previous state. We are now in state 2 of $\mathbb{B}$ and seek to take the transition with predicate g. For the transition from (0.5, -1) to (1, 0), we receive positive reward for getting closer to g, and for the transition from (1, 0) to (0.5, 1) we receive positive reward for successfully moving closer to g. Note that each transition of our trajectory ξ qualitatively makes progress towards visiting the accepting state of $\mathbb{B}$, and this is reflected in the positive rewards assigned by r_{qs}.

In environments where the individual variables in AP are difficult to satisfy, the usage of QS can offer a more dense reward that still leverages the full expressivity of LTL. In our experiments, we show that using the QS version of CyclER strongly outperforms existing approaches of using QS in learning to satisfy indefinite-horizon LTL specifications (Li et al. 2017; Balakrishnan et al. 2019) and enables the learning of LTL-compliant policies in complex environments.

$$r_{\text{qs}}(s, b, s', b', e, c) = \begin{cases} \frac{\rho_{c[b]}(s') - \rho_{c[b]}(s)}{(\rho_{\max} - \rho_{\min}) * |c|} & \text{if } (b, L^{\mathcal{M}}(s'), b') \in c \text{ and } e[b, L^{\mathcal{M}}(s'), b'] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

We use $\rho_{\max}$ and $\rho_{\min}$ to normalize the quantitative progress made towards taking a transition[2]. Reward function 3.7 acts as a direct substitute to function 3.6 in Alg. 1 to compute r_{CyclER}. Unlike function 3.6, function 3.7 allows for nonzero rewards to be given

[2]We note that the reward in fn. 3.7 is most well-shaped when the robustness measures for all $x \in \text{AP}$ are of similar scale, but we make no such assumptions for the sake of generality.

more than once per transition in an LDBA, allowing for dense reward even in LDBAs with few transitions.

In our experiments (section 3.4), we show that incorporating quantitative semantics into CyclER for shaping r_{LTL} leads to improvement in empirical performance when compared to existing methods of using QS.

3.4 Experimental Results

We demonstrate experimental results in several domains with continuous state and action spaces on LTL tasks of varying complexity. We seek to answer the following questions: **(1)** Does CyclER learn satisfactory policies and avoid ignoring r_{LTL} in favor of r_{MDP}? **(2)** Does a policy that optimizes the dual reward formulation in r_{DUAL} gain higher r_{MDP} than a policy that only seeks to satisfy the LTL constraint? **(3)** How does the value of λ in r_{DUAL} affect the performance of the learned policy?

Experimental Domains and Tasks

In our experiments, we evaluate the efficacy of CyclER on indefinite-horizon (ω-regular) tasks expressible by LTL. We use environments where r_{MDP} does not explicitly correlate with r_{LTL} in order to effectively distinguish between policies that learn to only optimize r_{MDP} and policies that learn to satisfy the LTL specification.

FlatWorld The FlatWorld domain (3.1) is a two dimensional world with continuous state and action spaces. The agent (denoted by a green dot) starts at (-1, -1). The agent's state, denoted by x, is updated by an action a via $x' = x + a/10$ where $x \in \mathbb{R}^2$ and $a \in [0, 1]^2$. There exists a set of randomly generated purple 'bonus regions', which offer a small reward when visited. We use the specification from Figure 3.1 as our LTL task.

ZonesEnv We use the Zones environment from the MuJoCo-based Safety-Gymnasium suite of environments (Ji et al. 2023). In this domain, a robot must navigate the environment, which includes four differently colored goal regions and 'hazard' areas that offer a small negative reward. The robot receives an observation of lidar data that detects the presence of nearby objects at each timestep. The LTL task description instructs the agent to oscillate amongst visiting the four colored regions.

ButtonsEnv We use the Buttons environment, also from Safety-Gymnasium. This domain is a more challenging version of the Zones environment, where an agent must press a number of small buttons in a larger space while avoiding cube-shaped 'gremlins' that move in a fixed circular path. The LTL task description instructs the agent to press two specific buttons infinitely often, while avoiding making contact with gremlins. Unlike the ZonesEnv, 'bonus' regions are scattered around the environment, offering a small reward if visited.

For each random seed of training, the locations of the following objects were randomized and fixed: in FlatWorld, the location of the bonus areas, in ZonesEnv, the locations of all colored regions and hazard regions, and in ButtonsEnv, the locations of the buttons, bonus

areas, and gremlins. In ZonesEnv and ButtonsEnv, we use the Point robot from (Ji et al. 2023) as our agent, which has a 2-dimensional action space $a \in [-1, 1]^2$.

The observation space and environment used in our ZonesEnv are the the default spaces provided the Zones Level 1 environment in (Ji et al. 2023), with the following changes: there are additional lidar observations for each of the four colored zones, and we place four static collidable walls as boundaries to enclose the agent's environments at the border of where objects can be randomly placed. The observation space and environment used in our ButtonsEnv experiments are the default spaces provided the Button Level 1 environment in (Ji et al. 2023), with two gremlins and eight bonus regions.

In ZonesEnv and ButtonsEnv, we define a simple robustness measure for each atomic propositional variable in the environments (the red, yellow, green, and purple regions in ZonesEnv, and buttons 1 through 4 and gremlin for ButtonsEnv). The robustness measure for a general variable x at a given state s is defined as follows:

$$f_x(s) = \text{distance}(s, x) \leq 0$$

For example, if an agent was a distance of 2 units away from the red region in ZonesEnv, the robustness measure of the variable "red" at that state would evaluate to -2. For ButtonsEnv, where the gremlin variable refers to multiple moving objects, the robustness measure corresponds to the minimum distance from the agent to any gremlin. We set $\rho_{\max} = 0$ in our environments and $\rho_{\min}$ to be the negative largest distance achievable in each environment.

Our LTL task specifications are defined in Table 3.1. We use the Spot tool Alexandre Duret-Lutz et al. 2022 to convert our specifications into corresponding LDBAs. We report the number of states, transitions, and *MAC*s in each LDBA used in our experiments in Table 3.3.

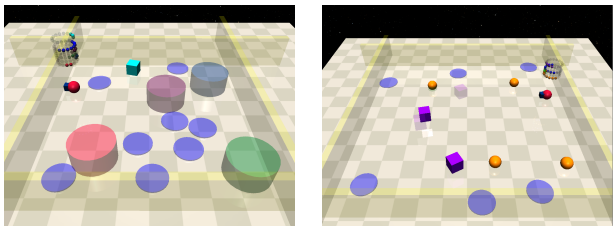


Figure 3.6: Example visualizations of the ZonesEnv (left) and ButtonsEnv (right) environments.

Implementation Details and Baselines

We use entropy-regularized PPO (Schulman et al. 2017) with a Gaussian policy over the action space as our policy class.

Although we are not aware of an existing approach that considers reward optimization under general LTL constraints for deep RL, we compare against a number of existing reward methods for temporal logic-guided RL as r_{LTL} in our r_{DUAL} formulation. We use a baseline policy trained using the LCER method (Voloshin et al. 2023), a state-of-the-art approach to RL for general LTL that uses an unshaped reward with counterfactual experience replay to improve the sample efficiency of learning. Additionally, we compare against the LCER baseline, but trained *only* on the

Environment	LTL φ
FlatWorld	$\mathbf{G}(\mathbf{F}(\text{red})\&\mathbf{X}(\mathbf{F}(\text{green})\&\mathbf{X}(\mathbf{F}(\text{yellow}))))\&\mathbf{G}(\neg\text{blue})$
ZonesEnv	$\mathbf{G}(\mathbf{F}(\text{blue})\&\mathbf{F}(\text{purple})\&\mathbf{F}(\text{red})\&\mathbf{F}(\text{green}))$
ButtonsEnv	$\mathbf{G}(\mathbf{F}(\text{button1})\&\mathbf{F}(\text{button2}))\&\mathbf{G}(\neg\text{gremlin})$

Table 3.1: Specification for each domain.

Environment	CyclER	LCER	TLTL	BHNR
FlatWorld	400	1000	-	-
ZonesEnv	200	1000	10	1
ButtonsEnv	100	1000	10	1

Table 3.2: Values for λ used by baselines for each domain.

unshaped LTL reward function r_{LTL}, in order to observe the performance of a policy that does not get 'distracted' during training by r_{MDP}.

In the ZonesEnv and ButtonsEnv domains, where the dynamics are more complex, we define simple robustness measures for each atomic proposition and **use the QS version of CyclER defined in section 3.3**. In these environments, we compare against two additional baselines that also use QS for reward shaping and are computable for infinite-horizon LTL tasks: a TLTL-based reward (Li et al. 2017) and BHNR (Balakrishnan et al. 2019).

For each baseline, λ was chosen to be that which led to best performance (on unshaped r_{LTL}, using r_{MDP} as a tie-breaker) from a hyperparameter sweep.

Training details. For all experiments, results are averaged over five random seeds. We provide hyperparameter choices for PPO for each experiment in Table 3.4 and choices for λ in Table 3.2. In Table 3.4, batch size refers to the number of trajectories. In our PPO implementation, we use a 3-layer, 64-hidden unit network as the actor using ReLU activations, and a 3-layer, 64-hidden unit network architecture with tanh activations in between layers and no final activation function for the critic. The actor outputs the mean of a Gaussian, the variance for which is learned by a 3-layer, 64-hidden unit network that shares the first 2 layers with the actor policy itself. All experiments were done on an Intel Core i9 processor with 10 cores equipped with an NVIDIA RTX A4500 GPU. We use the Adam optimizer in all experiments.

In Figure 3.7, reward was computed by evaluating the policy every ten trajectories in the case of FlatWorld, and every 25 trajectories for ZonesEnv and ButtonsEnv. The r_{LTL} and r_{MDP} values shown are from averaging performance over ten rollouts for each data point with a smoothing window of size 5.

For the BHNR baseline, we use a partial signal window size of 60 for FlatWorld, 700 for ZonesEnv, and 750 for ButtonsEnv, treating this value as a hyperparameter and performing

Environment	States	Edges	*MAC*s
FlatWorld	5	18	14
ZonesEnv	8	35	44
ButtonsEnv	3	9	4

Table 3.3: Details for the LDBAs used in each experimental domain.

Environment	Critic LR	Actor LR	α	Update freq.	γ	Batch size	$\lvert\tau\rvert$ (training)
FlatWorld	0.001	0.0003	-	1	0.98	128	120
ZonesEnv	0.0125	0.0025	0.3	3	0.99	128	700
ButtonsEnv	0.001	0.0003	0.2	3	0.99	128	750

Table 3.4: Hyperparameters used during training for each domain.

a sweep to select the window size. Our TLTL baseline is trained by assigning the TLTL value of a trajectory as the reward at the end of the trajectory, and using the discounted reward-to-go for each prior timestep as the reward signal. For our TLTL and BHNR baselines, we tried computing r_{LTL} in two ways: first, we tried using the original quantitative evaluation of the formula as the reward, where the resulting rewards were mostly negative due to how we defined our robustness measures for each variable. To evaluate if the sign and magnitude of the reward caused difficulty during learning, we normalized the quantitative evaluation done in TLTL and BHNR using $\rho_{\min}$ and $\rho_{\max}$, so that the reward value would be between 0 and 1. We found that this adjustment did not have a significant impact on either baseline's ability to learn r_{LTL}, but it did allow for easier optimization of r_{MDP}; therefore, we report the results from the normalized evaluation in our experiments.

Results

(1) Does CyclER learn satisfying policies and prevent ignoring r_{LTL}? Yes - our results demonstrate that CyclER achieves significant improvement in performance in satisfying the LTL task when compared to our baseline methods. In Figure 3.7, we plot the learning curves for both the *unshaped* r_{LTL} and r_{MDP}. We record the (stochastic) performance of the best policies found during training on an extended horizon to enable repeated visits to the accepting state, and present the results (averaged over 50 rollouts) in Table 3.5.

We find that unshaped rewards (LCER) quickly ignore r_{LTL} in most trials. From Table 3.5, we see that the LCER baseline even without r_{MDP} was not able to accomplish the LTL task as consistently as CyclER, even when successful in visiting an accepting state. This implies that reward shaping is critical for LTL-guided RL even in settings where no

	FlatWorld		**ZonesEnv**		**ButtonsEnv**	
	$\mathcal{B}^*$ visits	r_{MDP}	$\mathcal{B}^*$ visits	r_{MDP}	$\mathcal{B}^*$ visits	r_{MDP}
CyclER	$\mathbf{2.0 \pm 0.5}$	45.3 ± 8.5	$\mathbf{1.8 \pm 0.4}$	-27.8 ± 4.55	$\mathbf{2.6 \pm 0.3}$	30.4 ± 5.6
LCER	0.0 ± 0.0	103.4 ± 76.6	0.0 ± 0.0	-3.8 ± 1.9	0.0 ± 0.0	118.8 ± 143.2
LCER, no r_{MDP}	0.8 ± 0.4	30.8 ± 10.1	0.0 ± 0.0	-2.7 ± 0.9	0.6 ± 0.4	13.2 ± 8.53
TLTL	-	-	0.0 ± 0.0	-4.0 ± 2.2	0.0 ± 0.0	9.6 ± 6.7
BHNR	-	-	0.0 ± 0.0	-0.8 ± 0.6	0.0 ± 0.0	35.8 ± 7.9

Table 3.5: Reward average and standard deviation achieved on each domain with an extended horizon. $\mathcal{B}^*$ visits identifies the average number of visits to an accepting state in $\mathbb{B}$ achieved for a trajectory from π, and r_{MDP} refers to the average MDP reward collected during a trajectory.

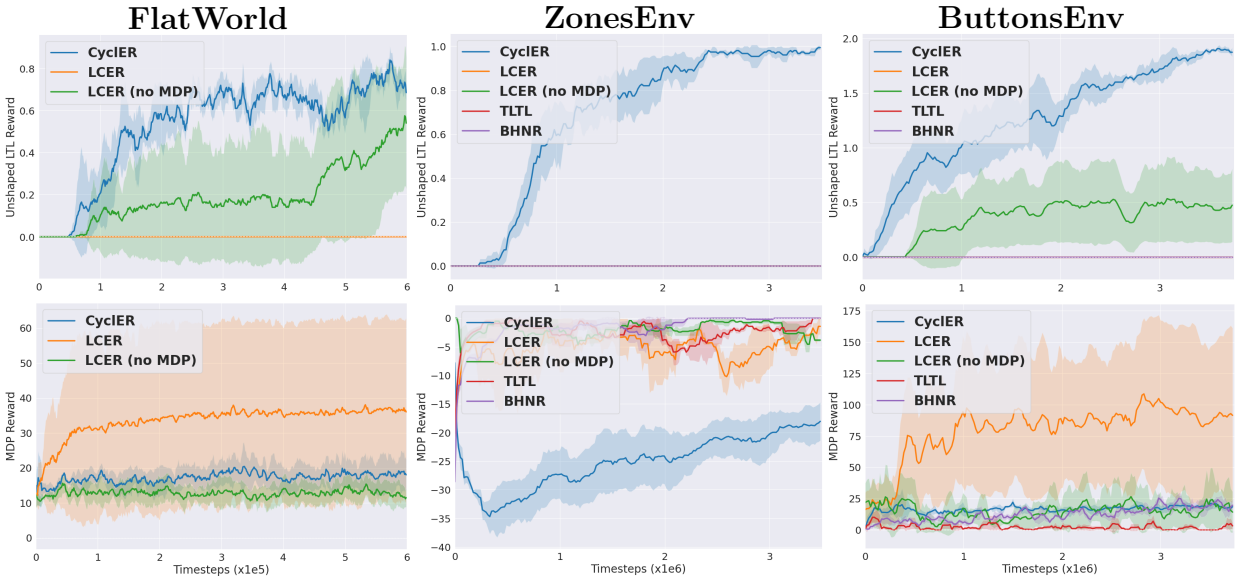


Figure 3.7: Training curves showing unshaped r_{LTL} (top) and r_{MDP} (bottom) performance averaged over 5 random seeds. Each point is the mean of 10 stochastic policy rollouts.

r_{MDP} is present. In ZonesEnv and ButtonsEnv, the TLTL and BHNR baselines, which are not suited to infinite-horizon tasks with multiple unordered subgoals, learned behavior that optimized their respective QS-shaped LTL rewards but did not correlate with task satisfaction (zero r_{LTL} achieved in Figure 3.7). CyclER with QS, on the other hand, quickly learned to achieve the tasks in these two domains. Most meaningfully, CyclER is able to repeatedly visit the accepting state in all domains (as evidenced by Table 3.5), demonstrating that our technique enables consistent-behaving policies that can indefinitely traverse accepting cycles. We additionally provide a qualitative analysis of learned behavior in the FlatWorld domain

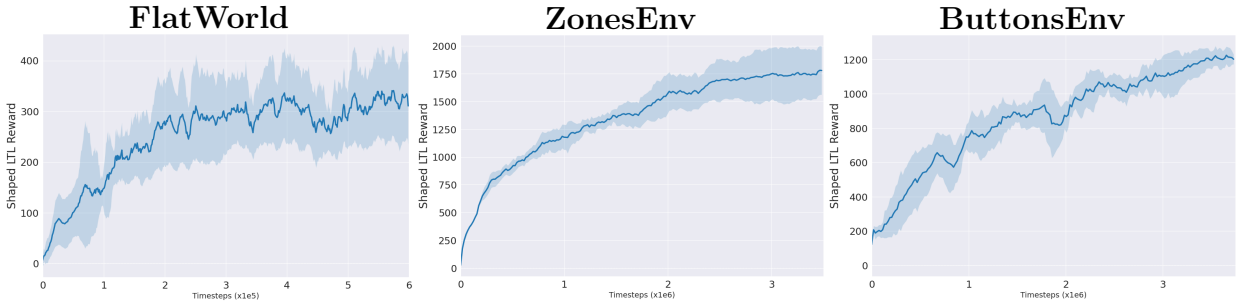


Figure 3.8: Training curves showing the *shaped* r_{LTL} achieved by CyclER-learned policies, averaged over 5 random seeds. Each point is the mean of 10 stochastic policy rollouts.

in Appendix 3.4.

(2) Does optimizing r_{DUAL} improve r_{MDP}? To evaluate this question, we conducted an ablation study where we trained a CyclER-based policy in the FlatWorld domain, making it completely unaware of r_{MDP}, and then evaluated its performance to observe if the r_{DUAL} formulation led to a nontrivial difference in behavior between policies. We found that the r_{DUAL}-optimizing CyclER policy visited $\mathcal{B}^*$ an average of 2.0 times (std. dev. 0.3) and achieved an MDP reward of 45.3 (std. dev. 8.5), and the r_{LTL}-only policy visited $\mathcal{B}^*$ an average of of 2.2 times (std. dev. 0.4) and achieved an MDP reward of 27.4 (std. dev. 0.7). Optimizing r_{DUAL} does lead to an improvement in r_{MDP}, albeit at the potential cost of LTL satisfaction. We provide a qualitative comparison of the two learned policies in Appendix 3.4.

(3) How does varying λ affect the resulting policy? In Table 3.6, we report results from a study where we vary the value of λ in r_{DUAL} for the FlatWorld domain experiment. We observe, as expected, a tradeoff in the performance of LTL satisfaction and r_{MDP} as λ increases. However, we notice that the tradeoff diminishes once a value for λ is reached that enables LTL-satisfying behavior. This supports our intuition that λ can effectively be used as a hyperparameter to trade off the empirical performance of LTL satisfaction and the MDP reward achieved by a policy.

	FlatWorld	
	$\mathcal{B}^*$ visits	r_{MDP}
$\lambda = 100$	0.0 ± 0.0	62.7 ± 4.36
$\lambda = 200$	0.0 ± 2.0	69.6 ± 4.1
$\lambda = 300$	2.0 ± 0.4	37.2 ± 9.0
$\lambda = 400$	2.1 ± 0.4	34.3 ± 7.3

Table 3.6: Performance results for CyclER with differing λ.

3.5 Conclusion and Takeaways

This chapter proposes a novel approach to finding policies that are both reward-maximal and probability-optimal with respect to an LTL constraint. Specifically, we introduce CyclER, an experience replay technique that automatically shapes the LTL proxy reward based on cycles within a Büchi automaton,

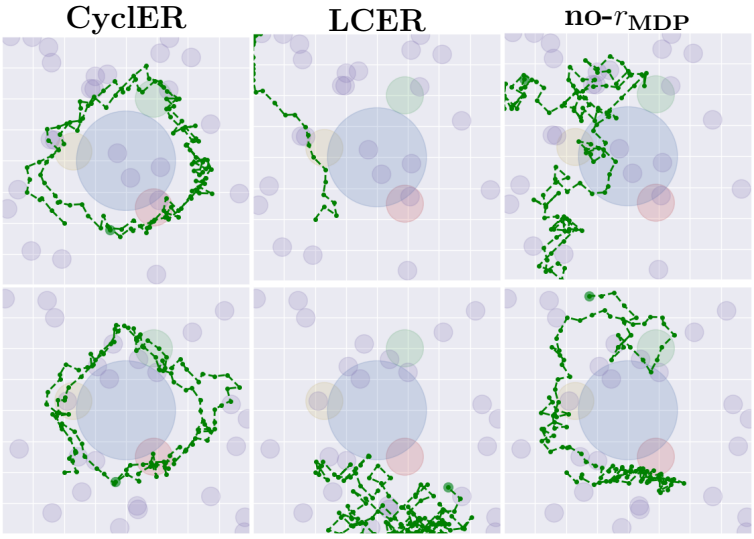


Figure 3.9: Sample trajectories from each baseline method in the FlatWorld domain (each row is a different seed). CyclER is able to consistently learn behavior that repeatedly visits and accepting state. The LCER baseline cannot achieve this long horizon task and instead optimizes for r_{MDP}. The LCER (no r_{MDP}) baseline, even when successful in visiting the accepting state (bottom right), does qualitatively learn satisfying behavior despite visiting the accepting state.

alleviating a sparsity issue that often plagues LTL-driven RL approaches. CyclER enables LTL-constrained policy optimization in continuous spaces using function approximators. We extend CyclER to effectively use quantitative semantics for full LTL and demonstrate its success empirically.

Shaped Reward Training Curves In Figure 3.8, we provide training curves for CyclER's performance on the *shaped* LTL proxy reward (i.e., CyclER-shaped reward) in each of our experimental domains. These curves allow us to compare the difference in performance between unshaped and shaped LTL reward and provide insight into a potential correlation between the two. We notice that in all baselines, CyclER-learned policies are able to achieve nonzero shaped reward early in training. Our learned policies achieve nonzero *unshaped reward* (recall the curves in Figure 3.7) slightly after significant returns in shaped reward are achieved, which supports our hypothesis that CyclER-shaped reward will successfully 'guide' a policy towards visiting the LDBA accepting state. We note that the scales of the CyclER-shaped LTL rewards visualized in Figure 3.8 vary significantly; the exact numerical values are dependent on specific environments and do not hold any exact meaning with respect to the unshaped LTL reward.

Qualitative Analysis To provide more insight into how CyclER learns to repeatedly visit the accepting state, we visualize sample trajectories from policies learned by each of our baselines in the FlatWorld domain, and present these samples in Figure 3.9. Recall that in

this domain, our LTL specification instructs an agent to traverse the red, yellow, and green regions indefinitely, while avoiding the blue region. It is obvious that the most efficient path to achieve this behavior is to navigate around the blue region and visit each colored region along a circular path.

On the left hand column of Figure 3.9, we see that policies learned using CyclER are able to perform this behavior, only slightly diverting from the circular path to visit purple regions and collect reward from r_{MDP}. In the middle column, we visualize trajectories from policies learned where LCER is used as r_{LTL}. Due to the sparsity of this reward function, the policy quickly finds areas in the MDP where purple regions are clustered together, and repeatedly visits those areas (in the top middle, it traverses to a corner where the entropy of the policy will affect its position the least.) On the right hand column, we show trajectories from LCER trained without r_{MDP} in its reward formulation. On the top right, we see that the baseline fails to achieve the task at all. More interestingly, however, on the bottom right, we find that the baseline does indeed achieve the task (i.e., visits the accepting state once), but does not exhibit behavior that qualitatively satisfies our task specification. After completing the task once, the agent turns back in the opposite direction and does not make obvious progress back towards either the red or yellow regions, suggesting that this baseline has not learned to repeatedly satisfy the task.

We also provide sample trajectories from the two policies learned in the experiment addressing question **2** in section 3.4. The trajectories, visualized in Figure 3.10, show the difference in behavior between the two successful policies when one is aware of r_{MDP} during training and when one is not. The MDP reward-aware policy acts notably different, altering its trajectory to reach purple bonus areas (even those that are somewhat far away from the colored regions of interest), while the MDP reward-unaware policy makes no such adjustments and optimizes for completing the LTL task as often as possible.

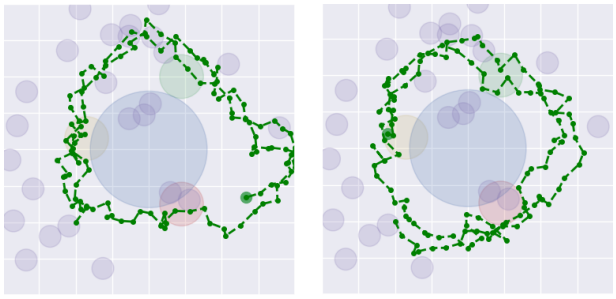


Figure 3.10: Trajectories from a CyclER-trained policy on r_{DUAL} (left) and a CyclER-trained policy on just the shaped r_{LTL} (right) in the FlatWorld domain.

Direct comparison in LTL-only setting To more directly evaluate CyclER's efficacy in alleviating the reward sparsity problem, we recreate the FlatWorld experiment from (Voloshin et al. 2023) with the same LTL specification. This domain does not have an additional MDP reward to be optimized. As such, the LCER baseline is able to consistently accomplish this objective. This allows us to observe whether CyclER-based reward shaping provides improvements in LTL-guided policy learning even in the absence of an MDP reward. We visualize the training curves in Figure 3.11. We can see that CyclER not only converges to a satisfying policy more quickly, but consistently achieves a higher reward when compared to the LCER baseline. This confirms

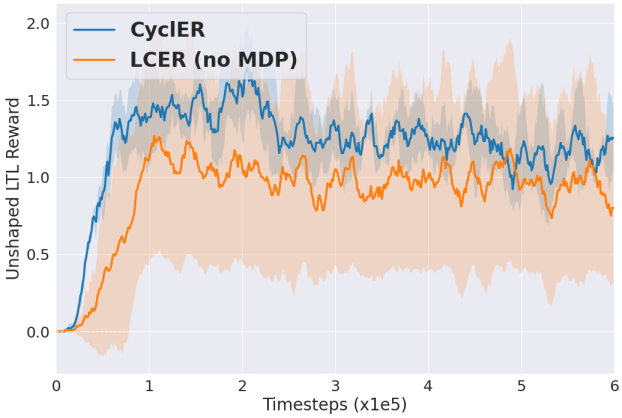


Figure 3.11: Training curve comparing unshaped LTL reward achieved by CyclER and LCER in the FlatWorld domain with $\varphi = G(F(r)\&F(y))\&G(\neg b)$. Performance is averaged over 5 random seeds. Each point is the mean of 10 stochastic policy rollouts.

our hypothesis that reward shaping is valuable even in settings where the LTL specification serves as the lone objective.

Limitations

The success of CyclER is ultimately limited to the quality and achievability of the atomic propositional variables in a given environment. If robustness measures are not available and a task specification has relatively few variables that are difficult to satisfy, CyclER will not provide significant improvement over existing unshaped LTL reward proxies. For example, CyclER will offer the same sparse reward for the specification $F(G(x))$ when no robustness measures are available, making it equivalent to existing reward structures. When robustness measures are available, it is important that measures for each variable in the set AP are of a similar scale, so that the QS-shaped rewards do not vary highly across different transitions in $\mathbb{B}$. The issue of needing similar scale for robustness measures is well-known in the temporal logic literature and is an open direction for future work (Balakrishnan et al. 2019; Li et al. 2017).

Although we show that the performance of policy learning is somewhat robust to λ in our formulation of r_{DUAL}, we do not have a systematic way of finding an appropriate value for λ beyond traditional hyperparameter search methods. We see an interesting opportunity for future work to intelligently search for λ based on the desired r_{LTL} and r_{MDP} of a user.

The CyclER approach incurs computational overhead by (1) computing all possible cycles prior to policy learning and (2) keeping track of all potential reward values for each cycle for each trajectory stored in an agent's replay buffer. We did not observe a significant slowdown or memory increase in our experiments as a result of this overhead. We acknowledge that in

complex specifications with a large number of cycles this overhead may become meaningful.

Future Work

There are numerous directions for future work. For example, the reward shaping idea behind CyclER can be extended to other classes of logical specifications, such as Reward Machines (Toro Icarte et al. 2022), that are particularly well-suited for reinforcement learning-based applications. We are also interested in applying CyclER to accelerate learning in multi-task LTL settings, such as (Vaezipoor et al. 2021; Qiu et al. 2023). Beyond standard reinforcement learning settings, the paradigm of CyclER holds promise in shaping rewards that are otherwise sparse and difficult to shape with standard reward shaping methods (Hu et al. 2020). Recently, RL has become an effective method for post-training foundation models. However, performing RL with 'verifiable' rewards (that is, reward functions that are based on programmatic correctness as opposed to human-based preferences) leads to a number of challenges, chief amongst them reward sparsity. Extending CyclER to these settings by shaping multi-turn, sparse reward agentic interactions offers the promise of greatly enhacing the reasoning capabilities of our current foundation models.

Chapter 4

Learning Symbolic Task Decompositions for Multi-Agent Teams

Until now, this thesis has only reasoned about policy learning settings involving a single agent on a single task. In reality, long-horizon tasks are often sufficiently complex that they require *teams* of agents to successfully coordinate and cooperate. In these settings, the challenge of reward sparsity we have discussed manifests in a different way: Using a single reward signal for a team of agents in multi-agent reinforcement learning (MARL) can make it challenging for agents to understand how their individual behavior impacts the overall reward. This challenge in MARL is known as the *credit assignment problem* (Agogino et al. 2004b) and can severely limit the effectiveness of naive reinforcement learning approaches in the multi-agent setting (Oroojlooy et al. 2023; Sunehag et al. 2017).

One method for addressing the credit assignment problem is to formulate the task as a *symbolic concept*, like those we have discussed earlier, that can be precisely decomposed into sub-tasks for assignment to individual agents (Neary et al. 2021; Smith et al. 2023). By using the reward signal from each sub-task, agents are credited for completing the specific sub-task they are assigned, even if the overall task is not achieved.

Previous literature has established sufficient conditions for "valid" decompositions of multi-agent *reward machines* (Toro Icarte et al. 2022), an automaton-based symbolic task structure, where satisfaction of the sub-tasks provably satisfies the overall task (Neary et al. 2021). Human-designed decompositions that satisfy these validity conditions help the agents learn to accomplish the task more quickly in tabular settings. Further work demonstrates that valid decompositions can be automatically generated based on human-designed heuristics (Smith et al. 2023).

These prior approaches to reward machine decomposition are limited when many possible decompositions exist. In these cases, the optimal decomposition of a task must be carefully designed and selected by a human. Selecting a meaningful task decomposition amongst many possible options often requires prior knowledge of the environment that is not assumed

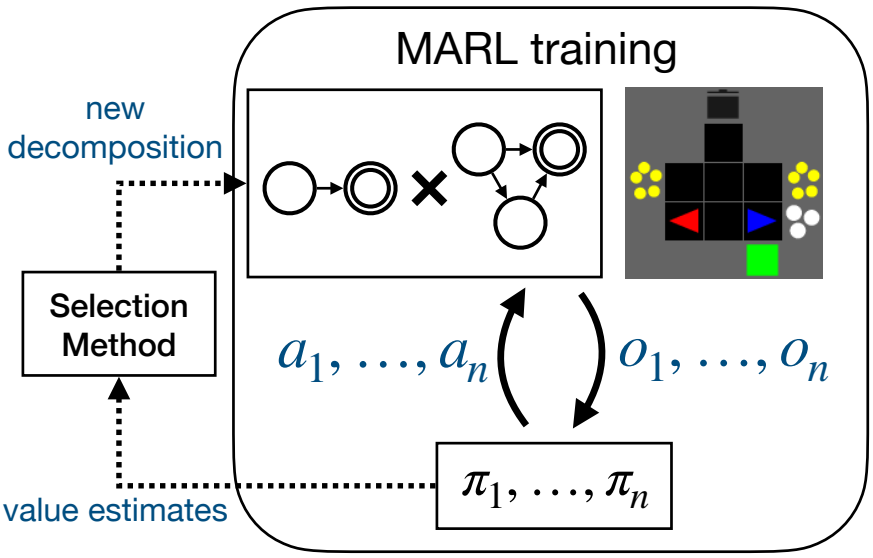


Figure 4.1: Visualization of our learning framework. At each new episode of training, a selection method chooses a possible symbolic decomposition of the task and assigns sub-tasks to each agent in the team. As each agent learns the viability of different sub-tasks, our selection method simultaneously finds the optimal task decomposition.

in standard RL, making the process tedious at best and impossible at worst. If a task decomposition is selected arbitrarily, the decomposition may not be compatible with the specifics of the environment to effectively break down the task.

Motivating example. Consider the "Repairs" environment in Figure 4.2, where a team of three robot agents must visit a number of communication stations in their environment to make repairs. First, any two of the agents must meet at headquarters (HQ, denoted by the control tower symbol), at which point a ready signal is sent out to both stations (red and yellow) to inform the stations that agents will be visiting each station to perform repairs. The agents are then tasked to visit these stations in any order. The agents can independently move around in grid in any of the four cardinal directions, or remain still. We can formulate this task in the form of a *Reward Machine* (RM), an automaton that encodes the objective over high level 'events' which occur in specific states of the environment (visualized in Figure 4.2). RMs offer a precise notion of task completion and can easily capture temporal dependencies required in objectives (i.e., visit the stations *after* HQ).

In the Repairs environment, there exists a hazardous region, encoded in orange, that prevents more than one agent from entering the region at a time. The location and existence of this hazardous region is *not* known to the agents a priori, which prevents this constraint from being encoded in the reward machine as part of the task description.

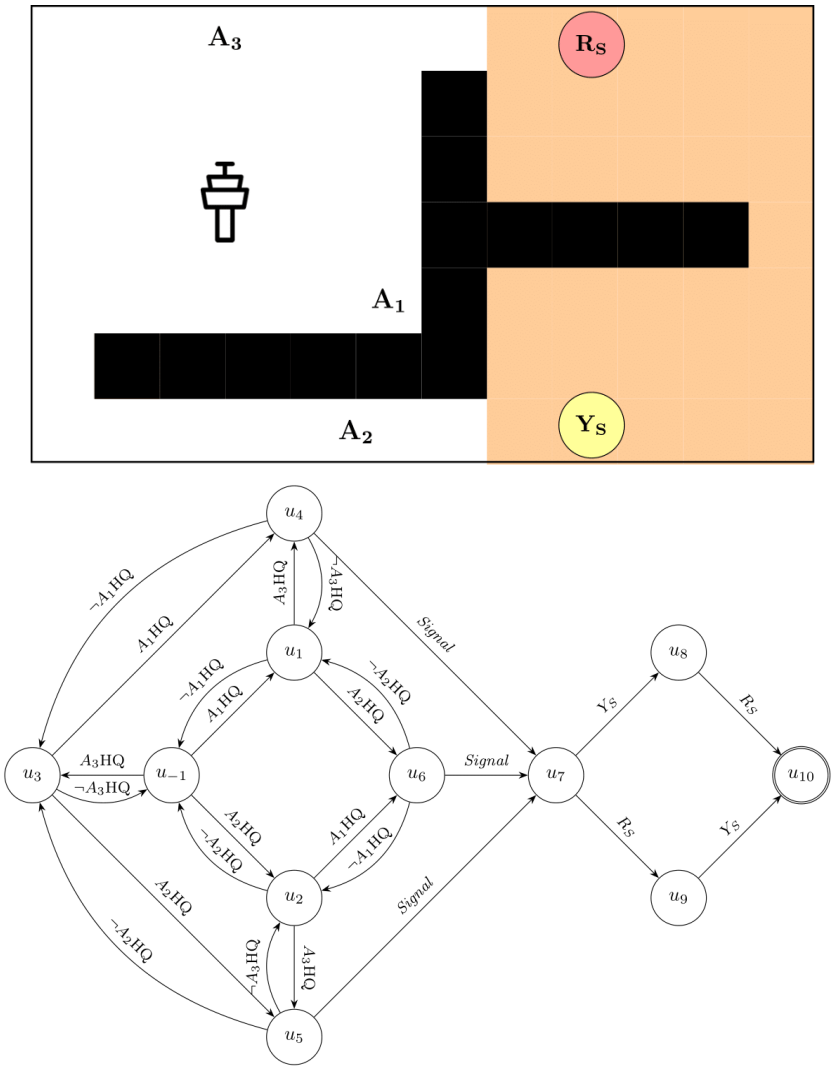


Figure 4.2: (Top) The "Repairs" MDP with a team of 3 agents. (Bottom) A task completion reward machine (RM) encoding the task: agents must navigate the environment to visit the HQ control tower, and then visit a set of communication stations. The goal state of the RM is denoted by concentric circles.

This task, even with no prior knowledge of the environment, can naturally be decomposed in a number of ways. For example, Agent 1 and Agent 2 can be tasked with meeting at HQ, and then Agent 2 can visit the yellow station and leave the hazardous region while Agent 3 visits the red station. Alternatively, Agents 2 and 3 can meet at HQ, and then Agent 1 can visit both the red and yellow stations. Although many plausible decompositions exist, knowing which decomposition of the overall task leads to the most efficient completion of

the task (i.e., is *optimal*) largely depends on the dynamics of the underlying environment. In our case, Agent 3 happens to be closest to the red station, which might mean that assigning that station to them would be optimal. However, without knowledge of the layout of the environment, as is standard in a model-free setting, this would be impossible to know ahead of time.

Our Contributions. In this chapter, we address the aforementioned limitations by introducing *an approach to automatically find optimal decompositions of an overall task into sub-tasks* for a team of agents. Our method is a lightweight extension of standard MARL and is applicable to model-free settings with no prior information about the environment or individual capabilities of the agents.

We summarize our approach briefly as follows. At the beginning of training, we generate possible *candidate* decompositions of our overall task as assignments of sub-tasks for our agents to accomplish. Then, during training, we explore selecting different decompositions for our team of agents and observe their performance as they attempt to learn a goal-conditioned policy that can achieve the variety of possible sub-tasks captured by our candidates. We record the value of agents' performances for different sub-tasks, and use this information to intelligently select subsequent decompositions for our agent team. This allows us to simultaneously learn (1) the optimal decomposition for our task and (2) policies for each agent that optimize said decomposition. We leverage selection strategies popularized in the multi-armed bandit algorithm literature (Katehakis et al. 1987; Auer et al. 2002a; Audibert et al. 2009) to balance exploring different candidate decompositions with exploiting the value our agents achieve as they learn sub-tasks during training. We visualize our approach in Figure 4.1.

In addition to our decomposition selection strategy, we introduce a novel training setup for RM–driven MARL that rectifies issues caused by dependent agent dynamics. Prior approaches (Neary et al. 2021; Smith et al. 2023) to RM-driven MARL make the assumption that the dynamics of each agent are independent of other agents, so that each agent can be trained individually in an environment absent from other agents. This assumption is often impractical: it is inefficient to train individual agents independently, and in many cases, multi-agent dynamics are codependent. For example, an agent that has completed their sub-task may obstruct another agent from the completion of their own sub-task. Our setup gives each agent in the team a *global* view of the overall task along with their sub-task, enabling agents to learn policies that accomplish their individual sub-tasks and help facilitate the completion of other agent's sub-tasks as well.

We summarize our contributions as follows: **(1)** We introduce a method for learning optimal decompositions from model-free interactions with the environment. **(2)** Within our method, we provide a novel training architecture that allows multiple agents to train simultaneously within an environment and avoids conflicts that arise due to dependent dynamics across agents. **(3)** We demonstrate our approach's improvement in learning multi-agent policies against baseline approaches in several cooperative environments. Our results show that

MARL benefits substantially from the improved credit assignment of task decompositions even when the optimal decomposition is not known *a priori*.

4.1 Preliminaries

MDPs and Labeled Markov Games

A Markov Decision Process (MDP) $\mathcal{M} = (S, A, T^{\mathcal{M}}, d_0, \gamma)$ is a tuple that consists of a state space S, an action space A, a transition function $T^{\mathcal{M}} : S \times A \to S$, an initial state distribution $d_0 \in \Delta(S)$, and a discount factor $0 < \gamma < 1$. A (stationary) policy $\pi : S \to \Delta(A)$ in $\mathcal{M}$ produces a distribution over actions given a state. Following standard RL notation, we define $\pi(a_t|s_t)$ as the probability of taking action a_t in state s_t at timestep t. A policy takes an action in a given state in $\mathcal{M}$, transitions to a new state via $T^{\mathcal{M}}$, and repeats, generating a **trajectory** of length τ as $(s_0, a_0, \dots s_\tau, a_\tau)$.

To generalize to the MARL setting, we extend $\mathcal{M}$ to a cooperative Markov game with homogeneous action spaces. We define a cooperative Markov game with n agents as $\mathcal{G} = (\mathcal{S}, \mathcal{A}, T^{\mathcal{G}}, d_{0^1} \dots d_{0^n}, \gamma, L^{\mathcal{G}})$, which corresponds to the joint set of states $\mathcal{S} = S_1 \times \cdots \times S_n$, the joint set of actions $\mathcal{A} = A_1 \times \cdots \times A_n$, a joint transition operator $T^{\mathcal{G}} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, a set of independent initial distributions for each agent $d_{0^1} \times \cdots \times d_{0^n}$, and a discount factor γ that remains unchanged. We consider the *centralized training, decentralized execution* (Bernstein et al. 2002) setting of MARL, where each agent receives independent observations and deploys their individual policies $\pi_i : S_i \to \Delta(A_i)$ at execution time, but are jointly trained. The joint policy $\boldsymbol{\pi} : \mathcal{S} \to \Delta(\mathcal{A})$ executes all individual policies simultaneously at each timestep, and successive states are dictated by $T^{\mathcal{G}}$, generating joint trajectories $((s_{0^0}, a_{0^0}, \dots s_{0^n}, a_{0^n}), \dots (s_{\tau^0}, a_{\tau^0}, \dots s_{\tau^n}, a_{\tau^n}))$.

In addition to the standard components of our Markov Game, we assume access to a known *labeling function* $L^{\mathcal{G}} : \mathcal{S} \to 2^{\Sigma}$ that maps states in S to a set of *environment events*, denoted by Σ. Each environment event $e \in \Sigma$ is represented by a variable that takes on a Boolean truth value (True or False). For example, in Figure 4.2, if Agent 1 is in the yellow station, Agent 2 is in the red station, and Agent 3 has not moved from its initial position, the labeling function for this joint state would return $\{Y_S, R_S\}$.

Task completion Reward Machines

In this work, we consider task completion reward machines (RMs) (Xu et al. 2020b; Neary et al. 2021; Smith et al. 2023) as our team objective. A task completion RM is specified by a tuple $\mathcal{R} = (U, u_{-1}, \Sigma, \delta, U^*)$ consisting of a set of states U, an alphabet of events Σ that trigger transitions in $\mathcal{R}$ via the transition function $\delta : U \times \Sigma \to U$, an initial state u_{-1}, and a set of goal states U^*. There are no outgoing transitions from any goal state $u^* \in U^*$. Task completion RMs additionally define an output scoring function $\sigma : U \times U \to \mathbb{R}$, where $\sigma(u, u') = 1$ whenever $u \notin U^*$ and $u' \in U^*$, and 0 otherwise. Task completion RMs are

Mealy machines (Mealy 1955) where reaching the goal state represents a valid completion of the task. We remark that δ and σ are partial functions defined on subsets of $U \times \Sigma$ and $U \times U$. The transition operator for a task completion RM takes in single events e in as input; when multiple events occur simultaneously, the events are passed in sequence to the RM in arbitrary order.

A task completion RM is connected to a Markov game by the labeling function $L^{\mathcal{G}}$. We can project a trajectory in $\mathcal{G}$ to a trajectory in $\mathcal{R}$ by applying $L^{\mathcal{G}}$ to each state, creating a sequence of event sets $(\{e_i \dots e_k\}_0, \dots \{e_i \dots e_k\}_\tau)$. This sequence of event sets transitions $\mathcal{R}$ to create a trajectory $(u_0, \dots u_\tau)$, where u_0 is the state resulting from $\delta(u_{-1}, \{e_i \dots e_k\}_0)$, and so forth. We say $\mathcal{R}$ *accepts* a trajectory if $u_\tau \in U^*$. By definition, the cumulative score of σ will be 1 for accepting trajectories, and 0 for all non-accepting trajectories.

Using task completion RMs in MARL

Recall our problem setting of *centralized training, decentralized execution*: each agent will receive independent observations during execution. In other words, each agent will view the cooperative Markov game as an MDP, where the presence and actions of other agents are captured by the dynamics of their respective environments. To train our agent team, we can use the acceptance condition of a task completion RM as a reward function for MARL. This objective can be naturally compiled down to a reward function expressed over a **product MDP** for individual agent policies $\pi_1 \dots \pi_n$. Concretely, a product MDP synchronizes $\mathcal{M}$ and $\mathcal{R}$ so that an agent may learn a policy over the joint space by coupling the reward machine state u_t during a trajectory with the MDP's state s_t, producing an action conditioned on both: $\pi(a_t|s_t, u_t)$. If the agents transition to a goal state in $\mathcal{R}$, they will receive a reward of one; all other transitions will receive a reward of zero. Existing deep RL algorithms, such as PPO (Schulman et al. 2017), have shown success in learning performant policies for RMs in a variety of single-agent settings by learning policies over the product MDP (Li et al. 2024; Voloshin et al. 2023).

However, in a MARL setting, providing the full task completion RM for each agent creates difficulties in learning. If all agents see the same states and transitions in $\mathcal{R}$, then all agents will receive credit when the task is completed, even if one or more agents did not contribute to completion of the task. Recall our running example in Figure 4.2. Suppose Agent 1 and Agent 2 visit HQ at the same time, and then Agent 2 visits the yellow and red stations, while Agent 3 remains stationary. Because the task is completed, all agents will receive equal reward for that episode. As a result, Agent 3 will think that its stationary behavior is desirable and be encouraged to act similarly in future episodes.

Existing work addresses this shortcoming by introducing the notion of *task decomposition* for a given task completion RM (Neary et al. 2021). A decomposition of a task completion RM creates sub-tasks in the form of n smaller RMs $\mathcal{R}_1 \dots \mathcal{R}_n$, derived from the original, that can then be assigned to each agent. Each agent is only concerned with accomplishing their own sub-task encoded by the RM assigned to them. In order to compute a decomposition, a practitioner provides *Local Event Sets* (LES) for each agent. An LES is a subset of events

$\Sigma_i \in \Sigma$ that is deemed relevant to agent i's individual sub-task and restricts agent i to only observing events in Σ^i. A task completion RM is then *projected* onto these local event sets to create an individual's sub-task reward machine $\mathcal{R}_i = (U_i, u_{-1^i}, \Sigma_i, \delta_i, U_i^*)$. The states, initial state, and goal states of $\mathcal{R}_i$ are sets of equivalence classes over states in $\mathcal{R}$ based on an equivalence relation that subsumes states whose transitions do not contain any event in Σ_i. The transition function is a projection of the original δ where a transition between two states u_j and u_k in $\mathcal{R}_i$ exists only if an event in Σ_i triggered a transition between two distinct states in $\mathcal{R}$ that were subsumed by u_j and u_k respectively. For the exact procedure of this projection, see (Neary et al. 2021).

For each agent's sub-task RM $\mathcal{R}_i$, we define an individual labeling function L_i that projects the set of environment events returned by L to the events belonging to an agent's local event set Σ_i. We say an event e is a *shared event* if it belongs to more than one local event set. For example, the event *"Signal"* in Figure 4.2 is a shared event. In the case of shared events, agents' sub-task RMs must *synchronize* on this event. This means that a synchronized event must trigger a transition for all agents' RMs that share the event (i.e. all sub-task RMs must be in the appropriate state for the synchronized event to cause a transition), or no transition is taken from encountering that event for any agent's RM.

The decomposition approach outlined in (Neary et al. 2021) relies on a practitioner manually designing the local event sets to assign to each agent. In order to help guide the search for an appropriate assignment of local event sets, (Neary et al. 2021) provides a notion of *validity* for a given decomposition: a decomposition is valid if and only if the parallel composition of all reward machines $\mathcal{R}_1 \dots \mathcal{R}_n$ is bisimilar to the original $\mathcal{R}$. If a decomposition is valid, then for any trajectory of events ξ, $\mathcal{R}$ accepts ξ if and only if all sub-task RMs $\mathcal{R}_1 \dots \mathcal{R}_n$ accept ξ. In other words, a trajectory of events that accomplishes all decomposed sub-tasks encoded by $\mathcal{R}_1, \dots, \mathcal{R}_n$ is guaranteed to solve the overall task.

4.2 Methodology

In practice, multiple valid decompositions often exist for a given RM. However, a valid decomposition may not be *feasible* under the dynamics of a given MDP; that is, the resulting learned policies may not be able to achieve the individual tasks prescribed by a decomposition. Moreover, when multiple feasible decompositions do exist, we do not know which decomposition most efficiently achieves the task or provides the best credit assignment for learning. Recall the Repairs environment and task in Figure 4.2. A feasible decomposition of the task would be for Agent 1 and Agent 2 to meet at HQ, then for Agent 2 to visit the yellow, then red stations in that order. However, this decomposition is less efficient than a decomposition where Agents 1 and 3 visit HQ, then Agent 2 visits the yellow station and leaves the hazardous region while Agent 3 visits the red station (assuming Agents complete their sub-tasks with optimal efficiency).

Recent work attempts to automate the search for RM decompositions by leveraging additional information provided by a practitioner (Smith et al. 2023). In this work, subsets

of events in Σ can be provided that either require or forbid an event to belong to a specific agent's local event set Σ_i, along with a utility function that quantifies how valuable an event would be to a specific agent. This information is then used to find a subset of events in Σ that still leads to a valid decomposition of $\mathcal{R}$, if one exists. However, the aforementioned approach does not leverage knowledge gained about the MDP during training and therefore cannot ensure that the decomposition generated from their method is feasible or optimally efficient. In what follows, we will introduce our approach, which aims to find the optimal decomposition by learning a policy on-the-fly for many possible sub-tasks during training.

Our approach can be broken down into three primary components: **(1)** automatically generating a set of possible (candidate) decompositions for our task, **(2)** using a task-conditioned policy to generalize learning across multiple decompositions, and **(3)** employing the Upper Confidence Bounds (UCB) strategy (Auer 2002) to balance exploration and exploitation of candidates throughout training.

Generating candidate decompositions

Approaches to procedurally generate decompositions of reward machines and similar automaton-based task specifications have been explored in the literature (Smith et al. 2023; Lauffer et al. 2022). Such methods can be used to generate a finite set of candidate decompositions $\mathcal{D} = \{(\mathcal{R}_1^1 \dots \mathcal{R}_n^1), \dots$
$(\mathcal{R}_1^{|\mathcal{D}|} \dots \mathcal{R}_n^{|\mathcal{D}|})\}$ from which the optimal decomposition can be found. We will denote an individual decomposition in our set as $d \in \mathcal{D}$. In our work, we will use the decomposition approach for task completion RMs introduced in (Smith et al. 2023). In this approach, all possible *valid* decompositions are generated given Σ, and each one is assigned a score based on three factors: (1) minimizing the average number of events assigned to each agent's local event set, (2) maximizing the similarity amongst the sizes of all local event sets, and (3) maximizing the average 'utility' of each agent's local event set, based on a practitioner-provided utility function that maps the assignment of an event to an agent to a scalar value. In our work, we assume that no utility function has been provided. We will therefore enumerate the 'top-k' valid candidate decompositions based on a weighted sum of scores pertaining to factors (1) and (2), where k is a hyperparameter that sets the number of decompositions we will consider, i.e. $k = |\mathcal{D}|$. Our objective is to find the decomposition $d^* \in \mathcal{D}$ that leads to a learned policy which will most efficiently achieve the original task $\mathcal{R}$.

Task-conditioned policies

We consider each agent policy in our team as sharing a *task-conditioned* policy. Instead of learning separate policies for individual sub-tasks, we learn a single policy that outputs actions conditioned on a specific sub-task RM belonging to a decomposition generated in section 4.2. We can then deploy separate copies of our policy on each agent during execution time.

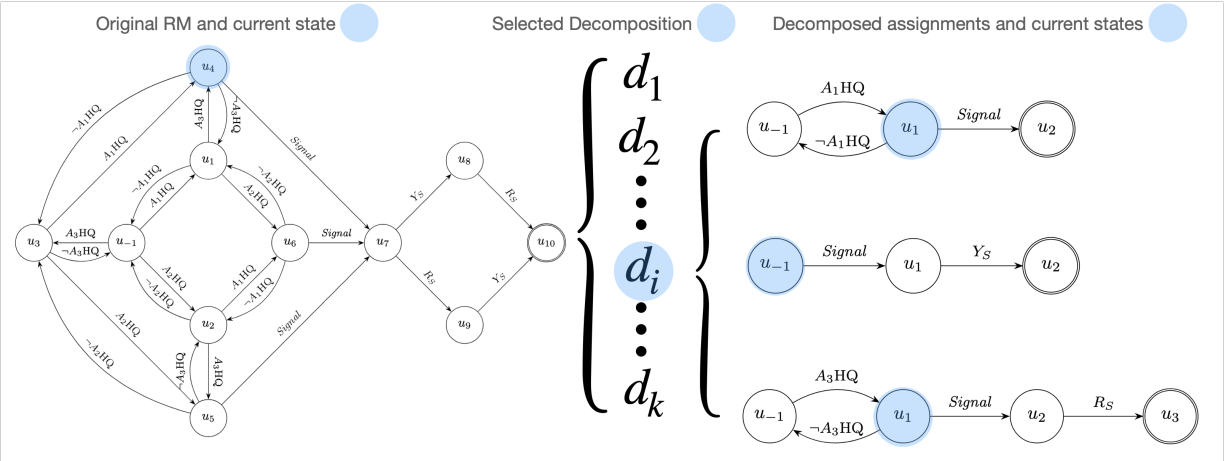


Figure 4.3: A visualization of the information each agent receives using the policy architecture described in section 4.2 for the Repairs task from Figure 4.2. In addition to the observation gathered from the MDP, each agent's policy is conditioned on (1) the current state of the original RM task, (2) which decomposition is currently selected, and (3) the current state of their assigned sub-task RM within the selected decomposition.

Policy Architecture Our policy is represented by a feedforward neural network that is shared and learned across all agents. The neural policy receives four inputs, depending on the individual agent:

- an observation from the underlying MDP (different for each agent);
- an encoding of the selected decomposition $d_j \in \mathcal{D}$ for the current execution episode (same for each agent);
- the current state in their sub-task $\mathcal{R}_i^j \in d_j$ (different for each agent);
- an encoding of the overall task and the current state in the overall task (same for each agent).

We visualize these inputs in Figure 4.3 for an example decomposition of our Repairs task. Learning a task-conditioned policy allows us to distinguish between different sub-tasks within different decompositions while exploiting the generalization capabilities of multi-task learning (Qiu et al. 2023; Vaezipoor et al. 2021) as well as the natural curriculum learning inherent in decomposition exploration. Let us return to the running example from Figure 4.2. Imagine that as part of a decomposition selected during training, agent A_1 is only assigned the task of meeting at HQ and nothing else. Even if this decomposition is not optimal, succeeding in this sub-task teaches A_1 the valuable skill of how to reach HQ. This

experience will be useful in the future when A_1 is tasked with more complicated sub-tasks, such as *"first go to HQ and then the red station"*.

Conditioning on the overall task allows us to relax the assumption of independent dynamics as we will describe in Section 4.2. In our experiments, we use one-hot encodings of the selected sub-task and the current position in the sub-task during the rollout. However, the embeddings could in principle be generated in any way, such as learned embeddings from a graph neural network that encode the structure of an RM (Yalcinkaya et al. n.d.).

Selecting decompositions during training

A naive approach to our objective of finding an optimal ordered decomposition would be to learn a task-conditioned policy for each $d \in \mathcal{D}$ independently, and then choose the decomposition that performs the best after a certain amount of training. As the number of candidate decomposition increases, this approach quickly becomes intractable. We seek to improve the efficiency of this process by simultaneously learning both the most efficient decomposition and the corresponding policies that achieve this decomposition.

A key insight of our work is that we can use rewards from previous executions of a sub-task $\mathcal{R}_i^j$ to estimate the satisfaction likelihood of that sub-task. These estimates, which we call *value estimates*, are used as a heuristic in decomposition selection to assess how well a policy is performing on a specific sub-task. Our approach computes value estimates as an *exponential weighted moving sum* of previous rewards, as more recent rewards are typically a more accurate reflection of the performance of the current policy.

We will denote the value estimates for a sub-task $\mathcal{R}_i^j$ as $V_{\mathcal{R}_i^j}$. On episode H of training on decomposition j, we can compute a sub-task's value estimate as $V_{\mathcal{R}_i^j} = \sum_{h=0}^{H} \alpha^{H-h} r_h$. Here, r_h represents the reward achieved by agent i under sub-task $\mathcal{R}_i$ from the h-th execution and α is a hyperparameter defining the decay rate.

With value estimates in hand for each agent policy in our team, on each sub-task, we can consider a number of heuristic approaches to utilize them. In this work, we use the Upper Confidence Bound (UCB) algorithm (Auer 2002), which balances exploring different decompositions with exploiting higher scoring decomposition via a hyperparameter β. The score assigned to each decomposition d_j is an average of the decomposition's current value estimates for each sub-task $\{V_{\mathcal{R}_1^j}, \ldots, V_{\mathcal{R}_n^j}\}$.

We note that the value estimates early in training may be arbitrarily inaccurate due to the lack of progress made in learning the policy for different sub-tasks by each agent. When inaccuracy is high early in training, exploration is critical so that agents can sufficiently optimize towards achieving each candidate sub-task before the value estimates are exploited to converge on the optimal decomposition. Once the optimal decomposition is converged upon, additional training will further optimize the policy conditioned on its corresponding sub-tasks.

Dealing with dependent dynamics

Prior work in RM-guided MARL (Neary et al. 2021; Smith et al. 2023) assumes that the underlying dynamics of the MDP are independent between agents. This prevents the use of an MDP with dynamics that model collisions or interactions between agents unless the interaction is explicitly modeled by the task's reward machine, which requires knowing about the interaction a priori.

Consider the motivating example visualized in Figure 4.2. The example includes dependent dynamics between the agents in the form of the hazardous region. Only one agent can occupy the region at a time, meaning that an agent's ability to enter the region is dependent on the other agents' positions and actions.

Prior works (Neary et al. 2021; Smith et al. 2023) require the assumption of independent dynamics so that the completion of an individual agent's reward machine is independent of the behavior of other agents. With dependent dynamics, this is rarely the case. For example, imagine that Agent 3 is assigned to reach the yellow target and Agent 2 is assigned to reach the red target. If Agent 3 enters the hazardous region and reaches the yellow target, it accomplishes its sub-task. Now, completion of the overall task only requires Agent 2 to reach the red target. Since the red target is located in the hazardous region, this requires Agent 3 to exit the hazardous region. However, Agent 3 has already accomplished its sub-task (and has no knowledge of Agent 2's task), so it has no incentive to leave the hazardous region.

We are able to relax the assumption of independent dynamics made in previous work (Neary et al. 2021; Smith et al. 2023) and handle environments such as the one in Figure 4.2 by allowing agents to condition not only on the status of their sub-task, but on the status of the overall task as well. In addition, agents are given a small reward for the completion of the overall task during training along with their primary reward for completing their assigned sub-task. This reward structure incentivizes agents to condition on the status of other agent's tasks (e.g., that Agent 2 needs to go to red) so that the overall task can be completed (e.g., Agent 3 is incentivized to leave the hazardous region).

4.3 Implementation and Experiments

We evaluate our proposed approach, which we refer to as **LOTaD** (**L**earning **O**ptimal **Ta**sk **D**ecompositions), in a variety of MARL settings with varying task complexity. Our code is available at https://github.com/thomasychen/LOTaD.

In our experiments, we seek to answer the following research questions: **(RQ1)** Does LOTaD outperform existing approaches to RM-guided MARL that are not learning-informed, including approaches that do not perform task decomposition? **(RQ2)** Can LOTaD learn to avoid pitfalls in learning that may occur due to dependent dynamics across agents? **(RQ3)** How does varying the number of candidate decompositions $k = |\mathcal{D}|$ affect the learning performance of LOTaD?

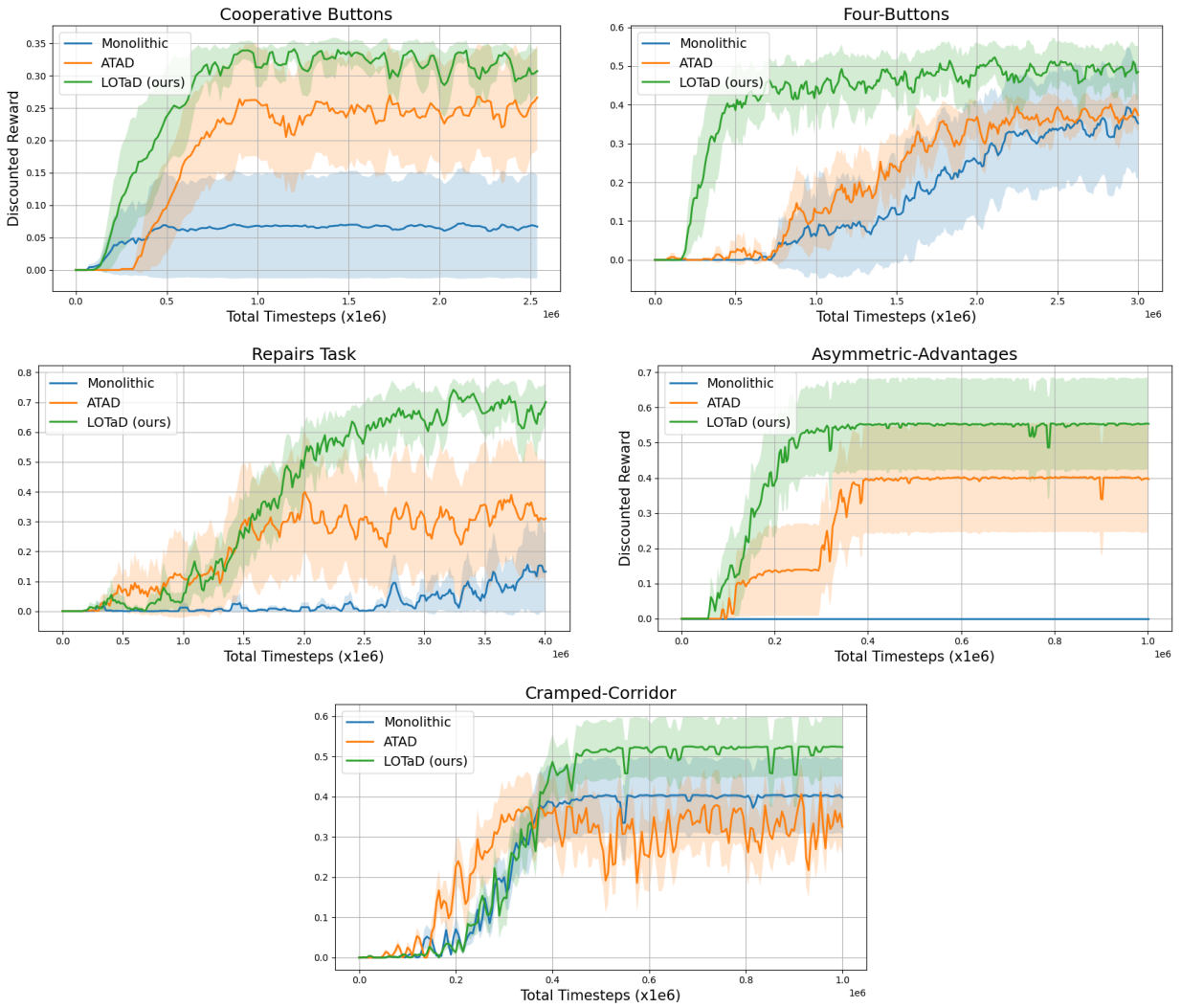


Figure 4.4: Training curves for LOTaD and baseline methods in our experimental domains. Results are averaged over 5 random seeds.

Environments and tasks

Our environments include the **Repairs** environment and RM task presented in Figure 4.2. The environment is instantiated as a 7x12 grid in which agents can move in any of the cardinal directions, or take a no-op action where no movement occurs. We make the Repairs environment stochastic by giving a small chance of an agent "slipping" and taking a random action rather than the action selected by their policy. Each agent observes only their position in the world at each timestep.

In addition to the Repairs environment, we also include two "Buttons" environments that

require a team of agents to press a series of buttons in a particular order. We instantiate two environment-task pairs: First, we use **Cooperative Buttons**, a task from (Neary et al. 2021) where any agent must reach a specified goal location, but in order to do so, must traverse regions that can only be crossed once a corresponding button has been pressed. Second, we use **Four-Buttons**, a task where two agents must press four buttons (yellow, green, blue, and red) in an environment, with an ordering constraint that the yellow button must be pressed before the red button. Both of these environments are represented as 10x10 grids with the same observation space, action space, and "slipping" dynamics as the Repair environment. Visualizations of these environments are presented in Figure 4.5.

Lastly, we evaluate LOTaD on two environments from the popular multi-agent benchmark *Overcooked* (Carroll et al. 2019). In both environments, we provide the same simple task of delivering a soup to the delivery station, which requires putting three onions in a pot, plating, and delivering the soup. Agents must coordinate depending on the dynamics of their environment to efficiently cook and deliver the soup. We use a **Cramped-Corridor** environment, where two agents must navigate around one another to reach the pot at the end of a small corridor in a cramped room, and an **Asymmetric-Advantages** environment, where two agents are in separate rooms, but have access to an asymmetric set of resources in their respective rooms. We visualize both Overcooked environments in Figure 4.5. The observation space and action space for these environments are the same as those provided in the Overcooked implementation from (Rutherford et al. 2023) with the adjustment that we do not expose the number of onions in the pot or whether the soup has finished cooking in order to ensure that no information provided by the labeling function is redundant in the agents' observations.

In all environments, we apply a discount factor $\gamma < 1.0$ to the reward offered by our RM to incentivize our agents to accomplish the task as efficiently as possible. We generate $k = 10$ decomposition candidates in all experiments using the generation method described in Section 4.2 for LOTaD to search amongst.

Baselines

To evaluate LOTaD, we compare against a baseline that selects a decomposition using the **ATAD** method (Smith et al. 2023). This approach selects a set decomposition prior to learning based on the scoring method described in Section 4.2. We break ties between top scoring decompositions arbitrarily. In addition to this baseline, we also compare against a baseline approach that assigns each agent the overall task. We call this baseline **Monolithic** and use it to evaluate how MARL would perform if no decomposition of the RM was used. We use PPO (Schulman et al. 2017) with a Gaussian policy over the action space for each agent as our RL algorithm in every environment.

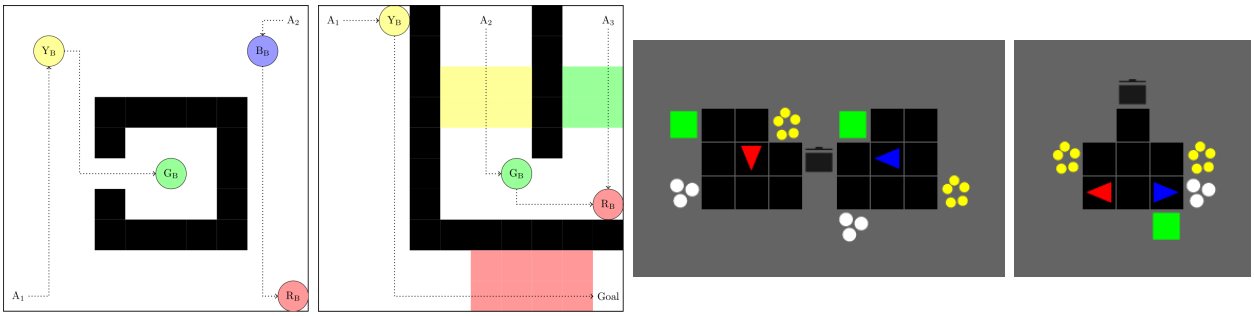


Figure 4.5: From left to right: The Four-Buttons environment, The Cooperative Buttons environment, the Asymmetric-Advantages Overcooked Environment, and the Cramped-Corridor Overcooked Environment.

Reward Machines

In Table 4.3, we provide the task completion RMs used in each experimental domain. For the Overcooked domains (Cramped-Corridor and Asymmetric-Advantages), the same RM task is used. We omit sink state transitions from the Four-Buttons RM for the sake of visual clarity. These transitions would be taken if the event R_B is triggered at states $\{u_{-1}, u_1, u_2, u_3, u_5, u_6, u_7, u_{10}\}$ and would lead to a sink (rejecting) state with no outgoing transitions.

As discussed in section 4.2, we use the ATAD method from (Smith et al. 2023) to generate candidate decompositions. The ATAD method uses weight hyperparameters to balance the importance of condition (1) (decomposition size) and (2) (decomposition balance); we use weights 2 and 0.5 respectively in all experiments for both LOTaD and the ATAD baseline. The sub-task RMs generated by this approach are called *accident avoidance* RMs, and transition sub-task RMs to sink states if any event is experienced that does not belong to any local event set. We use these accident avoidance RMs as our sub-tasks, with the modification that and transitions to sink states receive a reward of 0 rather than a reward of -1.

ATAD also allows for the provision of forbidden and required event sets, which specify certain environment events that either cannot or must appear in a specific agent's local event set, respectively. The forbidden event sets for the Repairs environment were $A_1 = \{A_2\text{HQ}, \neg A_2\text{HQ}, A_3\text{HQ}, \neg A_3\text{HQ}\}$, $A_2 = \{A_1\text{HQ}, \neg A_1\text{HQ}, A_3\text{HQ}, \neg A_3\text{HQ}\}$, and $A_3 = \{A_2\text{HQ}, \neg A_2\text{HQ}, A_1\text{HQ}, \neg A_1\text{HQ}\}$. These forbidden event sets prohibit events that are specific to other agents from appearing in an agent's local event set. The required event sets for the Repairs environment are {Signal} for all agents. For the Cooperative Buttons task, the forbidden event sets are $A_1 = \{A_2^{R_B}, A_2^{\neg R_B}, A_3^{R_B}, A_3^{\neg R_B}\}$, $A_2 = \{A_3^{R_B}, A_3^{\neg R_B}\}$, and $A_3 = \{A_2^{R_B}, A_2^{\neg R_B}\}$, following a similar pattern to the Repairs environment. For the Four-Buttons task and the Overcooked task, both the forbidden and required event sets for all agents are empty. All decompositions generated by ATAD are required to comply with these forbidden and required event sets. We note that the provided forbidden and required

event sets do not exploit any information about the environment and are independent of the MDP's dynamics.

Hyperparam	**4-B.**	**Coop. B.**	**Repairs**	**Cramped-Corr.**	**Asymm.-Adv.**
Learning rate	5e-4	5e-4	5e-4	2.5e-4	2.5e-4
Batch size	256	256	256	256	256
Number of epochs	5	5	5	5	5
Discount	0.97	0.95	0.95	0.99	0.99
Entropy Coefficient	0.1	0.1	0.1	0.01	0.01
Max. grad. norm.	0.5	0.5	10	0.5	0.5
Value loss coef.	0.5	0.5	10	0.5	0.5

Table 4.1: PPO training hyperparameters.

Hyperparam	**4-B.**	**Coop. B.**	**Repairs**	**Cramped-Corr.**	**Asymm.-Adv.**
Exponential Decay α	1	1	1	1	1
UCB Exploration β	0.5	0.5	0.5	0.5	0.5
Num. Candidates k	10	10	10	10	10
Maximum Episode Length	100	100	400	400	200

Table 4.2: Additional experiment hyperparameters.

Hyperparameters and Environment Details

Table 4.2 shows the PPO hyperparameters used for each training run in our experiments.

Cooperative Buttons In the Cooperative Buttons task from (Neary et al. 2021) used in our experiments, a goal region (denoted as 'Goal') must be reached by any of the three agents. There are additionally three colored regions (red, yellow, and green), and three buttons of the same color present. In order for any agent to enter a colored region, the corresponding button must be pressed by at least one agent, otherwise, the attempt to enter that region results in a no-op. In the case of the red region, two agents must press the red button in order to enable an agent to traverse the red region. Walls (in black) separate Agent 1 from Agents 2 and 3. To reach the goal location, Agent 1 must first press the yellow button, which allows Agent 2 to traverse the yellow region to press the green button. Then, Agent 3 can traverse the green region, and both Agents 2 and 3 can press the red button, allowing Agent 1 to traverse the red region and reach the goal.

Four Buttons In the Four-Buttons environment, a team of two agents must press the four buttons placed in the environment with an ordering constraint that the yellow button must be pressed prior to the red button.

Overcooked Environments In the two Overcooked environments used in our experiments, the teams share the same task: three onions must be placed in the pot in order for the soup to start cooking. After a small number of timesteps have passed, the soup is considered cooked, and an agent must then pick up a plate and bring it to the pot in order to plate it. An agent then must take the plated soup and bring it to a delivery station. In the Asymmetric-Advantages environment, agents are in separate rooms, and both agents have shared access to a pot. Each agent has a distinct advantage (for example, the red agent can more easily place onions in the pot, whereas the blue agent can more quickly plate and deliver a cooked soup). In the Cramped-Corridor environment, only one agent may interact with the pot at a time, and the pot is placed in a small corridor. This can cause issues due to dependent agent dynamics: for example, if one agent finishes putting all of the onions in the pot, it must get out of the corridor either so the other agent can plate and deliver the soup.

Results

We plot training curves for all experimental domains in Figure 4.4. In these curves, we report the current best discounted reward achieved by LOTaD amongst all decomposition candidates under consideration. We find that LOTaD outperforms both the monolithic and ATAD baselines across all environments, answering **(RQ1)** in the affirmative. In many of our environments, learning decompositions with LOTaD allows for agents to explicitly parallelize their contributions towards achieving the task. For example, in the Four-Buttons environment, a candidate decomposition allows for one agent to visit the yellow and green buttons while the other agent visits the blue button.

Interestingly, LOTaD-learned decompositions are useful even when explicit parallelization is not possible, such as the Overcooked RM task. Decompositions of this task can still facilitate multi-agent learning: for example, one agent may be tasked with putting all three onions in the pot, while the other agent must wait for the soup to be cooked before plating and delivering the soup. In this decomposition, the agent tasked with plating and delivering the soup may fetch a plate and stand near the pot so that the soup can be quickly delivered upon completion of cooking.

In comparison to LOTaD, the ATAD baseline is unable to consistently find performant task decompositions. Although ATAD selects a decomposition based on the same scoring method by which we select $\mathcal{D}$, there exist many possible decompositions tied for the highest achievable score in a given task. As a result, ATAD may select the optimal decomposition when the optimal decomposition is also the highest scoring, but performs suboptimally when this is not the case. In addition to lower reward, this leads to a higher variance in performance by ATAD, as evidenced by Figure 4.4. The Monolithic baseline consistently achieves the

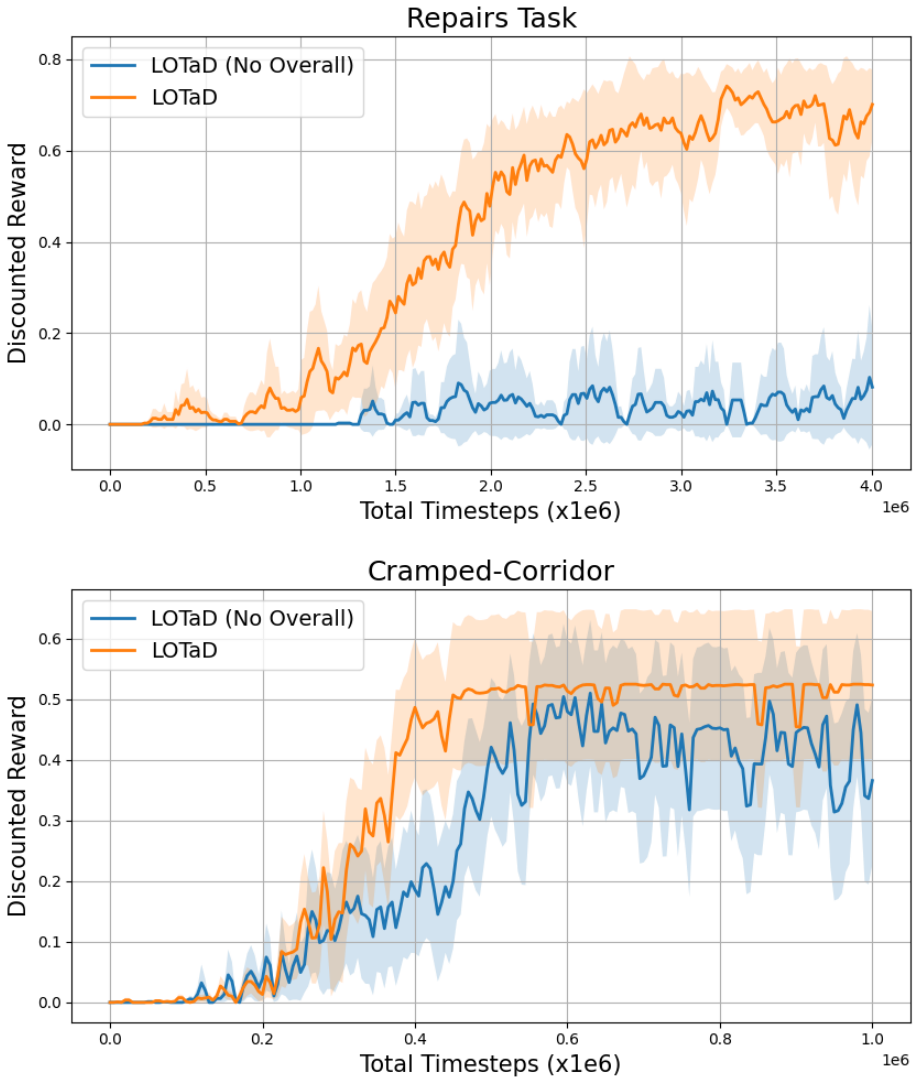


Figure 4.6: Training curves for LOTaD in the Repairs Task and Cramped-Corridor environments demonstrating the effect of conditioning on the overall task state along with individual sub-task states for each agent.

lowest reward due to the sparsity of the overall task reward. Similar to ATAD, we notice that policy learning with the Monolithic baseline is unreliable and tends to converge more slowly than LOTaD.

We find that LOTaD is able to successfully accomplish the task in the Repairs and Overcooked Cramped-Corridor environments, which both involve dependent dynamics amongst agents. LOTaD is indeed able to avoid issues when training agent teams in environments with dependent dynamics, affirming **(RQ2)**. To further investigate our hypothesis of whether a

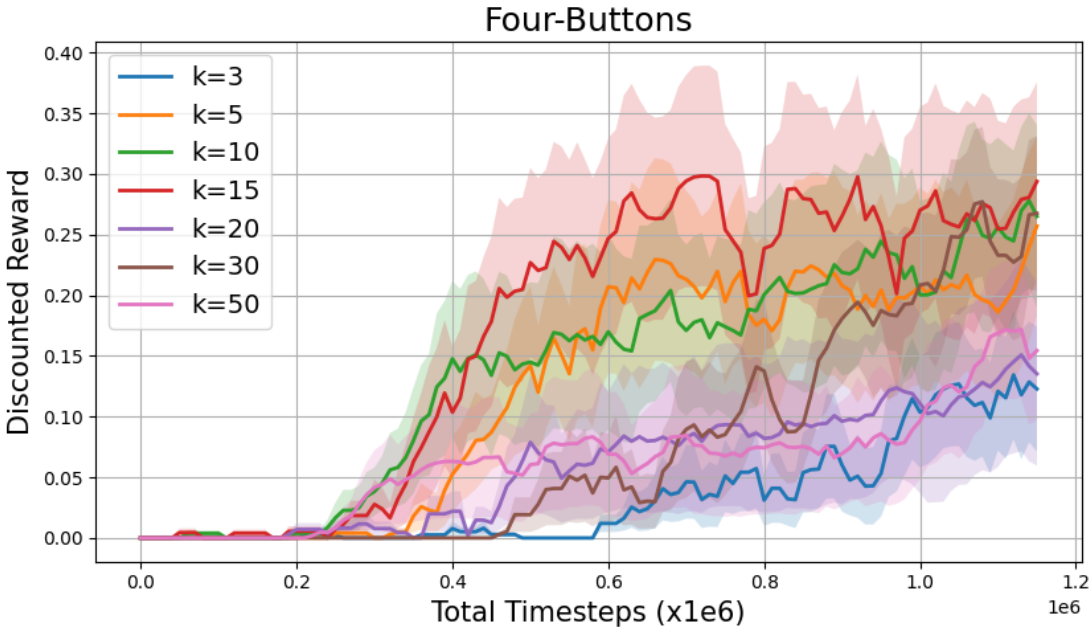


Figure 4.7: Training curves for the UCB selection strategy on increasingly sized $\mathcal{D}$ for the Four-Buttons task. Results are averaged over 10 random seeds.

global task view alleviates these issues, we ran LOTaD *without* an encoding of the overall task in the Repairs and Cramped-Corridor environment. We present the training curves from this ablation study in Figure 4.6. We find that without the overall task encoding, LOTaD is largely unable to accomplish the Repairs task, and is slower and less stable in accomplishing the Cramped-Corridor task. This suggests that the overall task encoding enables our agents to learn policies that progress towards completion of the overall task even when an agent's individual sub-task is already achieved. In our other environment settings, where codependent agent dynamics do not exist, removing the overall task encoding does not significantly affect training.

To answer **(RQ3)**, we perform ablation studies where we vary the size of $\mathcal{D}$. These training curves are visualized in Figures 4.7. We see that values of k that are either very small or very large make the learning problem more challenging for LOTaD. We reason that if k is too large, more exploration is required, and LOTaD may struggle to find the optimal decomposition amongst many candidates. If k is too small, there is a lower chance of "good" decompositions appearing in the set $\mathcal{D}$, which inherently limits the performance potential of LOTaD. We note that the inherent randomness of LOTaD, as well as the choice of environment, task, and the amount of exploration prioritized by our UCB hyperparameter β, may confound results.

4.4 Related Work

Automaton-based task specifications in RL An extensive body of work has explored automaton-based task specification for RL agents. Previous efforts have proposed RL approaches to policy learning for reward machines (Icarte et al. 2022; Toro Icarte et al. 2022; Camacho et al. 2019) or automata-based representations of temporal logic (Voloshin et al. 2022; Voloshin et al. 2023; Hasanbeig et al. 2020; Alur et al. 2022; Sadigh et al. 2014; Fu et al. 2014; Jothimurugan et al. 2019; Shah et al. 2024) by augmenting the state space of the MDP. These efforts focus primarily on single-agent settings and are not designed to handle the MARL case with shared objectives or formally decompose the automaton.

(Symbolic) task decomposition for multi-agent teams Symbolic structures have been leveraged to facilitate efficient multi-agent learning in a variety of settings. A number of these approaches rely on a known dynamics model (Karimadini et al. 2011; Schillinger et al. 2016; Schillinger et al. 2018) for planning-based approaches. In contrast, we assume no prior knowledge of the environment dynamics as is standard in reinforcement learning.

Closely related to our contributions, previous works explored learning decompositions of tasks for teams of agents. Some previous works explored decomposing symbolic tasks by hand designing decompositions or using task- and environment-agnostic heuristics (Neary et al. 2021; Smith et al. 2023). However, these works either require extensive human involvement in determining the optimal decomposition or do not offer a way to choose a decomposition based on MDP dynamics. Moreover, these approaches are limited to MDPs in which the dynamics of agents are independent. Other works learn role assignments for traditional reward functions (i.e., non-symbolic) with multi-agent teams (Wang et al. 2020b) or decompose traditional (non-Markovian) reward functions (Sun et al. 2020) for teams of agents but these methods do not easily extend to non-Markovian rewards which we consider.

Credit assignment in multi-agent RL Credit assignment, originally explored in the single-agent setting (Sutton 1984; Riedmiller et al. 2018; Ferret et al. 2019; Hasselt et al. 2021), is a problem that has been extensively explored in multi-agent learning literature and can be divided between *explicit* and *implicit* solutions. Explicit credit assignment most closely resembles our work, typically assuming that value functions are given in a specific form that allows certain types of decompositions, such as additive decompositions (Nguyen et al. 2017), or value factorization (Wang et al. 2021; Sunehag et al. 2017). Other explicit methods are based on assuming a hierarchical execution structure (Agogino et al. 2004a; Feng et al. 2022; Rashid et al. 2020). Implicit credit assignment instead attempts to perform credit assignment without an explicit structure, for example, by deriving individual policy gradients for each agent derived from a centralized critic (Zhou et al. 2020).

4.5 Conclusion

We introduce a novel approach for learning the optimal decomposition of a task completion RM for MARL through model free interactions with the environment. Our framework simultaneously learns the optimal decomposition and the policies that solve that decomposition amongst a set of candidates. Our experiments show that we improve the sample efficiency of multi-agent learning in model-free deep RL settings even when the optimal decomposition is not known a priori. Through our ablations, we have shown that both including the encoding of the overall task and intelligent decomposition selection is critical for sample efficient learning of reward machines in multiagent settings.

Limitations

Our proposed approach, LOTaD, searches through a pre-defined set of decompositions $\mathcal{D}$ to efficiently find a decomposition that optimally solves the task. However, the quality of our found decomposition will only be as good as the quality of $\mathcal{D}$. If all candidates in $\mathcal{D}$ are highly suboptimal, our approach will be limited in its efficacy. To avoid this, we use on a decomposition generation technique that relies on heuristics to create balanced decompositions amongst agents. However, there is no way to ensure a priori that these decompositions will be successful in the context of the unknown MDP dynamics. Studying how the generation of $\mathcal{D}$ affects the learning outcome of LOTaD is left as a compelling direction for future work.

One of LOTaD's key features is that it provides a *global* view of the overall task to each agent, so that agents can coordinate within the MDP to solve the overall task, even after having solved their respective sub-tasks. Providing this global view (in the form of the overall task RM) requires agents to receive the same information from a labeling function when environment events occur. This level of information sharing may be impractical in some cases. However, we reiterate that this does not mean that control is shared across agents.

Future Work

We see a number of compelling directions for future work. For example, we are interested in exploring how different representations of an RM may enable greater sample efficiency by exploiting semantic similarity amongst overlapping sub-tasks. Lastly, we are interested in exploiting the inherent curriculum present in decomposition exploration by generalizing our multi-agent setting to solving multiple tasks, building on prior work in goal-conditioned RL on automata-based tasks (Vaezipoor et al. 2021; Qiu et al. 2023).

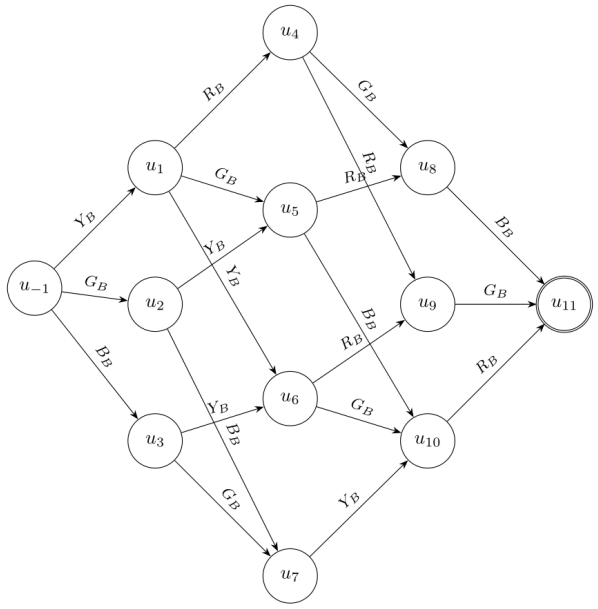
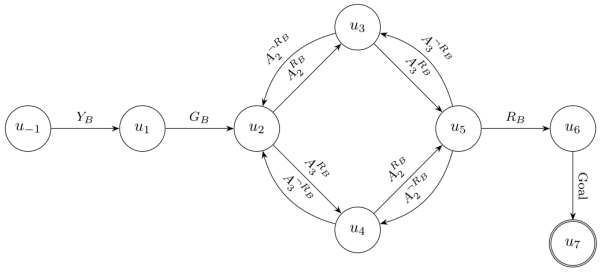
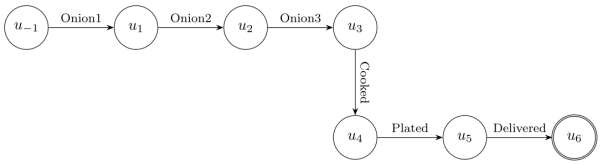
Environment	Reward Machine Task
Four-Buttons	
Cooperative Buttons	
Overcooked	

Table 4.3: Reward machines for various environments. In Four-Buttons, each variable corresponds to whether a button (subscript B) of a certain color (Red, Green, Blue, or Yellow) has been pressed. In Cooperative Buttons, the same notational scheme applies; the additional variables correspond to whether a certain Agent ($A_1, A_2, or A_3$) is or is not currently pressing the Red Button.

Chapter 5

Learning Formal Specifications with Membership and Preference Queries

As we've seen, the body of work that comprises this thesis centers around the usage of *formal logic* to model task specifications for learning-based autonomous agents (Vazquez-Chanlatte et al. 2018; Chou et al. 2020; Pan et al. 2023; Baert et al. 2024). The formal specifications that have been used as rewards in this thesis provide a number of benefits relative to Markovian rewards: they offer a precise notion of satisfaction, are composable, and can be easily transferred and understood across environments. These benefits make logical specifications appealing in safety-critical applications such as planning, verification, and robotics (Webster et al. 2020; Yifru et al. 2023). However, we have yet to discuss *where* these task specifications come from.

In practice, creating formal task specifications often requires intimate knowledge of both the environment and formal logic, which proves difficult for non-experts (Hurley et al. 2024). To avoid this challenge, practitioners have developed methods to automatically learn formal specifications from data. One popular approach is *active* learning, where a non-human membership oracle labels generated agent trajectories as either positive (belongs in the set of desired behaviors) or negative (Bastani et al. 2018; Bongard et al. 2005; Angluin 1987). We aim to extend these approaches to cases where we learn from non-expert teachers, such as humans.While asking only membership queries suffices for completeness in active specification learning, we expect existing approaches to be challenging for human oracles, who are relatively bad at answering membership queries (Palan et al. 2019; Burton et al. 2021; Phelps et al. 2015).

To address this challenge, we identify *preference querying* as a promising alternative to membership querying, where the human is asked to rank two trajectories. Preferences are relatively inexpensive to obtain, are generally preferred by human teachers, and are known to be less susceptible to mislabeling than membership query responses (Palan et al. 2019; Burton et al. 2021; Phelps et al. 2015). Moreso, the combination of the two signals is promising: *Preference queries are comparatively more accurate but less informative than membership queries* (Palan et al. 2019).

In this chapter, we contribute a general framework that allows for active learning from a combination of preferences and labeled examples. The framework works as follows: First, from existing known facts about the desired behavior, we generate candidate specifications that are consistent with previously observed membership and preference constraints. Next, we use heuristics to generate a preference query and a membership query that will rule out as many incorrect candidate hypotheses as possible. We choose either to ask the preference query or membership query based on previous queries and the user's preferences for each query type. Then, we iteratively generate new candidate hypotheses that are consistent with the newly updated facts, and continue asking queries until the correct specification is found, if one exists. To robustify our approach against a (limited) number of wrong answers, our algorithm can identify and ignore sets of answers that are inconsistent with one another (for example, preferring A to B and B to C but C to A).

In our experiments, we implement the aforementioned framework in two different classes of formal specifications, highlighting the generality of our approach. Our automated query selection process can avoid a substantial number of membership queries by asking additional informed preference queries. This trade-off between queries can be easily configured by setting the relative cost of answering each type of query to the teacher.

Contributions We present the formalism of membership-preserving preferences that allows for specification learning from oracles that can answer comparison and membership queries. To show the feasibility and efficacy of our formalism, we propose a *concept class-agnostic* algorithm for querying oracles and run empirical evaluations on two different domains. Finally, we provide a novel method based on Boolean satisfiability (SAT) for identifying DFAs from labeled examples *and* pair-wise preferences.

5.1 Learning with Membership-Respecting Preferences

We begin by developing the machinery to specify which behaviors within a set are desirable, both in a relative and satisficing sense. For a given *universe*, $\mathcal{U}$, containing *atoms* $x \in \mathcal{U}$, a formal specification, or *concept* $\varphi \subseteq \mathcal{U}$, contains a set of atoms. We denote by $\varphi(x)$ the indicator, $[x \in \varphi]$ and refer to a collection of concepts, Φ, as a *concept class*[1]. W.l.o.g., we assume that the universe coincides with the union of the concepts in a concept class, $\mathcal{U} = \bigcup \Phi$.

Example 1. *Concept classes and their universes can be finite or infinite. For example, when representing formal task specifications, one takes as the (infinite) universe, Σ^*, i.e., all words from a finite alphabet Σ. Similarly, languages represented by classes of automata, e.g., DFAs, are (infinite) concept classes.*

[1]For simplicity, we conflate a concept with its representation.

Next, we formalize the notion of pairwise preferences on atoms via *preorders*, $\preceq$, on universes, i.e., transitive and reflexive relations on $\mathcal{U}$. Namely, we call a (fixed) preorder on a universe a *preference order* and interpret $x \preceq y$ as *"y may be preferred over x"*. We write $x \prec y$ if $x \preceq y$ and not $y \preceq x$, interpreted as *"y is preferred over x"*. Two atoms have equal preference, $x \equiv y$, if $(x \preceq y) \wedge (y \preceq x)$. Finally, two atoms are *incomparable*, $x \,||\, y$, if $\neg(x \preceq y \vee y \preceq x)$.

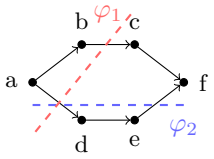


Figure 5.1: A preference order over atoms, represented as a Hasse Diagram.

Example 2. *An example preference order over the universe* $\{a, b, c, d, e, f\}$*, is shown in Fig 5.1. It is represented using a directed acyclic graph,* $H = (V, E)$ *called a* Hasse diagram. *The nodes of* H *represent equivalence classes, i.e., for all* $v \in V$*,* $x, y \in v \implies x \equiv y$*, and the edges of* H *represent strict preferences, i.e.,* $(x, y) \in E \implies x \prec y$*. The full preference order is the transitive reduction of* H*.*

Example 3. *Costs and rewards offer a common way to define preference orders. For example, let* $x \in \mathcal{U}$ *denote the set of paths through a maze and assign a cost,* $c(x) \in \mathbb{R}$*, to each path* x *based on its length. A natural preference order is then given by comparing costs:* $c(x) \leq c(y)$ *implies* $y \preceq x$*. This order is total (no atoms are incomparable) and illustrates that two distinct atoms can have equal preference, i.e.,* $c(x) = c(y)$ *implies* $x \equiv y$*, but not* $x = y$*.*

Learning with Preferences

We now turn our attention to identifying an unknown concept, φ^*. Namely, we shall assume access to a *membership oracle*, $\mathcal{M} \colon \mathcal{U} \to \{\in, \notin\}$, to evaluate if $x \in \varphi^*$, as well as a comparison oracle, $\mathcal{C} \colon \mathcal{U}^2 \to \{\prec, \|, \succ, \equiv\}$, to provide preferences between atoms, e.g., $[C(x, y) = \prec]$ iff $[x \prec y]$. Invocations of these oracles are *queries*.

Problem Statement: Let φ be an *unknown* specification in concept class Φ_{init}. Given membership and comparison oracles $\mathcal{M}$ and $\mathcal{C}$, infer φ.

Remark 5.1.1. *For finite concept classes it suffices to consider only the membership oracle* $\mathcal{M}$*, ignoring query and time complexity. In particular, one may pair-wise consider all concepts and pose a membership query from the symmetric difference of the concepts to uniquely identify the concept. However, for many domains (Burton et al. 2021), obtaining accurate membership oracles is expensive. The key question in this work is* how to exploit the availability of the comparison oracle $\mathcal{C}$*.*

To leverage the comparison oracle, we relate preferences to membership in φ^*. We therefore focus on preference orders that respect membership: atoms outside of the concept, $x \notin \varphi^*$, cannot be preferred to atoms in the concept, $y \in \varphi^*$.

Definition 5.1.1. *A preference order is a* membership-respecting preference *(MemReP) w.r.t. φ if*

$$x \preceq y \implies \varphi(x) \leq \varphi(y). \tag{5.1}$$

Example 4. *All preference orders are membership respecting w.r.t. concepts $\top \stackrel{\text{def}}{=} \mathcal{U}$ and $\bot \stackrel{\text{def}}{=} \emptyset$. Similarly, any oracle that yields $C(x, y) = \|$ for all $x, y \in \mathcal{U}$ is vacuously membership-respecting.*

Example 5. *Cost-based preferences, like in our path example, together with thresholded cost concepts, i.e., $\varphi_\delta \stackrel{\text{def}}{=} \{x \in \mathcal{U} : c(x) \leq \delta\}$, for some cost map $c\colon \mathcal{U} \to \mathbb{R}$ and threshold $\delta \in \mathbb{R}$, are membership respecting.*

Example 6. *We continue with Ex. 2. This order is membership-preserving for $\varphi_1 = \{a, b\}$, but not for $\varphi_2 = \{d, e\}$. Graphically, φ_2 is not membership-preserving as there is an edge from $\bar{\varphi_2}$ to φ_2.*

Example 6 illustrates that the MemReP assumption is sometimes strong enough to distinguish concepts without any membership queries. Namely, if there exists a pair of atoms, $x \in \varphi_1 \setminus \varphi_2$ and $y \in \varphi_2 \setminus \varphi_1$, such that $\mathcal{C}(x, y) \neq ||$, then $\mathcal{C}(x, y)$ is guaranteed to distinguish φ_1 and φ_2 under the MemReP assumption.

Definition 5.1.2. *Let $X = (X_\mathcal{M}, X_\mathcal{C})$ be a tuple of sets of membership and comparison labeled examples respectively, e.g.,*

$$X_\mathcal{M} = \{(x_1, \in), \ldots, (x_j, \notin)\}$$
$$X_\mathcal{C} = \{(x_k, y_k, \|), \ldots, (x_n, y_n, \prec)\}.$$

A concept φ is consistent *with X if (i) φ agrees with all membership assignments in $X_\mathcal{M}$, e.g., $(x, \in) \in X_\mathcal{M} \implies x \in \varphi$; and (ii) All preferences in $X_\mathcal{C}$ are membership respecting under φ, e.g., $(x, y, \prec \varphi(x) \leq \varphi(y)$. Given a concept class Φ, we denote by Φ^X the set of all concepts in Φ consistent with X.*

Abstract Algorithm

We now return to the problem of identifying an unknown concept given access to a membership oracle and a membership respecting comparison oracle. In Alg. 4, we outline the general process for learning concepts from *finite* concept classes given such oracles. The learner begins with a concept class, asks a membership or comparison query, and then removes concepts that are inconsistent with the query result until a unique concept remains.

Algorithm 4: Generic algorithm for concept identification.

Input: Φ_{init}
Output: Φ^X ; // $\Phi^X = \emptyset$ or $\Phi^X = \{\phi^*\}$
Assign Φ as the initial concept class Φ_{init};
Initialize the set of labeled examples X as $(\emptyset, \emptyset)$;
while $|\Phi| > 1$ **do**
  Ask either a membership or a comparison query given X and Φ^X;
  Add the result to X;

Proposition 1. *Suppose for every iteration, the probability of asking a distinguishing membership query, i.e., asking $\mathcal{M}(x)$ for x in the symmetric difference of two concepts in Φ^X, is bounded from below. Then, Alg 4 almost surely terminates.*

Remark 5.1.2. *To treat infinite concept classes, we appeal to Occam's Razor, and seek to find the "simplest" concept that is consistent with the data. Formally, we assume that the finite concept class is the (countably infinite) union of finite concept classes, $\Phi = \bigcup_{i=1}^{\infty} \Phi_i$, where i, is taken as a complexity measure, e.g, number of states of an automaton. We extend the above process by seeking to find the smallest i such that the above process returns a singleton. Finally, as is standard in this setting, we will additionally assume access to an equivalence oracle to provide completeness guarantees. In practice, such equivalence queries are often impractical, and are approximated via conformance testing and sampling.*

In order to realize this algorithm in practice, we require three ingredients. First, we must develop methods that can synthesize a consistent concept over both membership query and preference query results for a specific concept class. This operation enables symbolically interacting with Φ^X in Alg. 4. We provide synthesis methods for the concept classes used in our experiments in the appendix. Second is an intelligent means to select a query for $\mathcal{M}$ or $\mathcal{C}$, which we expand on in the sequel. Last is a means to handling the non-trivial possibility of labeling mistakes.

5.2 Asking the Right Queries

We now discuss a concept class agnostic strategy to select queries for an efficient version of Alg. 4.

Cost model

Before we optimize our algorithm, we must set the measure that we aim to optimize. In any active learning algorithm, the selection of the queries is central to its performance. In particular, we seek to balance minimizing the number of queries asked and computational costs. To this end, we model the costs of preference and membership queries to be time

invariant and constant, based on a weighted sum:

$$\text{cost}(\text{queries}) \triangleq a \cdot \#\text{mem} + b \cdot \#\text{pref}, \tag{5.2}$$

where #mem and #pref refer to the number of membership and preference queries and $a, b \in \mathbb{R}_\infty$. For example, $a = b$ treats membership and comparisons interchangeably and $a = \infty, b = 1$ lexicographically prefers comparisons over membership queries[2].

Contextual Bandit Formulation

Because the results of the queries are a priori unknown, naïvely optimizing a given cost model is often infeasible. Furthermore, as the next example illustrates, adversarial teachers can induce arbitrary regret.

Example 7. *Recall the lexicographic example, $a = \infty, b = 1$. If all preferences yield incomparable, then one would regret making any comparison queries. On the other hand, if one ignores comparisons, but the underlying preference lattice is total as in Ex. 5, then one may ask many more membership queries than is required. In particular, if the (total) preference order were known, then the learner could binary search (using $\mathcal{M}$) for the unique point where membership changes.*

Further, we highlight that, even if a model for query responses is known, optimally planning a sequence of queries is often computationally intractable. There are $M = \binom{|\mathcal{U}|}{2} + \mathcal{U}$ queries to ask in a single step, and thus roughly 3^{M^t} possible combinations when asking up to t queries. Thus, even assuming $\mathcal{U}$ is finite, but non-trivially small, e.g., strings of length at most 10, the search space quickly becomes intractable.

Thus, we propose a two stage formulation to optimize the query selection based on adversarial contextual multi-armed bandits (CMABs) (Auer et al. 2002b), specifically, multi-armed bandits with expert advice. In this formulation, the classic multi-armed bandit framework is extended by introducing *experts* that consider the arms and context in the problem, and provide recommendations in the form of probability distributions over the arms that our algorithm can then make use of in its policy. In order to instantiate our CMAB algorithm with experts, we make the following choices: (i) We use a heuristic to select candidate comparison and membership queries, the *arms* in our formulation. The heuristic is a (Monte Carlo) estimate of the concept class size's reduction for each query's possible outcome. (ii) We define a (bounded) proxy cost per arm weighing the query cost against the (estimated) reduction in the concept class:

$$\text{loss}_c \stackrel{\text{def}}{=} \frac{c}{\max(a, b)} \cdot \frac{|\Phi'|}{|\Phi|} \in [0, 1], \tag{5.3}$$

[2]Our algorithm can handle arbitrary cost models over #mem and #pref and is not dependent on the specific structure of equation 5.2.

Algorithm 5: CMAB round for selecting a query.

Select comparison and membership queries via Alg. 6;
Estimate the loss (5.3) for different query outcomes;
Provide average and worst-case expert advice distributions on the arms E_1, E_2 (see Sec. 5.2);
Sample which expert, $E \sim \{E_1, E_2\}$, to listen to based on historical performance;
Sample the arm (query) based on E's arm distribution;
Compute the actual loss and update expert distributions;

Algorithm 6: Query selection heuristic.

Select a set of up to α unrefuted concepts, $\Psi \subseteq \Phi$;
Let X be a set of atoms such that:
(i) $|X| \in [\alpha, \beta]$ and
(ii) X distinguishes concepts in Ψ, i.e.,

$$\forall \phi_1, \phi_2 \in \Psi. \exists x \in X. x \in \phi_1 \Delta \phi_2.$$

Find $x \in X$ that minimizes the (worst-case) number of concepts in Ψ consistent after $\mathcal{M}(x)$;
Find $y, z \in X$ that minimize the (worst-case) number of concepts in Ψ consistent after $\mathcal{C}(y, z)$;
return candidate queries $\langle \mathcal{M}(x), \mathcal{C}(y, z) \rangle$;

where c is the cost of the selected arm's query type (thus either a or b). Finally, (iii) we encode two heuristic query strategies as *experts* assigning probabilities to each arm, described in Section 5.2. The resulting CMAB game proceeds in rounds described in Alg. 5, each round corresponds to an iteration in Alg. 4.

To avoid considering all atoms in $\mathcal{U}$, we propose the concept class agnostic heuristic in Alg. 6 to select candidate membership and comparison queries. For our implementation, we use $\alpha = 2$, and take at most two atoms per concept, i.e., $\beta = 2\alpha$.

Worst and average case advice

We propose to give advice towards either of the two queries based on combining two perspectives. In particular, we propose to use the following two experts with the following advice arm distributions: **(1) Pessimistic**: Weighs each arm by its worst-case loss. Ignores the incomparable answer for preference queries. **(2) Historical**: Weighs each arm by its expected loss, computed by averaging the previous losses incurred by pulling that particular arm.

To construct the advice distribution, we take the softmax of the weights of the comparison and membership queries. The expert selection distribution is then updated using the

standard exp4 contextual bandit algorithm (Auer et al. 2002b) which guarantees that we will switch between our pessimistic and historical arm strategy in a manner such that, with respect to loss (5.3), the algorithm would not have done much better sticking to advice from a single expert.

Let us share the following intuitions for our choice of experts. The pessimistic expert helps to ensure some notion of progress. However, as an incomparable answer for the comparisons never yields any progress, we exclude this case from the expert. On the other hand, we have seen in Ex. 7 that the utility of a comparison query depends heavily on the oracle's underlying preference order. To capture this observation, the historical expert learns whenever almost all comparisons yield incomparable and will promote using membership in these cases. Likewise, this expert will discover when there is a total preference order and all atoms are comparable, and perhaps even equivalent. Finally, observe the following proposition about the proposed algorithm:

Proposition 2. *Assume the unknown concept, φ^*, is in Φ. Running Alg 4 using Algs 5,6 for queries (almost surely) identifies φ^* after finitely many queries.*

Proof Sketch. Under exp4, a series of unproductive preference queries, i.e., ones that do not change Φ^X, will exponentially increase the weight of the historical expert. Similarly, the historical expert will exponentially increase the weight of the distinguishing membership query arm. Finally, because the per round loss is bounded, there exist a lower bound on asking distinguishing membership query. By Prop 1 the algorithm almost surely requires finite queries. □

Handling Error

Our concept learning algorithm can be adapted to gracefully handle two kinds of labeling errors: (i) *Preorder violations* and (ii) *MemRep violations*. A preorder violation occurs when the underlying order relation is observed to not be transitive or reflexive. This can be visualized using a Hasse Diagram where such a violation would either correspond to a cycle or an inconsistency in the node equivalence classes. Similarly, a MemReP violation occurs when the Hasse diagram contains an edge, (x, y), where $x \prec y$, but $x \in \varphi^*$ and $y \notin \varphi^*$.

Both classes of violations are easy to detect and isolate. In an explicit Hasse diagram representation, a topological pass over the graph suffices. In the case of SAT based concept classes, e.g., our DFA learning experiment in the following subsection, we simply analyze the UNSAT-core to determine which queries must be dropped to find a consistent hypothesis. In either setting, one can alert the user, allowing the violating query responses to be dropped or corrected before resuming the learning algorithm. Combining with a final conformance tester which asks additional redundant queries from a test distribution, yields a probably approximately correct concept Kearns et al. 1994.

5.3 Implementation and Experiments

In this section, we instantiate Alg. 4 on two families of concept classes, demonstrating the flexibility and efficacy of the proposed formalism and heuristics.

Deterministic Finite Automata

To begin, we apply the algorithm and operations outlined in the previous subsection to the space of DFA identification, relying on the performance of modern Boolean satisfiability (SAT) solvers.

A *Deterministic Finite Automaton*, DFA, is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of a finite set Q of *states*, a finite *alphabet* Σ, a *transition function*, $\delta \colon Q \times \Sigma \to Q$, an *initial state* q_0, and the *accepting states* $F \subseteq Q$. The function $\delta^* \colon \Sigma^* \to Q$ denotes the lifting of δ to sequences of symbols (strings), via repeated application. Finally, the language of DFA $\mathcal{D} = \{w \in \Sigma^* \mid \delta^*(w) \in F\}$ is set of strings that reach an accepting state, $q \in F$.

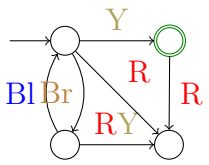


Figure 5.2: A DFA encoding the considered task.

Implementation

In order to realize our algorithm for DFA learning, we need to support synthesizing a DFA consistent with *labeled examples* V_+, V_- and an observed set of preferences, $V_\prec$, i.e. the results of previously observed membership and comparison queries respectively.

We extend the SAT encoding presented in (Ulyantsev et al. 2015; Heule et al. 2010) for the passive identification of DFAs from positive and negative examples to support membership respecting preferences. The encodings operate by using the provided positive and negative examples to form a prefix tree. The nodes indexed by positive and negative examples are annotated with whether they accept or reject. Two states can be merged if they are indistinguishable in the resulting transition system. This feature, together with the determinism of a DFA, are captured by transforming the problem into a k-color graph coloring problem, where k is the fixed size of the DFA that is to be identified. The resulting graph coloring problem is then encoded as a Boolean satisfiability (SAT) query. More specifically, for each labeled word, v, the SAT encoding includes (i) a variable $x_{v,i}$ indicating if word v accesses state (color) i and (ii) a variable z_i indicating whether state (color) i is accepting.

Encoding

We extend the existing encoding to incorporate membership respecting preferences. Using these variables, a membership preserving preference, $(w, v) \in V_\prec$, can be encoded using the

following constraints:

$$\forall i, j \;.\; (w \prec_\varphi v) \in V_\prec : \underbrace{(x_{w,j} \wedge x_{v,i})}_{v \;\&\; w \text{ access } i \;\&\; j} \implies \underbrace{(z_j \implies z_i)}_{z_j \implies z_i \text{ is equiv to } z_j \leq z_i} \tag{5.4}$$

This constraint formalizes the previously mentioned logic that non-preferred trajectories' acceptance will lead to preferred trajectories' acceptance, and preferred trajectories' rejection will lead to non-preferred trajectories' rejection. If this were not the case, the MemReP condition would lead to a contradiction.

Learning Task Specifications for Robots We study the problem of conveying a task to an agent (robot) moving about an environment. For example, consider the task specification that says "avoid red (lava) tiles and do reach a yellow (recharge) tile (RY)" and "between visiting blue (water) tiles and yellow tiles, the agent must visit a brown (dryer) tile. (BBY)" from (Vazquez-Chanlatte et al. 2018). This task specification is described by the DFA shown in Fig 5.2. We assume that the robot is pre-programmed to universally assume that a task will contain the RY constraint and seeks to interactively learn the domain-specific BBY constraint (in the form of a DFA) from a user.

First, observe that because DFAs are closed under conjunction, a new DFA can be easily derived that will not violate the a priori specified RY task. Further, depending on the system dynamics, the user may prefer some traces to others, and is able to provide these pair-wise comparisons. Thus, we pose this task learning process as an active learning problem, where the user can provide (i) *pair wise preferences* and (ii) *membership query responses (labels)* that specify whether an example is good or bad. We want to leverage that query mechanism to potentially accelerate or robustify the learning of the DFA.

The oracle uses a randomly generated membership respecting preference order, where approximately $\frac{1}{10}$ of the comparisons yield incomparable. To refute equivalence queries, we sample a random string from the symmetric difference of the true DFA and the current hypothesis. Finally, we fix the comparison cost to 1 and vary the membership cost to study the trade off in queries the algorithm makes.

The results of the experiment are shown in Fig 5.3. We observe that, as expected, as the membership cost increases (i) the total number of queries increases (ii) the total number of membership queries decreases. Specifically, *removing 1 membership query adds an average of 2.03 preference queries* each time the cost doubles. Additionally, the use of preference queries allows the use of membership queries to be well below the baseline rate of setting the costs such that only membership queries are used; that is, setting the cost of a preference to ∞.

Finally, we compare with the discriminate tree variant of L* (Kearns et al. 1994; Angluin 1987), a classic algorithm for learning DFAs from membership and equivalence queries. We use the AALpy library (Muškardin et al. 2022) implementation of L*, and find that the implementation requires 11 membership queries and 2 equivalence queries (the same number as Fig 5.3) that cannot be answered by the RY task prior knowledge. While our algorithm

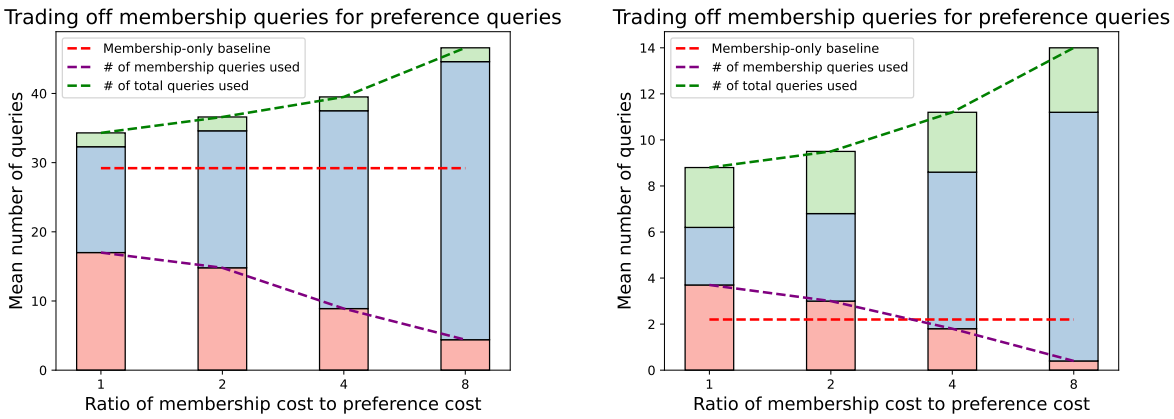


Figure 5.3: Trade-off between preference queries and membership queries in the DFA domain (Left) and the Monotone Predicates domain (Right). The bars plotted show the contribution of membership (red, bottom), preference (blue, middle), and equivalence (green, top) queries.

requires more overall queries, we observe that the number of membership queries can be fewer than 11 as at membership cost 4, all without concept class specific tailoring. Furthermore, we remark that these two algorithms could be used in concert by adapting the membership query selection to use L* provided queries.

In order to further study the performance of our algorithm in learning DFAs, we performed the following studies: First, we applied our algorithm to learning the Tomita Language DFAs, a standard benchmark in DFA learning, to evaluate performance across a diversity of DFAs. Next, we evaluated our algorithm on classes of DFAs that vary in size to evaluate how our algorithm scales as the target DFA increases in complexity. Finally, we implemented and evaluated a method in our algorithm that can identify and tolerate incorrect responses from our oracle.

Tomita Languages

The Tomita languages (Tomita 1982) are a standard set of regular languages and DFAs frequently used as a benchmark in DFA learning and identification. The seven languages have a number of appealing qualities: they are relatively parsimonious, and they collectively span a number of interesting properties, including distributions of accepting and rejecting strings, existence of sink states, and relative ease of identification with a small number of membership queries.

For the Tomita languages, we used a manually designed preference ordering to incoporate more semantic meaning into the ordering and thereby encourage better learning from preference queries themselves. The preference ordering works as follows: Positively labeled atoms are still always preferred over negatively labeled atoms, as expected. When comparing two

	DFA 1		DFA 2		DFA 3		DFA 4		DFA 5		DFA 6		DFA 7	
	O	B	O	B	O	B	O	B	O	B	O	B	O	B
C=1	3.1	3.6	6.6	7.0	5.1	7.3	12.5	16.6	8.4	11.4	5.7	6.7	17.3	18.7
C=2	3.1	3.6	6.2	7.0	4.7	7.3	11.2	16.6	6.6	11.4	5.5	6.7	16.1	18.7
C=4	3.0	3.6	5.3	7.0	3.8	7.3	11.4	16.6	5.6	11.4	5.5	6.7	14.9	18.7
C=8	3.0	3.6	3.5	7.0	2.7	7.3	6.8	16.6	4.2	11.4	5.4	6.7	12.2	18.7

Table 5.1: Mean number of membership queries asked by our membership-and-preference selection algorithm (O) in comparison to the membership-only baseline (B) on the seven Tomita DFAs. Little to no difference is seen in the simpler DFAs, whereas the discrepancy in query amount is more pronounced in more complicated DFAs.

negatively labeled atom, the atom that took longer to reach a sink state (i.e., a rejecting state that cannot be transitioned out from) in the DFA was preferred, if a sink state existed. If neither atom reached a sink state, the atom with a longer accepting prefix was preferred. When comparing two positive atoms, we enumerate the four possible two-token extensions to the atoms and prefer the atom that is more frequently accepted when considering all of the extensions.

In Tables 5.1, 5.2, and 5.3, we show the results for the Tomita languages experiment for our MemRePs algorithm and the membership query-only baseline (where the cost for asking a preference query is set to ∞), averaged across 20 trials. Note that we omit the comparison between the number of equivalence queries asked by each method since these numbers were the same for both. Overall, we notice that the tradeoff between membership and preference queries is apparent as the relative cost between the two increases. However, this tradeoff is more pronounced in some languages (languages #4, 5, and 7) than other (languages #1, 2, and 6). An example language (language #5) is further illustrated in

We also note that the number of preference queries needed in some cases, such as in languages 4 and 7, dramatically increase with the increased cost ratio. The high variance for preference queries needed also indicates that the combination of the CMAB algorithm and atom selection process leaves room for improvement and consistency.

Robustness Experiment

We design our algorithm to be robust in noisy settings, where the response to a query is flipped to be incorrect some proportion of the time. In our algorithm's implementation for DFAs, if a labeling error occurs that causes a violation, the UNSAT-core is extracted to see which existing assumptions caused this violation, and those assumptions are then dropped. We demonstrate the effect of labeling errors in an experiment, where an example DFA (in this case, Tomita language #6) is learned in settings of increasing proportions of labeling errors. The results are displayed in figure 5.5, where an increasing rate of error causes more assumptions that need to be dropped by our algorithm, resulting in more queries

	DFA 1		DFA 2		DFA 3		DFA 4		DFA 5		DFA 6		DFA 7	
	O	B	O	B	O	B	O	B	O	B	O	B	O	B
C=1	.13	.23	.74	0	.66	.49	1.6	1.6	1.3	1.0	.67	.50	2.0	1.8
C=2	.43	.23	1.0	0	.61	.49	1.6	1.6	1.2	1.0	.92	.50	.87	1.8
C=4	.22	.23	.78	0	.87	.49	1.6	1.6	.64	1.0	.81	.50	2.2	1.8
C=8	.41	.23	.67	0	.64	.49	.97	1.6	1.1	1.0	.49	.50	1.3	1.8

Table 5.2: Variances for number of membership queries asked by our membership-and-preference selection algorithm in comparison to the membership-only baseline on the seven Tomita DFAs.

	DFA 1		DFA 2		DFA 3		DFA 4		DFA 5		DFA 6		DFA 7	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
C=1	1.4	.75	4.7	1.1	4.6	2.7	10.5	6.4	5.1	2.1	3.3	1.1	11.5	4.2
C=2	2.6	.88	5.2	.92	5.0	1.7	25.1	11.1	7.0	1.8	3.3	2.3	22.0	9.3
C=4	3.1	.79	6.3	.83	9.7	2.1	41.3	12.7	8.1	3.0	3.7	1.4	44.3	8.2
C=8	3.8	.83	8.1	.77	15.9	2.3	84.3	17.2	9.1	5.1	4.1	1.3	78.7	16.7

Table 5.3: Mean and variance for number of preference queries asked by our membership-and-preference selection algorithm on the seven Tomita DFAs.

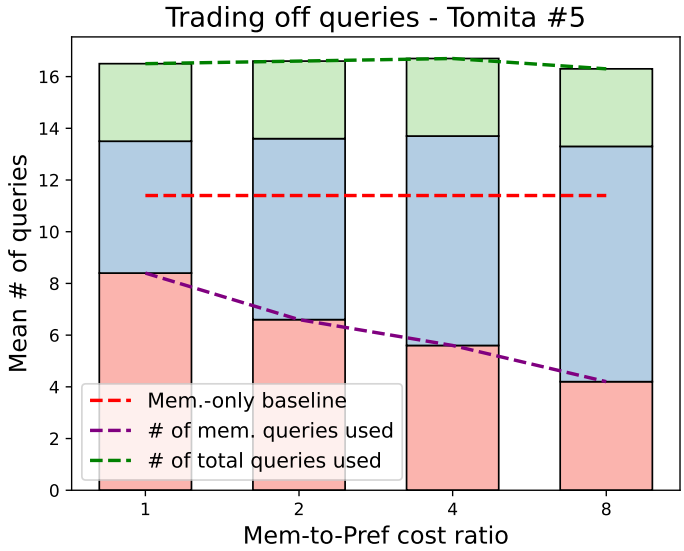


Figure 5.4: Trade-off between preference queries and membership queries for tomita language #5. The bars plotted show the contribution of membership (red, bottom), preference (blue, middle), and equivalence (green, top) queries.

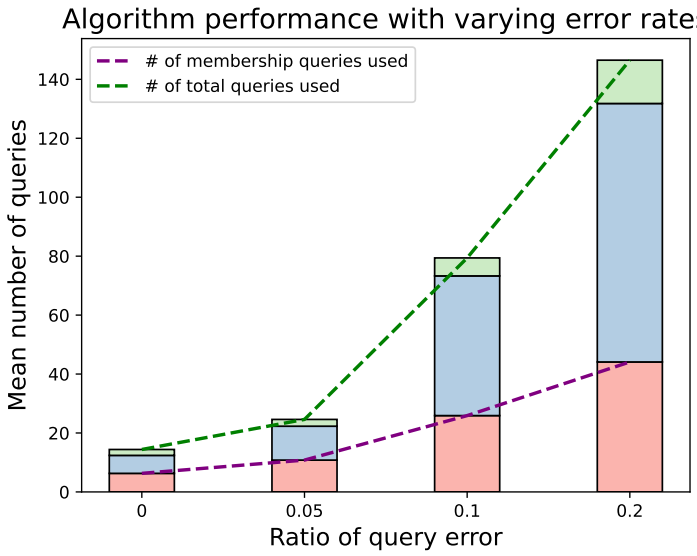


Figure 5.5: Greater numbers of queries are required to learn the target DFA (Tomita language #6) as the error rate increases. The bars plotted show the contribution of membership (red, bottom), preference (blue, middle), and equivalence (green, top) queries.

required to learn the correct concept. In other words, as errors are made more frequently and assumptions are discarded more often, it becomes harder for the necessary set of assumptions to be efficiently obtained by the learner.

Scalability Experiments

To understand how our algorithm scales as our target DFA increases in complexity, we evaluate our algorithm's scalability on a simple one-symbol language that determines whether the length of an input sequence is modulo some positive integer k. The size of the DFA is k states with a single accepting state. We provide the results of our algorithm's performance as k increases in Table 5.4. Not included in the table is the number of equivalence queries used in each DFA, which did not vary as a function of membership-to-preference cost ratio. The mean number of equivalence queries asked were 6.4, 9.1, 9.9, and 22.6 for DFA states 5, 10, 20, and 40, respectively. The equivalence queries in this setting were highly informative, allowing the number of other query types to scale efficiently but somewhat restricting those queries' utility.

In addition to the previous experiment, we generalized Tomita Language #4, which originally is defined as a 3-state DFA that encodes the task "any string without more than 2 consecutive '0's", to any string with more than n consecutive '0's. The size of the DFA in states is $n + 2$, including the rejecting sink state. We vary n from 1 to 4 and present

	5 States		10 States		20 States		40 States	
	# Mem.	# Pref.	# Mem.	# Pref.	# Mem.	# Pref.	# Mem.	# Pref.
Cost=1	2.2	1.2	5.7	3.6	9.4	7.4	18.7	7.8
Cost=2	2.0	1.5	4.7	5.6	8.1	8.4	17.9	9.9
Cost=4	1.4	2.6	4.6	6.6	6.2	12.2	15.4	19.5
Cost=8	1.0	2.9	3.9	10.9	3.7	17.8	11.3	31.4

Table 5.4: Number of Membership and Preference Queries for the scaled modulo DFA structure, averaged over ten trials.

	3 States		4 States		5 States		6 States	
	# Mem.	# Pref.	# Mem.	# Pref.	# Mem.	# Pref.	# Mem.	# Pref.
Cost=1	7.3	3.1	12.5	10.5	23.3	14.3	32.4	34.6
Cost=2	6.5	6.3	11.2	25.1	20.3	26.9	28.6	59.0
Cost=4	5.6	17.7	11.4	41.3	19.1	71.7	25.8	143.6
Cost=8	5.1	34.1	6.8	84.3	17.9	147.2	23.6	218.6

Table 5.5: Number of Membership and Preference Queries for the scaled Tomita #4 experiment, averaged over ten trials. The number of equivalence queries remained constant over costs.

our results in Table 5.5. We note that the number of queries required quickly increases with the increase in number of states, which is to be expected given the super-linear increase in search space size with number of states.

Monotone Predicate Families

Next, we study monotone predicate families. A *monotone predicate family* is a concept class with an (arbitrary but fixed) partial order $\sqsubset$ defined over the concepts such that $\varphi \sqsubset \varphi'$ implies $\varphi \subseteq \varphi'$. Increasing a concept thus monotonically increases the set of atoms included by the concept. We motivate studying monotone predicates using a series of motivating examples.

Example 8. *We consider the on-boarding process of a hypothetical car that queries the user to learn what (safe) distances to other objects they deem comfortable. In the scope of this chapter: (a) The resulting behavior should* never *violate any pre-defined safety constraints. (b) The on-boarding experience should be brief, i.e., the system should try to minimize the number of queries. (c) Communication should be unambiguous and concrete to cover edge cases.*

The above example can be cast as a 2-dimensional monotone predicate family, where one dimension corresponds to maximum time, $\tau \in [0, T]$, the user is willing to wait to reach the

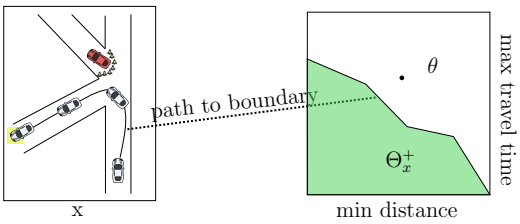


Figure 5.6: Mapping Ex. 8 to a geometric perspective of the considered concept class.

destination and the other dimension corresponds to the minimum distance, $d \in [0, D]$, to another car the user is comfortable with. The corresponding partial order has $(\tau, d) \sqsubset (\tau', d')$ if $\tau < \tau'$ and $d > d'$.

Geometric Perspective In order to study the generic behavior of our algorithm on monotone predicate families, it helps to focus on a geometric interpretation of the concept class. For that, we assume that we adequately parameterize the concepts in a concept class $\Phi = \{\varphi_\theta \mid \theta \in [0,1]^d\}$. The parameters θ induce a natural pointwise (or product) order $<$ with $\theta < \theta'$ if for all $i < d$. $\theta_i < \theta'_i$. It is then straightforward to define the order on the concepts as $\varphi_\theta \sqsubset \varphi_{\theta'}$ if $\theta < \theta'$. Next, observe that any atom, $x \in \mathcal{U}$, partitions the parameter space into two regions,

$$\Theta_x^+ \stackrel{\text{def}}{=} \{\theta \mid x \in \varphi_\theta\} \qquad \Theta_x^- \stackrel{\text{def}}{=} [0,1]^d \setminus \Theta^+, \tag{5.5}$$

called a *monotone bi-partition*. With this perspective, membership is entirely determined by whether or not the underlying concept's parameter, θ, is in the accepting set,

$$x \in \varphi_\theta \iff \theta \in \Theta^+. \tag{5.6}$$

Example 9. *For instance, scaling and reflecting the parameters, $\theta = \left(\frac{\tau}{T}, \frac{D-d}{D}\right)$, yields a monotone parametric family. Fig 5.6 illustrates. In particular, the ego (white) car wants to go to the yellow region within some time budget, while maintaining a minimum distance to the red parked car. This gets mapped to a bi-partition in the normalized parameter space.*

Thresholded rewards Monotone predicate families include task specifications as thresholded rewards to weighted sums over multi-dimensional rewards. In particular, let $\vec{f}(i) \in [0,1]^d$ be a sequence feature vector for $i \in [1, N]$, for $N \in \mathbb{N}$. Thresholded sums of linear rewards on these feature vectors can be cast as a $d+1$ dimensional monotone predicate family. In particular, let the thresholded sums be defined as $\sum_{i=1}^{N} w \cdot f(i) > \delta$ for $w \in [-1,1]^d, \delta \in [-d, d]$. We may define θ using:

$$\theta_j = \begin{cases} (1 - w_j)/2 & \text{if } j \leq d \\ (\frac{\delta}{d} + 1)/2 & \text{if } j = d+1 \end{cases}. \tag{5.7}$$

Remark 5.3.1. *If the $\mathcal{C}(x, y)$ is total, one derives a (noiseless) variant of the learning setting considered in Xu et al. 2020a.*

We seek to understand the trade-off between membership and comparisons incurred by our algorithm given different membership and comparison costs. We instantiated our algorithm with a size-indexed concept class, where each Φ_i corresponds to a uniform 2d grid of parameters with i points per axis. Equivalence queries provide a labeled bi-partition separating the concept. Like with our DFA experiment, the preference order was randomly generated such that approximately $\frac{1}{10}$ on atoms are incomparable and approximately $\frac{1}{3}$ of atoms whose preference is not forced by the MemReP condition are strictly ordered. Furthermore, as a baseline, we compare against an learner that only uses equivalence and membership queries by setting the comparison query cost sufficiently high.

Fig. 5.3 shows the averaged results on 100 randomly generated concepts with the comparison cost fixed to 1 and the membership cost changing on the horizontal-axis. The vertical axis corresponds to the average of number of queries across all preference orders. First, the average number of equivalence queries was the same for each instance on the same concept, including the shown membership only base-line. Furthermore, as desired, increasing the cost of the membership queries results in the average number of membership queries decreasing, at the expense of additional comparison queries: as the relative cost doubles, *removing 1 membership query adds on average 2.4 membership queries.* This is expected given that comparisons provide less information about the concept's label than a membership query.

We also observe that initially, introducing preferences occasionally *increases* the number of membership queries. This effect is due to: (i) the greedy nature of our algorithm, meaning that "good" membership queries are ignored as they correlate with "good" preference queries, and (ii) our hyperparameters - namely the temperature of the softmax in the expert advice - was tuned with the assumption that membership would cost significantly more than preferences.

5.4 Related Work

Active learning of rewards using preferences. Inverse reinforcement learning (IRL) (Ng et al. 2000; Abbeel et al. 2004) often relies on high-quality demonstrations to learn a reward function. More recently, works have proposed approaching IRL from an active learning perspective, asking a teacher for information-rich feedback on learner-generated examples, such as corrected examples or labels on sections of generated examples (Hadfield-Menell et al. 2016; Brown et al. 2018). Preference-based reward learning has emerged from these active approaches as a popular method due to the accuracy and relative inexpensiveness of preference queries (Holladay et al. 2016; Wilson et al. 2012). To overcome the limited information content gained from relative comparisons, various techniques have been devised to actively select preference queries that maximize the amount of information gained (Sadigh et al. 2017; Biyik et al. 2018; Xu et al. 2017; Basu et al. 2019; Xu et al. 2020a), typically by removing maximal volume from the hypothesis space.

Grammatical inference and concept learning. Grammatical inference (De la Higuera 2010) refers to the rich literature on learning a formal grammar (often an automaton) from data. Examples include learning the smallest automata consistent with a set of positive and negative strings (De la Higuera 2010), learning an automaton using membership and equivalence queries (Angluin 1987), and extensions that relax the assumption of equivalence oracles and the ability to ask any membership query (Moeller et al. 2023). Within this literature, the key contributions of our work are to take into account preferences via the concept-agnostic framework. Most similar to our work is (Hsiung et al. 2023), which uses preferences within the L* framework to learn reward machines, a specialized type of automaton. This work focuses on preference-based specification learning in a more specific setting and is complementary to our contribution. Finally, our algorithm can be seen as an extension of version space learning (Sverdlik et al. 1992), where we use preference-based learning to build on existing methods that leverage either explicit data structures or SAT-based DFA-identification (Ulyantsev et al. 2015; Heule et al. 2010) to realize candidate elimination.

5.5 Conclusion

In this chapter, we present a generic framework for learning task specifications (concepts) from actively acquired (noisy) preferences and labeled examples. Despite being concept class agnostic, we demonstrated the efficacy of our approach on two very different concept classes.

Nevertheless, interesting future work includes considering principled approaches to deriving domain-specific optimizations of the heuristics used in our framework. Furthermore, we hope to consider more expressive concept class families such as symbolic automata and context-free grammars: With symbolic automata we hope to support a mix of thresholded rewards and DFAs.

Chapter 6

Learning Affordances at Inference-Time for Vision-Language-Action Models

For the final contribution of this thesis, we will step away from formal task specifications, and take a look into robot foundation model (RFM)-based approaches to long-horizon policy learning. This chapter will explore how the general principles of the work we have discussed until now can be applied in open-world, unstructured environments. Specifically, we will show how a rigorous, structured approach to *constructing feedback* in general-purpose robotics can allow existing RFMs to eventually solve long-horizon tasks by iteratively leanring from previous failures.

Existing RFMs, typically based on powerful pre-trained vision-language model (VLM) backbones, have the potential to combine both the semantic and common-sense problem-solving abilities of LLMs and the flexible and dexterous end-to-end control capabilities of learned policies (Ahn et al. 2022; Brohan et al. 2023; Black et al. 2024; Kim et al. 2024; Octo Model Team et al. 2023). However, current robotic foundation models, most notably *Vision-Language-Action* models (VLAs), have primarily been studied in "single shot" settings, where they are evaluated on their ability to follow individual user commands. A practical robotic system needs to also plan through complex behaviors and, perhaps most importantly, adjust its behavior based on context and perceived capabilities. For example, if the robot needs to open a latched container, it might try to unlatch it in a particular way, and if that fails, it should modify its strategy and try a different approach. This kind of in-context adaptation has been observed as an *emergent* behavior in LLMs (Madaan et al. 2023; Gou et al. 2023; Shinn et al. 2023), but has proven difficult to enable in the robotics domain with current VLAs.

In this chapter, we propose an inference-time learning method that enables robotic policies to reason through complex tasks, form plans, and then adjust those plans based on observed outcomes (without any additional training). Our method, **Learning from Inference-Time Execution (LITEN)**, employs an off-the-shelf VLM to first attempt a

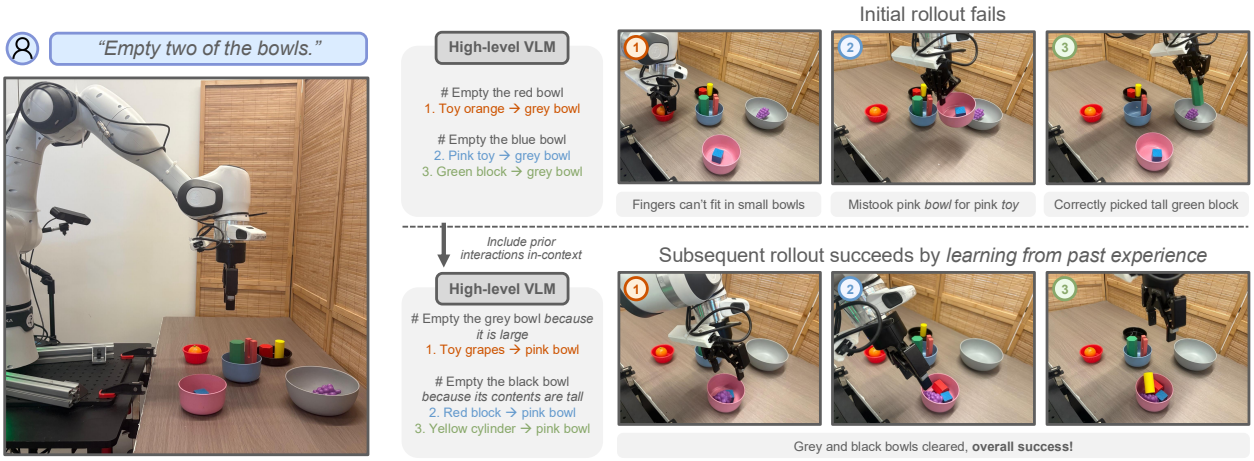


Figure 6.1: Our method, *Learning from Inference Time Execution* (LITEN), is a novel approach that leverages a two-level hierarchical model to reason about complex tasks based on prior experience collected at inference-time, allowing it to learn the robot's affordances and gradually improve performance.

task by controlling a low-level VLA, then retrospectively reason about the induced robot behaviors. These insights can then be included in-context in future attempts, allowing the VLM to progressively learns *affordances* (Gibson 2014): what the robot can or cannot do, subject to the constraints of dynamics, the robot embodiment, and the policy's learned behaviors. LITEN's high-level VLM "feels out" the policy's capabilities, gradually strengthening its interface with the low-level controller and *improving its high-level task reasoning as experience is accumulated*. We illustrate how LITEN can utilize past experiences in Figure 6.1.

Our approach can be seen as an adaptation of inference-time self-refinement approaches for agentic foundation models (Shinn et al. 2023; Tong et al. 2024; An et al. 2023) to real-world robotics. As shown in Figure 6.2, LITEN adopts a two-phase iterative approach to self-refinement: a *reasoning* phase, followed by an *assessment* phase. In the initial reasoning phase, our high-level VLM has no prior experience of how the VLA will act when instructed, and must "guess" a plan (i.e., a sequence of subtask instructions). After the plan is attempted in the physical world, we move to the assessment phase and enlist a VLM *judge* to evaluate the result of the VLA's execution. Unlike existing self-refinement methods that can leverage precise feedback in simulated or computational environments, we must deal with unstructured data (e.g., raw videos) and draw meaningful conclusions for use in refining robot task reasonings.

To gain valuable takeaways from unstructured videos of robot executions, LITEN breaks the process of assessment down into a sequence of steps that are individually manageable by the VLM judge. Specifically, the judge assesses the outcomes of robot rollouts by determining which subtask commands failed, what the policy did incorrectly, possible reasons why, and

how these errors might be avoided. These insights are then provided in-context when the VLM next attempts the task, allowing it to revise its task reasoning in light of its prior experience.

In summary, our contribution is LITEN, a novel inference-time technique for high-level VLMs to solve complex robotic tasks by learning affordances through repeated interactions with the environment, which is leveraged for better task reasoning. Crucially, LITEN requires no additional training and can be used by off-the-shelf VLMs. We demonstrate the efficacy of LITEN on a set of long-horizon manipulation tasks using the DROID Franka robot setup (Khazatsky et al. 2024). Our results show that our approach is able to iteratively improve its planning capabilities from repeated attempts at a task, outperforming baseline approaches to inference-time learning that are not designed for real-world robotics.

6.1 Related Work

Vision-Language-Action models. Vision-Language Models (VLMs) are a powerful source of representations for control. They are popularly leveraged via behavioral cloning (BC), which yields Vision-Language-Action policies (VLAs): VLMs that have been fine-tuned on vast robot demonstration corpora (O'Neill et al. 2024; Walke et al. 2023; Khazatsky et al. 2024; Jiang et al. 2025) into robot policies (Brohan et al. 2023; Kim et al. 2024; Black et al. 2024; Bjorck et al. 2025). VLAs are often generalists: due to their diverse training data, they can perform many tasks in unstructured settings, typically as commanded in text. Despite this, current VLAs are mainly trained on low-level tasks, and thus fail at high-level tasks that demand reasoning or common sense. While hierarchical (Physical Intelligence et al. 2025; Shi et al. 2025) and reasoning (Gemini Robotics Team et al. 2025; Zawalski et al. 2024; Chen et al. 2025; Lin et al. 2025) approaches alleviate this, such methods require expensive data collection, annotation, and training. In contrast, LITEN uses an off-the-shelf VLM to learn from inference-time rollouts by storing and reasoning about them in-context, without training.

Planning and affordance reasoning. An alternative approach that addresses this shortcoming is to use pre-trained foundation models as high-level planners or reasoners, typically interfacing with a separate low-level policy or controller (Huang et al. 2022; Zeng et al. 2022). These methods are *zero-shot*: they forgo robot-specific training and instead use the common-sense knowledge and reasoning capabilities in VLMs to determine (1) what the robot *should* do (e.g., by decomposing a goal into subtasks) and (2) what the robot *can* do, grounded in the scene and the robot's capabilities (also called "affordances" (Gibson 2014)). This dichotomy is outlined by SayCan (Ichter et al. 2022), which uses a language model to determine appropriate subtasks, while a separate learned value function estimates affordances. Other works learn a value function estimation for foundation-model based approaches either in an offline reinforcement learning setting (Nakamoto et al. 2024) or through visual heuristics (Ma et al. 2025). In contrast, VoxPoser (Huang et al. 2023) leverages (V)LMs to determine affordances by constructing a semantic voxel map in code. Other

approaches couple affordance estimation and task reasoning more tightly, using the VLM both to interpret the scene and decide on appropriate behaviors (Liang et al. 2023; Fang et al. 2024).

However, contrasting LITEN, the zero-shot nature of these approaches can also be a drawback: because the high-level VLM is not trained on robot data, it has limited understanding of the robot's capabilities and limitations, leading to suboptimal performance (Ichter et al. 2022). This is only exacerbated as considered tasks become more complex, necessitating both more flexible low-level controllers and the VLM to have a more fine-grained grasp of possible behaviors.

Inference-time learning for agentic tasks. Outside of robotics, iterative inference-time refinement methods have been explored for agentic tasks in other domains (Shinn et al. 2023; Tong et al. 2024; An et al. 2023). Such works often have similar considerations as in robotics, especially when applied to control in simulated environments (Du et al. 2023; Wang et al. 2023; Bhat et al. 2024). However, translating these methods directly to real-world robot control can be difficult. Specifically, existing works (1) abstract away low-level control by delegating it to a powerful planner (often leveraging privileged simulation data), and (2) makes use of that same ground-truth simulator state for expressing agent observations, skills, and even interaction feedback in text. Both these assumptions cannot be made for real-world robotics. We compare LITEN with a direct adaptation of Reflexion (Shinn et al. 2023) to illustrate our method's comparative efficacy in unstructured real-world robot manipulation applications.

6.2 Problem Statement

We consider the regime of robotic control for long-horizon instruction following. We define a *task* ℓ as a natural language description of the desired instruction, e.g., "Empty two of the bowls" from Figure 6.1.

In order to accomplish free-form natural language instructions like this, we make use of language-conditioned policies $\pi^{\text{low}}(a \mid o, \ell')$, which map observations o and input instruction text ℓ' to a distribution over low-level robot actions a. We specifically use *vision-language-action* models (VLAs) as π^{low}, as they are an effective end-to-end approach for learning generalist language-conditioned control policies (Brohan et al. 2023; Black et al. 2024; Kim et al. 2024).

Critically, while VLAs are open-vocabulary policies and can condition on arbitrary language, they often struggle when given open-ended, long-horizon instructions. Instead, they excel at the shorter-horizon, atomic behaviors found in their training data. To accomplish long-horizon tasks, they also need separate high-level policies to break the overall task into behaviors that the VLA is trained for (Physical Intelligence et al. 2025; Shi et al. 2025). To address this limitation, we use a highly-capable off-the-shelf VLM for high-level reasoning, π^{high}, decomposing the overall task ℓ into a sequence of subtask instructions $\ell'_0 \dots \ell'_n$. The VLA can then execute each subtask in sequence. When rolled out via a standard

action-perception loop, this yields a subtask trajectory $\tau_i = \{(o_0, a_0)_i, (o_1, a_1)_i, \dots (o_T, a_T)_i\}$ (where actions are sampled from the VLA conditioned on the current observation and subtask language $a_t \sim \pi^{\text{low}}(\cdot \mid o_t, \ell_i')$ and T is the length of the subtask trajectory). The overall trajectory is simply the concatenation of all these subtask trajectories τ_i in sequential order. The overall task ℓ is successful if it is accomplished by the end of the last subtask trajectory.

Finally, we note that the VLA has learned to map commands to a diverse range of low-level actions. While we have a sense of what language the VLA has been trained to follow (which is useful for giving examples of "reasonable" subtask language ℓ_i' for π^{high} to output), we do not have any a priori description of its affordances (e.g., in what scenarios it is able to accomplish certain commands, and with what likelihood). We thus consider an *iterative multi-round setting*, wherein the system can attempt the task multiple times. Critically, the system can pass forward some information to subsequent attempts. Our experiments thus compare different ways to use pre-trained VLMs as π^{high}, specifically showing that our approach, LITEN, is an effective way to infer affordances and improve performance from past robot rollouts.

6.3 Learning From Inference-Time Execution

We now present LITEN, an approach that allows the VLM planner π^{high} to learn the affordances of the VLA π^{low} at inference time through repeated interactions with the physical world, subsequently using these affordances to improve its planning capabilities. We show that, by reasoning about prior attempts at a task – how they succeed or fail, the underlying semantic or physical constraints that lead to these outcomes, and what sorts of behaviors the policy is empirically suited to – LITEN is an effective way to gradually improve task success rates as more experiences are accumulated in-context, all *with no extra policy training*.

LITEN consists of an iterative loop with two phases: a *reasoning* phase and an *assessment* phase. In the reasoning phase (Section 6.3), the VLM *reasoner* is given a task instruction ℓ and an initial image observation of the environment. It is instructed to reason through and decompose this goal into subtasks, which the VLA then executes as described in Section **??**. We then move to the assessment phase (Section 6.3), where we evaluate the outcome of each subtask execution in a structured format, invoking a VLM *judge* with a chain of prompts to draw meaningful conclusions from unstructured trajectories. Finally, the result of this assessment is then added back in-context to the prompt for the VLM reasoner in the next iteration, which generates a new plan that takes the prior experience into account (Section 6.3). We provide a diagrammatic representation of LITEN in Figure 6.2. We now detail each phase of LITEN.

Reasoning Phase: Generating and executing plans

The reasoning phase begins by asking a VLM to act as π^{high}. We give it the overall task ℓ and initial observation image, then prompt it to produce a list of subtasks that solve this

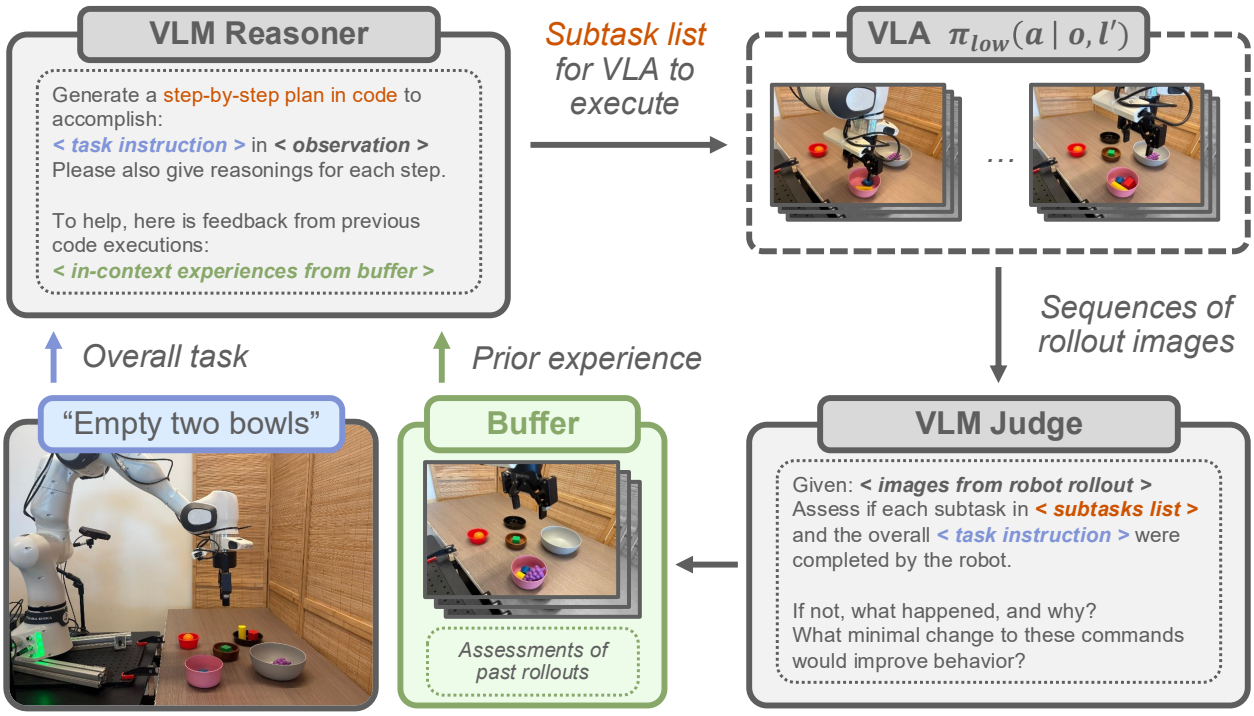


Figure 6.2: An overview of our approach. LITEN cycles between **(1) the reasoning phase**, wherein the VLM reasons about the task to decompose the task into subtasks that the VLA low-level policy can roll out in sequence and **(2) the assessment phase**, wherein the VLM judges the outcomes of inference-time rollouts. The reasoning phase includes the outputs of the assessment phase by including them in the VLM's context, allowing the reasoner to iteratively learn robot affordances and improve performance.

task, along with a justification for each step.

To help direct π^{high}'s outputs, we supply the VLM reasoner with additional context. First, we give it provide a description of language commands that are representative of instructions in the VLA's pre-training data. This constrains π^{high}'s outputs to the "style" of language subtasks that π^{low} can follow (though critically *not* its affordances, i.e., when each command is feasible). We also describe the general class of instructions that are relevant to our task setup: in our setting, these are pick-and-place and move operations by a robot arm. Lastly, to ground the reasoner in its environment, we employ a similar technique to prior work (Chen et al. 2022; Smith et al. 2025) and provide it with a list of manipulable objects, identified by a separate VLM call. Executing each subtask ℓ'_i in sequence with the VLA yields a corresponding trajectory segment $\tau_i = \{(o_0, a_0)_i, (o_1, a_1)_i, ...\}$.

Assessment Phase: Learning from previous executions

Once a plan has been fully executed, we begin the assessment phase by collecting the sequence of sub-trajectories for each subtask attempted by π^{low}, i.e., $\{(\ell'_0, \tau_0), \dots (\ell'_n, \tau_n)\}$. To generate useful feedback for future iterations, we present a *structured assessment procedure* that asks a VLM judge to answer a chain of prompts about each subtask that mirrors human reasoning. The prompts are as follows:

1. *Did it succeed?* We give a subtask instruction ℓ'_i and the first and last observation images of τ_i as input to the judge, and ask whether or not the robot successfully accomplished ℓ'_i.

2. *What happened instead?* In the case of failure, we ask the judge to describe what the robot did in the environment instead of ℓ'_i. In this step, we experimented with providing a video (sampled at various frame rates) of τ_i but found that our VLMs struggled to accurately comprehend unstructured sub-trajectory videos. For our experiments, we found it sufficient to provide the first and last observation images when asking the judge to describe what happened.

3. *Reason about failure.* We give the intended subtask ℓ'_i, the description of what actually happened from the previous step, and the initial observation image to the judge and ask it to reason about why π^{low} failed in the way it did. To ensure that the judge can correctly diagnose failure causes, we include substantial context in our prompt that outlines a general VLA training and inference procedure and briefly describes the way π^{low} attends to language (e.g., 'Our VLA tends to pay attention to the color, shape, and spatial orientation of objects in the instruction.') We additionally ask the VLM to suggest what *minimal* changes to either ℓ'_i or the environment would improve the chance of success. We emphasize minimality to encourage suggested changes that are grounded in the specific task context.

We provide an example illustrating the above steps in Figure 6.3. In the case of a successful subtask, we instead ask a VLM to briefly describe the environment in which the success occurred. Once we have generated assessments for each subtask, we then ask a VLM to evaluate the overall task. We provide the structured assessments for all subtasks in an execution, along with overall task language ℓ, and ask the judge to describe why the overall task succeeded or failed based on the outcomes of each subtask.

Using feedback in future iterations

Upon the start of the next reasoning phase, the structured feedback generated in all previous assessment phases is added in-context to the plan generation prompt. To ensure that assessments are used effectively, we add instructions on how to use prior experience to the VLM reasoner's prompt. In the prompt, we emphasize that a subtask's success is dependent on the environment, and that subtasks with multiple occurrences in-context do not necessarily

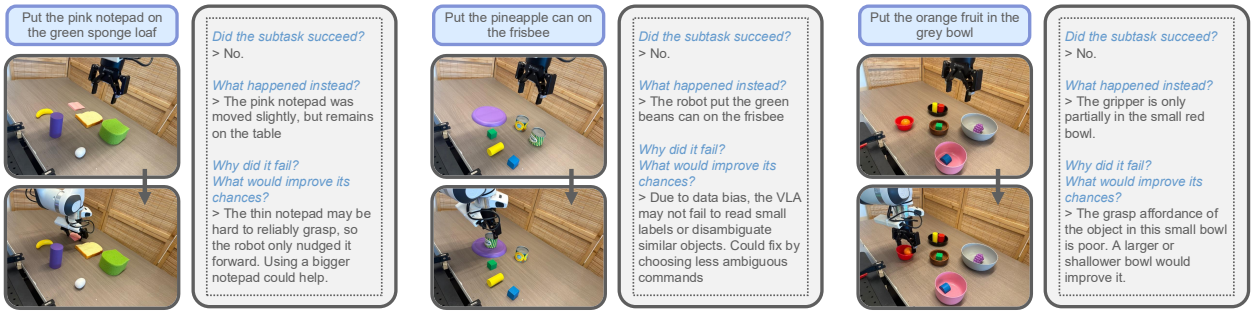


Figure 6.3: Examples of VLM judge's prompt and output for three subtasks (abridged for length). These generations are included in-context for the VLM reasoner in subsequent iterations attempting the task. Additional full examples are available in our codebase at https://github.com/ameesh-shah/liten-vla

share the same environment configuration with one another nor the current environment. Our prompt also notes that the VLA is stochastic, and its performance on the same subtask may vary across executions. We instruct the VLM reasoner to prioritize using subtask instructions in the following order: (1) instructions that are very likely to succeed in the current environment based on prior experience, (2) instructions that have not yet been tried, and (3) instructions that are unlikely to succeed but have been rephrased in a way that maximizes their success likelihood. The VLM reasoner takes into account its prior experience and the aforementioned instructions, then generates a new plan for execution.

Implementation

We now describe our concrete instantiation of LITEN on a real-world robotics setup. We use GPT-5-mini (OpenAI et al. 2025) as our high-level VLM in both the reasoning and assessment phases, given its cost effectiveness and state-of-the-art reasoning capabilities. For our low-level policy, we use $\pi_{0.5}$-DROID (Physical Intelligence et al. 2025), an open-source state-of-the-art VLA that has been fine-tuned on the DROID dataset (Khazatsky et al. 2024). Accordingly, our experiments use the standard DROID robot setup: a tabletop with a 7-DoF Franka Emika Panda robot arm and a 2F-85 Robotiq gripper end-effector operating at 5Hz.

The reasoning phase takes as input a user-specified ℓ, and an initial image of the scene that comes from a fixed ZED 2.0 camera on the right side of the tabletop. The reasoning phase is a single VLM request that includes the following in addition to the aforementioned inputs: the top-level plan generation prompt for ℓ, the in-context experience collected from prior attempts, and the instructions on how to use past experience. The VLM generates Python code with comments explaining its reasoning to serve as the plan and uses a provided API endpoint that allows it to invoke the VLA for a subtask ℓ'_i. Additionally, we ask the VLM

reasoner to justify its subtask generations in natural language.

During execution, $\pi_{0.5}$-DROID receives joint angle proprioception and RGB images from the same right-side ZED camera and a ZED mini wrist camera. Subtasks are run with a maximum horizon of 300 time steps. $\pi_{0.5}$-DROID produces joint velocity action chunks (Zhao et al. 2023) of length 8, so the VLA only runs inference every 8 steps. The robot's position is reset to home between subtasks.

In the assessment phase, the initial and final image observations from each subtask are provided to the chain of prompts in our structured assessment template. Each step outlined in Section 6.3 is a separate VLM request that uses the output from the previous steps. The results of all subtask assessments are then provided in a single request to the VLM judge when assessing the overall task. The overall task and subtask assessments, including the images used in those assessments, are then stored and provided in a hierarchical structure (subtasks are coupled with their overall task) in-context to future reasoning phases. Our implementation of LITEN, including the full prompts used, is available in our codebase.

6.4 Experimental Results

We evaluate LITEN on a collection of challenging multi-step tasks that requires the high-level VLM to learn relevant affordances, including (1) what the low-level robot controller is capable of and (2) spatial and physical properties of the environment and embodiment. Through our experiments, we seek to answer the following questions:

RQ.1. Can LITEN learn task-relevant affordances in its environment by interacting with the physical world?

RQ.2. As it gains experience, does LITEN effectively learn from both successes and failures, and iteratively improve its plans for complex tasks?

RQ.3. What is the importance of each step in the assessment phase of LITEN in facilitating learning from past experiences?

Experimental Setup

We select three complex, multi-stage tasks that are achievable by composing behaviors and instructions akin to those found in the DROID dataset. The initial configuration of our tasks do not vary across iterations in a given trial, but does vary slightly across trials; e.g., some object initial positions are switched. Our tasks are depicted in Figure 6.5 and described as follows:

1. **Stacking.** The robot must stack a total of three objects atop one another. The scene has 3 small blocks, 2 medium-sized cans, and a large upside down plate. The high-level VLM must learn which objects can be easily stacked atop one another by the VLA and which objects are too difficult to precisely balance.

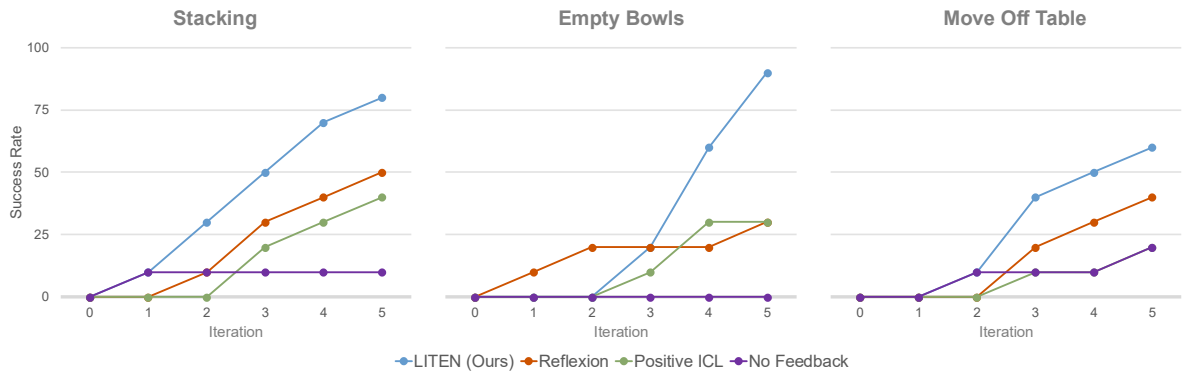


Figure 6.4: Success rates for the full completion of each multi-stage task over five attempt iterations. Our results show that LITEN is able to make effective use of prior attempts and consistently improves its plans as more experience is collected. Results are averaged over 10 trials.

2. **Emptying Bowls.** A number of bowls of different depths and sizes are set on the table. The robot must empty two bowls by moving their contents to other bowls. The high-level VLM must learn which bowls are either large enough for the gripper end-effector to fit in *or* shallow enough to allow objects to protrude and be grasped by the arm.

3. **Moving Off Table.** The robot must move objects on the table onto other objects such that only three objects remain in direct contact with the table. The high-level VLM must learn which objects are manipulable by the VLA, and which objects can serve as landing spots for other objects without rolling or falling off.

However, we found that using $\pi_{0.5}$-DROID without any additional training in our setup led to prohibitively poor performance. Specifically, we observed that $\pi_{0.5}$-DROID struggled to execute fine-grained operations required for pick-and-place instructions in our setting, such as properly orienting the gripper with the correct depth and angle to grasp small objects while avoiding others.

We further improved $\pi_{0.5}$-DROID's capabilities so that it could feasibly achieve the long-horizon tasks in our experiments by fine-tuning $\pi_{0.5}$-DROID on a small set of sub-task demonstrations collected in our tabletop setting. These demonstrations were collected through teleoperation in a manner identical to how demonstrations were collected in the original DROID dataset (Khazatsky et al. 2024). For each experimental task, we collected 150 demonstrations. We did not collect demonstrations on task instructions that were either physically infeasible or extremely difficult, e.g., balancing a rectangular block on a cylindrical block. We also varied the initial layout of the tabletop for each collected demonstration to prevent

overfitting (note that initial layouts from our demonstration set often differed significantly from the layout used during evaluation.)

We provide the specific task instructions collected along with the number of demonstrations collected for each instruction below:

```
% Stacking
Put the green block on top of the blue block:  15
Put the blue block on top of the green block:  15
Put the green beans can on top of the purple plate:  20
Put the green beans can on top of the yellow pineapple slices can:  15
Put the yellow pineapple slices can on top of the green beans can:  15
Put the yellow pineapple slices can on top of the purple plate:  15
Put the yellow cylinder block on top of the green beans can:  20
Put the yellow cylinder block on top of the yellow pineapple slices can:  20
Put the yellow cylinder block on top of the purple plate:  15
```

```
% Empty Bowls
Move the green block to the pink bowl:  15
Move the green block to the black bowl:  15
Move the orange fruit to the pink bowl:  15
Move the orange fruit to the gray bowl:  15
Move the yellow cylinder block to the gray bowl:  15
Move the yellow cylinder block to the pink bowl:  15
Move the red hexagonal block to the gray bowl:  15
Move the red hexagonal block to the pink bowl:  15
Move the purple grapes to the pink bowl:  15
Move the purple grapes to the gray bowl:  15
```

```
% Move Off Table
Put the purple cylinder on the green sponge:  30
Put the purple cylinder on the foam bread slice:  15
Put the white egg on the green sponge:  15
Put the white egg on the foam bread slice:  30
Put the white egg on the pink notepad:  10
Put the yellow banana on the foam bread slice:  15
Put the yellow banana on the pink notepad:  30
Put the pink notepad on the foam bread slice:  5
```

We fine-tuned $\pi_{0.5}$-DROID on each collected demonstration set separately for each experiment. We fine-tuned for 2500 training steps, using a batch size of 128 and an action chunk size of 16. The learning rate used began at 3e-5 and decayed on a cosine schedule every 250 steps with a decay rate of 2e-6. We performed training on a set of four NVIDIA A100 GPUs.

Baselines. We compare our approach against a number of alternative approaches to

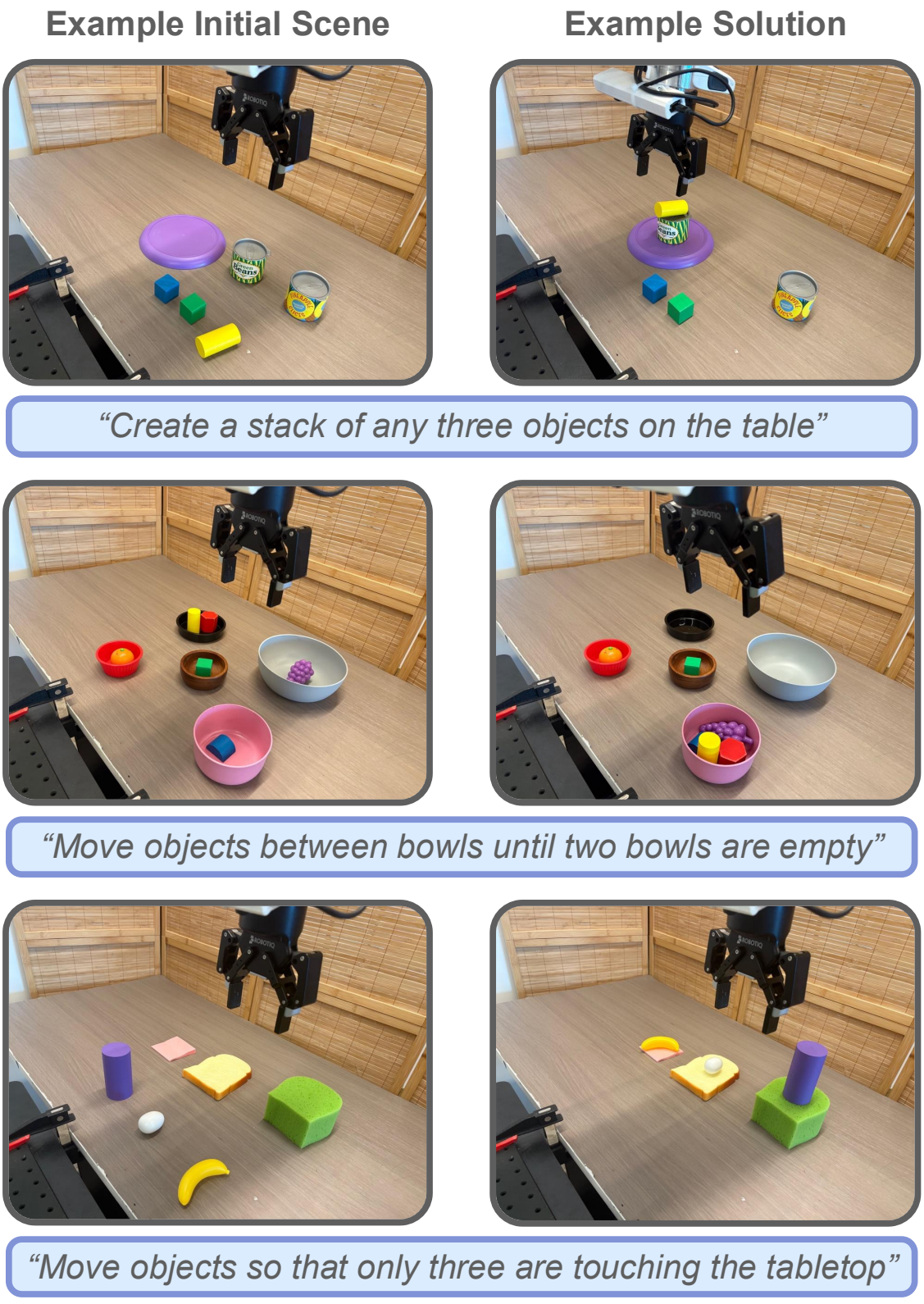


Figure 6.5: Examples of initial configurations and solutions for our tasks.

learning from inference-time experience that do not reason about experiences in the same way as LITEN. While to our knowledge prior VLA methods do not utilize inference-time experience in-context, we adapt *Reflexion* (Shinn et al. 2023), an inference-time learning technique for agentic LLMs, to a robotic setting. Our adaptation 'reflects' on unstructured videos of executed plans, and uses these reflections in-context for subsequent iterations. This baseline can be thought of as a naïve adaptation of standard self-refinement approaches to real-world robotics. We also compare against an approach that only stores successful subtask attempts and provides those successful attempts in-context for future plan generations, called *Positive-ICL*. This approach mirrors existing methods that use positive in-context examples for zero-shot open-world robotic manipulation (Liang et al. 2023), and can be viewed as an ablation of our method that does not use negative examples. Lastly, we compare against a naïve baseline that regenerates a new plan at every iteration without using any feedback, called *No-Feedback*.

Ablations. We answer **RQ.3.** by conducting an ablation study where we remove steps from the assessment phase (see Section 6.3) and run LITEN in the same manner as described above for the Empty Bowls task. We ablate the approach in two ways: (1) removing the failure reasoning (**w/o failure reasoning**: "why did the policy fail at this subtask?") and (2) removing the outcome analysis as well (**w/o failure reasoning and outcome**: "what did the policy do instead?"), leaving only whether the subtasks were successful or not.

Results

Main results. In Figure 6.4, we show the results of each approach in accomplishing the tasks over a total of five iterations, averaged over ten trials. For each trials, we run each method for a maximum of five iterations, stopping early if the attempt is successful. The success rates represent the rate of accomplishing the entire task and do not consider partial credit. Our results show that LITEN is able to make use of its collected experience and successfully instruct π^{low} with high-affordance subtask instructions to accomplish the overall tasks, answering **RQ.1.** and **RQ.2.** affirmatively. LITEN's success rate increases consistently over consecutive attempts, indicating that it effectively uses additional prior experience to further improve its plans. The baseline approaches are substantially less effective at leveraging prior experience. Notably, the naïve No-Feedback baseline is practically unable to accomplish any tasks, showing that the VLM will not occasionally generate a correct plan as a result of random chance. This shows the importance of learning from past experience at inference time.

Ablation results. We present the performance of LITEN under various ablation conditions in Figure 6.6, demonstrating the importance of the knowledge gained in each step of the assessment phase. Providing only the success or failure of each subtask (without failure reasoning or outcome analysis) yields the worst performance. Removing the failure reasoning only performs better, but will on occasion attribute differences between intended and actual outcomes to the instruction's language as opposed to physical explanations. The explicit "reason about failure" step provides useful context about why failures may have occurred,

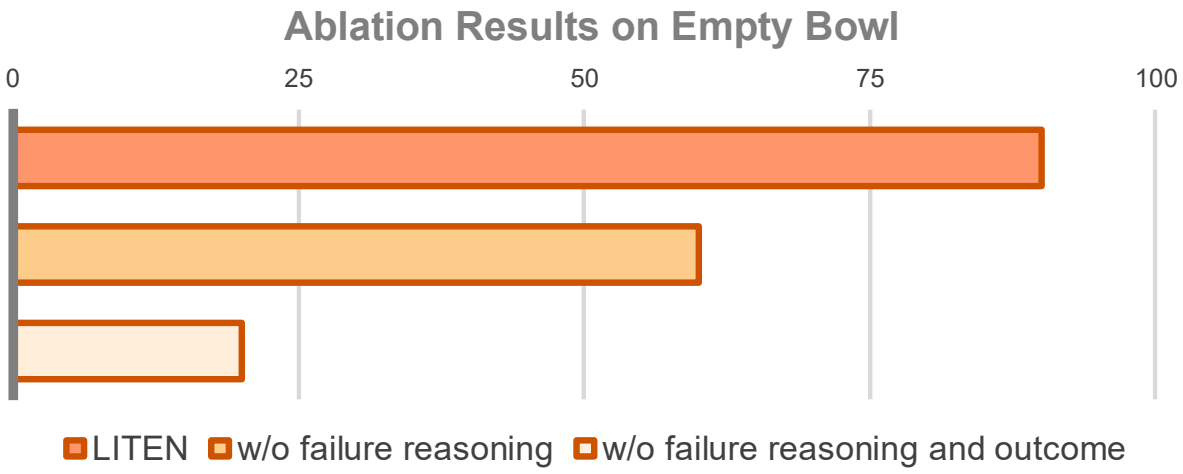


Figure 6.6: Success rates after five past attempts after ablating various components of LITEN's VLM judge. When any part of the assessment phase is ablated, performance drops dramatically, indicating that prior experience is most impactful when it includes whether the past rollout succeeded or not, what the policy did instead if not, and why that might be the case.

such as suggesting the limitations of VLA capabilities or physical properties of objects, and thus allows LITEN to learn more sophisticated affordances.

Qualitative analysis. We provide a qualitative analysis to answer **RQ.1.** and describe the affordances LITEN learns in our experiments. We find that LITEN is particularly successful at learning from failures that either indicate (1) biases from the VLA or (2) physical properties that cause obvious difficulties in control.

To exemplify (1), in the Stacking task, the VLA is biased towards manipulating larger objects in the scene, such as the two cans or the yellow cylinder block. In an attempt from our experiments, the high-level VLM instructed the VLA to "put the green cube on the purple plate", but the VLA instead moved the green beans can onto the purple plate. LITEN was able to clearly identify this mistake made by the VLA and recognize that moving the green can to the purple plate is a subtask that can be part of a potential solution. The subsequent plan included "put the green beans can on to the purple plate".

Regarding (2), we found that the high-level VLM often suggested initial plans that require precise control. For the Stacking task, the high-level VLM would frequently generate a plan involving stacking the three small blocks instead of the larger cans and plate, which is comparatively more difficult and requires finer manipulation skills. In the Moving Off Table task, the VLM occasionally suggested moving the green sponge on top of other objects, despite it being the largest object on the table and too wide for the gripper. However, LITEN enabled the high-level VLM to draw meaningful conclusions from these attempts

(e.g., "the VLA may lack precise top-down placement abilities when stacking the blue and green blocks" or "the gripper could not grasp the green sponge due to size constraints") and generate plans that adjusted accordingly.

The baselines are unable to learn as effectively. Because the Positive-ICL baseline only retains successful subtasks in-context, the method must effectively "get lucky" and find successful subtasks through the randomness of the VLM-generated plans, leading to high inefficiency. This illustrates how LITEN's fine-grained rollout assessments critically provide a strong signal for understanding the robot's capabilities and leveraging it for improvement. The Reflexion baseline reasons over entire raw video trajectories, as opposed to our structured reasoning approach, which we found often led to poor comprehension of physical outcomes. For example, we found that entire-video reflections would often hallucinate objects that were not in the scene, or mistakenly assume subtasks were successfully completed. This underscores the challenge of extracting meaningful feedback from raw robot interaction data.

6.5 Discussion

Multi-round Analysis

In this subsection, we provide examples of multiple attempt iterations generated by LITEN on each of our experimental tasks. The examples, provided in Figures 6.7, 6.8 and 6.9, illustrate the evolution of sub-task sequences generated by the VLM reasoner, along with feedback from the VLM judge produced during the assessment phase. We provide the initial and final images of the environment for each sub-task's execution that were given as input to the VLM judge during the assessment phase. For conciseness, we abridge the output from the VLM judge and include the most relevant text in our illustrations. The overall task-level reasoning performed after each attempt was also elided for brevity. Each of the provided examples ultimately culminates in a successful attempt by the VLM reasoner.

We identify additional observations from our evaluation of LITEN, highlighted by the provided examples. When asked to reason about the failure of a given sub-task, the VLM judge generates multiple possible hypotheses, which typically span across (1) potential inaccuracies in the sub-task's language instruction, (2) potential visual distractors or difficulties, (3) control errors either attributed to the VLA's capabilities or the dynamics of the environment, and (4) a lack of relevant data in the VLA's training or fine-tuning datasets. In our experiments, categories (2) and (3) are often the most valuable for the reasoner when generating the next sub-task sequence. For example, in the Stacking task example from Figure 6.8, the VLA fails to "put the pineapples can on the green beans can" in Attempt 2. Although the VLM judge suggests placing a marker on top of the green beans can to improve the chance of success, which is infeasible in our experimental setup, the VLM reasoner is able to utilize other parts of the judge's feedback, such as the difficulty of stacking two cans, and instead opts to stack the smaller, easier to place yellow cylinder in the third attempt.

Category (1), where the VLM judge suggests modifications to the sub-task's language

instruction, were generally not useful in our experiments due to the language following capabilities of the fine-tuned $\pi_{0.5}$ VLA. Prior work has shown that current VLAs struggle to attend to fine-grained semantics in an instruction's language (Glossop et al. 2025), and we observed this to be the case in our evaluations as well. As VLAs continue to improve at language following, category (1) will become increasingly valuable for inference-time adaptation. In a similar vein, category (4) did not provide valuable feedback in the context of our experiments, given that our work studies a purely in-context learning setting.

As discussed in section 6.3, LITEN's VLM judge assesses individual sub-tasks based on the first and last observation images of an execution, rather than (raw or downsampled) videos which led to poor performance. However, obvious limitations remain when using static images to assess a dynamic execution. Following the same example from the Stacking task in Figure 6.8, the second sub-task in Attempt 2 of stacking the pineapples can on the green beans can resulted in the robot placing the pineapple can partially on top of the green beans can, which led to the pineapple can falling back on to the table. The VLM judge, only able to see the before and after images, could only draw conclusions from seeing the pineapple can displaced from its original position, which led to less precise feedback than what otherwise would have been provided from an accurate interpretation of the video. As VLMs improve their ability to comprehend cause and effects in unstructured videos, LITEN will improve correspondingly.

Failure Cases

We characterize the failure cases of LITEN when it fails to correctly plan for a task after multiple execution attempts. The majority of failures can be attributed to either (a) the stochasticity of the VLA, (b) LITEN's tendency to attribute control failures to the wording of a subtask, or (c) LITEN's inability to reason causally about subtask orderings.

For the first failure case, a small number of potential subtasks in our experiments have high variance, such as stacking one can atop another in the Stacking task. If LITEN experiences a successful subtask execution in an early iteration, it will tend to fixate on using this subtask, even if multiple subsequent executions of the same subtask fail.

In the second case, if a subtask execution fails due to a control error, LITEN may misdiagnose this failure as a result of imprecise language in the subtask instruction. For example, if the subtask "put the blue block on the green block" fails, but the blue block gets moved close to the green block, LITEN may attribute this failure to the language and provide an instruction like "put the blue block directly on top of the center of the green block."

The third case is most easily understood via example. In the Move Off Table task, overall success is highly sensitive to the ordering and control precision of each subtask. In one instance, the purple cylinder was successfully placed on the green sponge. However, the next subtask was placed the white egg also on the green sponge, which, while successful, knocked the purple cylinder back onto the table in the process. Although the VLM judge managed to accurately describe this phenomenon during the assessment phase ("The VLA

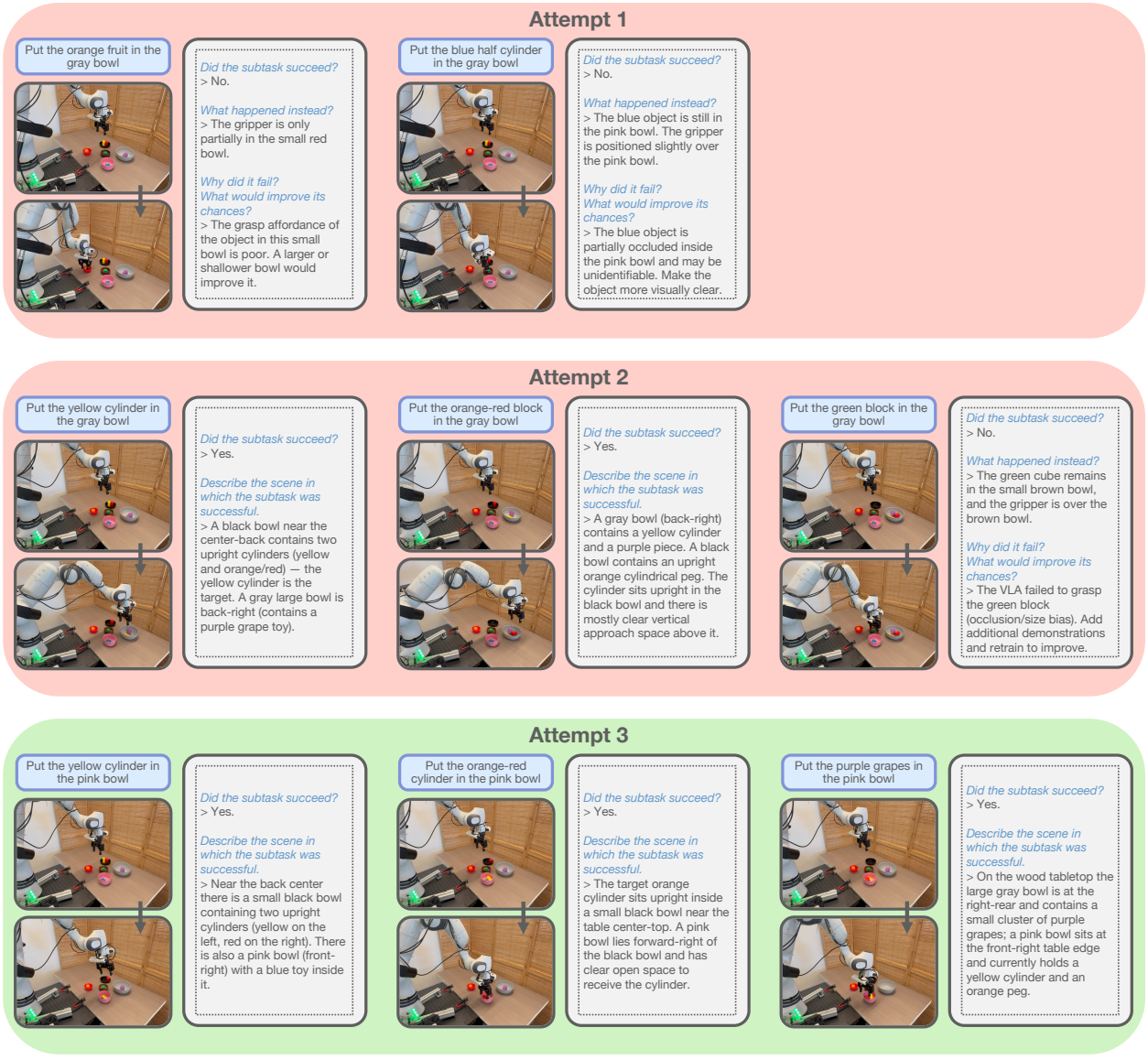


Figure 6.7: An example sequence of attempts for LITEN on the Empty Bowls task that resulted in a successful execution. VLM outputs from the reasoner and judge are condensed for length and readability.

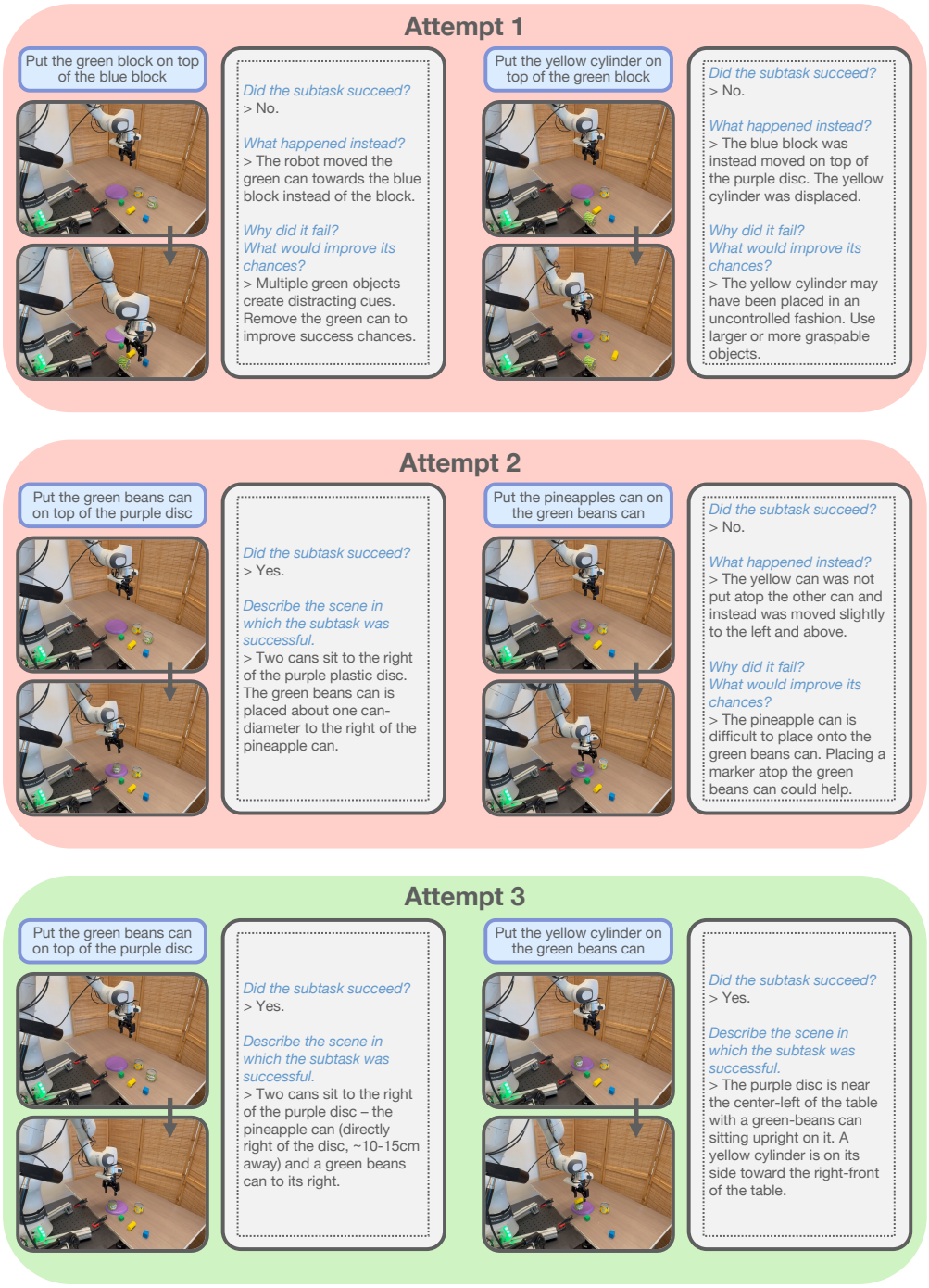


Figure 6.8: An example sequence of attempts for LITEN on the Stacking task from our experiments that resulted in a successful execution. VLM outputs from the reasoner and judge are condensed for length and readability.

Figure 6.9: An example sequence of attempts for LITEN on the Empty Bowls task from our experiments that resulted in a successful execution. VLM outputs from the reasoner and judge are condensed for length and readability.

likely ... displaced the cylinder when placing the egg"), the VLM reasoner struggled to make use of this insight in future plans.

Analysis and Limitations

We now take a more prescriptive view of the cases discussed earlier. Both the first and second failure cases represent failure of the VLM to adapt to VLA affordances on *individual* subtasks. In the first case, the VLM reasoner needs to strike a balance between avoiding "borderline affordances" of VLAs, while attempting to leverage the VLA's steerability through language at the same time.

The second failure case is an artifact of using language as the reasoning mode. We expect that as VLM video comprehension capabilities improve, the VLM judge can properly classify infeasible actions that no amount of subtask language modification will fix, mitigating this failure mode.

The last case represents what we see as the most salient current limitation of LITEN. Unlike the first two failure cases, it exhibits a sequence of successful individual subtasks but a failure at the overall task level. This indicates a deficiency in the ability of the VLM to reason across subtasks. Thus, instilling the high-level VLM reasoner with stronger logical and causal reasoning is a clear area of future improvement for inference-time affordance learning.

6.6 Conclusion

In this chapter, we presented LITEN, a novel inference-time learning method that enables generalist robot policies to reason about complex tasks, attempt them in the real world, and then distill insights from those past attempts to improve future reasoning, without requiring additional training. LITEN iterates through a two-phase process that first generates plans for a VLA using a high-level VLM, then assesses the execution of that plan through a chain of prompts that enables the VLM to draw meaningful conclusions for use in subsequent planning iterations. Our experimental results show that LITEN effectively uses past experience to solve complex manipulation tasks that require an understanding of the robot's affordances.

LITEN joins an emerging body of work on in-context learning for robotics (Vosylius et al. 2024; Fu et al. 2024). We see the broad applicability of LITEN as one of its key strengths: in addition to not requiring training, LITEN can be used with any off-the-shelf VLM and VLA. Further, our approach is hardware-agnostic, only requiring changes to our prompts for adaptation to new robot setups. In a similar vein, we expect LITEN to become increasingly useful as the capabilities of (robotic) foundation models improve: if VLMs can better understand the physical world and VLAs can more accurately attend to language instructions, LITEN will empower the combination of these stronger models to solve highly complex tasks.

There are many exciting directions for future work. Although LITEN is only demonstrated on single tasks, its usage in a *lifelong learning* setting is compelling, where past experience across many different tasks is used to solve new tasks. To that end, we are interested in understanding what past experiences are more or less useful in efficiently gaining affordances. We are also interested in scaling up LITEN to settings with large amounts of prior data that cannot entirely be fit in-context, which will require adapting context management techniques such as RAG (Lewis et al. 2020).

Chapter 7

Final Words

Before machine learning existed as a tractable approach to any problem, classical robotics automated physical tasks by decoupling their systems into *planning* and *control* (Murray et al. 2017). The motivation for this split was to separate the disparate methodologies between low-level manipulation or dexterity, often solved through manually designed physical controllers, and high-level task reasoning, solved through symbolic or algorithmic search. As learning-based approaches to robotics gained momentum, a growing body of researchers and practitioners sought to unify planning and control with learned policies that operated "end-to-end", taking raw sensory input and directly producing low-level actuation as output.

This thesis acknowledges the well-documented difficulty in end-to-end robot learning, particularly for long-horizon tasks (Shah et al. 2025b; Gao et al. 2025). The body of work that this thesis comprises joins a renaissance of planning-and-control-like approaches in robot learning, with a particular eye towards formal logic as a means of performing high-level planning and reasoning (Shah et al. 2025a). In doing so, this thesis aims to highlight what state-of-the-art robot learning is capable of, what remains a challenge, and how we can bridge these gaps in the short-to-medium term. In this conclusion, we will review the narrative threads from previous chapters, and lay out a vision for the future of robot learning where highly capable learned policies are able to *plan for themselves*, attempt those plans, and then learn and refine their own behavior as they dynamically interact with the real world.

First, let us review the challenges outlined in our introduction, and the approaches we develop to address them:

1. **Making incremental progress**: A critical shortcoming of modern policy learning techniques is that long-horizon tasks are typically learned as monoliths. For example, in a supervised learning setting, a practitioner may collect a demonstration for a long-horizon task, label it with a task specification, and then use it to train a model as-is. At most, modern-day practitioners will only label semantically clear sub-tasks (Shi et al. 2025), which is usually a coarse-grained decomposition of the long-horizon task at best. When we train a model to replicate this behavior, the result is brittle, myopic behavior that fails to generalize to novel settings. This issue is exacerbated in reinforcement

learning, where sparse reward feedback prevents learning any meaningful behavior altogether. In chapter 3, we address this problem by representing long-horizon tasks as symbolic 'plans' that can be structurally exploited to reward incremental progress through a task (Shah et al. 2025d). Crucially, our approach allows for a policy to learn different approaches to solving a problem, rather than unimodal demonstrations that fail to capture a diversity of acceptable behavior.

2. **Balancing completion with cost:** Existing reward design methodologies in reinforcement learning often ambiguously combine task-critical feedback with cost conditions, such as efficiency, safety, or fluidity of motion. Chapter 3 proposes a methodology that disambiguates correctness and cost conditions by representing task specifications with *formal logic*, and requires optimal policies to achieve the task with maximum probability before allowing it to optimize its behavior with respect to cost. This constrained optimization-based approach to correctness-critical reinforcement learning offers an approach that guarantees theoretical soundness: any policy that prioritizes cost-optimality over maximum correctness will not receive higher reward than any policy that maximizes correctness.

3. **Coordinating with other agents:** When robot policies must coordinate and cooperate to solve sufficiently complex tasks, standard approaches to multi-agent learning suffer from the *credit assignment problem* (Agogino et al. 2004b). To incentivize individual members of a team to contribute towards an optimally efficient strategy, we introduce LOTaD (Shah et al. 2025c) in chapter 4, a methodology that formally decomposes symbolic team task representations into individual sub-tasks. In doing so, we are able to greatly accelerate multi-agent reinforcement learning and find far more efficient cooperative strategies than existing approaches.

4. **Autonomously recognizing and learning from past mistakes:** In the real world, robots are going to make mistakes when attempting tasks. When these mistakes occur, our policies must be able to dynamically adapt to unexpected outcomes and replan through complex behaviors based on past experience and inferred context. Existing foundation-model based approaches to robotics, such as Vision-Language-Action models, are single-shot: if a policy fails to successfully achieve an instructed task, it has no recourse to learn from that past attempt and try again in a different way. To reconcile this, we present LITEN (Shah et al. 2025b) in chapter 6, a framework for VLA self-refinement that reasons about how to solve tasks by observing prior attempts and using that past experience to inform plans for accomplishing future long-horizon tasks.

Future Directions

Although end-to-end approaches to robot learning have significant shortcomings in reasoning and planning, the goal of using fully learned policies offers immense promise. When we create learning-based solutions that contain symbolic or algorithmic components (often referred

to as *Neurosymbolic* Machine Learning), we are limiting the generalizable capabilities of our learned policies in exchange for higher certainty in its operations. In the short term, neurosymbolic approaches like those introduced in this thesis are necessary to bridge the gaps of current end-to-end policy learning approaches and bring about reliable robotic real-world systems. However, as the field of learning rapidly progresses and foundation models become more and more integrated into our technology, we must as a community begin to think about how logic and symbolic methods can become *tools* for autonomous agents, as opposed to current neurosymbolic methods that force learned components to abide by symbolic rules or traverse symbolic plans. Chain-of-thought reasoning (Wei et al. 2022) has spurred an immense area of research on autonomous *reasoning* for foundation models. Although the current usage of the term 'reasoning' in foundation model research typically refers to the vague concept of using additional inference tokens before producing a final response to a prompt, symbolic tools and formal methods can make reasoning rigorous by requiring additional inference tokens to be used towards satisfying a formal logic specification. To improve assurance, consistency, and reliability of modern AI, the same symbolic structures and formalisms we exploit in this thesis must become second-nature languages to learned policies.

So how do we teach foundation models to use formal logic and create plans for themselves that abide by formal guarantees of correctness and safety? For starters, the neurosymbolic robotics community should start by taking a data-centric approach to 'logic-as-tool-use', and develop large-scale datasets that pair embodied behavior with representative task specifications. In doing so, we can enable embodied chain-of-thought reasoning (Zawalski et al. 2024) that produces *formal artifacts* alongside actionable behavior. Such plans can be refined through iterative improvement, similar to our approach in LITEN, and we can leverage the internet-scale prior of foundation models to facilitate this self-refinement.

In the future, we anticipate that enforcing robot foundation models to strictly adhere to the formal plans they develop for themselves will be highly difficult. Formal verification in robotics and cyber-physical systems lags behind their software-only counterparts. Due to the complexity of the real world, it is possible that we are never able to achieve full formal guarantees on learned embodied behavior. However, practitioners who work on safe robot learning can leverage policies that generate their own plans to perhaps 'constrain' those policies to specific plans by simplifying the policy space and leveraging tools like simulation to ensure that a policy's planned behavior will not strongly deviate from expectation. Achieving assured, open-world robotics is a grand challenge, but this thesis aims to take a step in that direction and open new avenues of thought for the robotics, machine learning, and cyber-physical communities.

Bibliography

Abbeel, Pieter et al. (2004). "Apprenticeship learning via inverse reinforcement learning". In: *ICML*. Vol. 69. ACM International Conference Proceeding Series. ACM.

Abel, David et al. (2021). "On the Expressivity of Markov Reward". In: *Advances in Neural Information Processing Systems*. URL: https://proceedings.neurips.cc/paper/2021/file/4079016d940210b4ae9ae7d41c4a2065-Paper.pdf.

Achiam, Joshua et al. (2017). "Constrained policy optimization". In: *International conference on machine learning*. PMLR, pp. 22–31.

Agogino, Adrian K et al. (2004a). "Unifying temporal and structural credit assignment problems". In: *Autonomous agents and multi-agent systems conference*.

— (2004b). "Unifying Temporal and Structural Credit Assignment Problems". In: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*. IEEE Computer Society, pp. 980–987. DOI: 10.1109/AAMAS.2004.10098. URL: https://doi.ieeecomputersociety.org/10.1109/AAMAS.2004.10098.

Ahn, Michael et al. (2022). "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691*.

Alexandre Duret-Lutz et al. (Aug. 2022). "From Spot 2.0 to Spot 2.10: What's New?" In: *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*. Vol. 13372. Lecture Notes in Computer Science. Springer, pp. 174–187. DOI: 10.1007/978-3-031-13188-2_9.

Alur, Rajeev et al. (2022). "A Framework for Transforming Specifications in Reinforcement Learning". In: *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Ed. by Jean-François Raskin et al. Vol. 13660. Lecture Notes in Computer Science. Springer, pp. 604–624. DOI: 10.1007/978-3-031-22337-2_29. URL: https://doi.org/10.1007/978-3-031-22337-2%5C_29.

An, Shengnan et al. (2023). "Learning from mistakes makes LLM better reasoner". In: *arXiv preprint arXiv:2310.20689*.

Angluin, Dana (1987). "Learning Regular Sets from Queries and Counterexamples". In: *Inf. Comput.* 75.2, pp. 87–106.

Audibert, Jean-Yves et al. (2009). "Exploration–exploitation tradeoff using variance estimates in multi-armed bandits". In: *Theoretical Computer Science* 410.19, pp. 1876–1902.

Auer, Peter (2002). "Using Confidence Bounds for Exploitation-Exploration Trade-offs". In: *J. Mach. Learn. Res.* 3, pp. 397–422. URL: http://jmlr.org/papers/v3/auer02a.html.

Auer, Peter et al. (2002a). "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47, pp. 235–256.

Auer, Peter et al. (2002b). "The Nonstochastic Multiarmed Bandit Problem". In: *SIAM J. Comput.* 32.1, pp. 48–77.

Baert, Mattijs et al. (2024). "Learning Temporal Task Specifications From Demonstrations". In: *Explainable and Transparent AI and Multi-Agent Systems - 6th International Workshop, EXTRAAMAS 2024, Auckland, New Zealand, May 6-10, 2024, Revised Selected Papers*. Ed. by Davide Calvaresi et al. Vol. 14847. Lecture Notes in Computer Science. Springer, pp. 81–98. DOI: 10.1007/978-3-031-70074-3_5. URL: https://doi.org/10.1007/978-3-031-70074-3%5C_5.

Baier, Christel et al. (2008). *Principles of model checking*. en. Cambridge, Mass: The MIT Press. ISBN: 978-0-262-02649-9.

Bakker, Vincent de et al. (2025). "Scaffolding Dexterous Manipulation with Vision-Language Models". In: *arXiv preprint arXiv:2506.19212*.

Balakrishnan, Anand et al. (2019). "Structured Reward Shaping using Signal Temporal Logic specifications". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, pp. 3481–3486. DOI: 10.1109/IROS40897.2019.8968254. URL: https://doi.org/10.1109/IROS40897.2019.8968254.

Bastani, Osbert et al. (2018). "Active learning of points-to specifications". In: *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. Ed. by Jeffrey S. Foster et al. ACM, pp. 678–692. DOI: 10.1145/3192366.3192383. URL: https://doi.org/10.1145/3192366.3192383.

Basu, Chandrayee et al. (2019). "Active Learning of Reward Dynamics from Hierarchical Queries". In: *IROS*, pp. 120–127. DOI: 10.1109/IROS40897.2019.8968522.

Bernstein, Daniel S et al. (2002). "The complexity of decentralized control of Markov decision processes". In: *Mathematics of operations research* 27.4, pp. 819–840.

Bharadhwaj, Homanga et al. (2023). "Zero-shot robot manipulation from passive human videos". In: *arXiv preprint arXiv:2302.02011*.

Bhat, Vineet et al. (2024). "Grounding llms for robot task planning using closed-loop state feedback". In: *arXiv preprint arXiv:2402.08546*.

Biyik, Erdem et al. (2018). *Batch Active Preference-Based Learning of Reward Functions*. arXiv: 1810.04303 [cs.LG].

Bjorck, Johan et al. (2025). "Gr00t n1: An open foundation model for generalist humanoid robots". In: *arXiv preprint arXiv:2503.14734*.

Black, Kevin et al. (2024). "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164*.

Bongard, Josh C. et al. (2005). "Active Coevolutionary Learning of Deterministic Finite Automata". In: *J. Mach. Learn. Res.* 6, pp. 1651–1678. URL: http://jmlr.org/papers/v6/bongard05a.html.

Bozkurt, Alper Kamil et al. (2020). "Control synthesis from linear temporal logic specifications using model-free reinforcement learning". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10349–10355.

Brohan, Anthony et al. (2022). "RT-1: Robotics Transformer for Real-World Control at Scale". In: *arXiv preprint arXiv:2212.06817*.

Brohan, Anthony et al. (2023). "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *arXiv preprint arXiv:2307.15818*.

Brown, Daniel S. et al. (2018). "Risk-Aware Active Inverse Reinforcement Learning". In: *CoRL*. Vol. 87. Proceedings of Machine Learning Research. PMLR, pp. 362–372.

Burton, Nichola et al. (Apr. 2021). "Beyond Likert ratings: Improving the robustness of developmental research measurement using best–worst scaling". In: *Behavior Research Methods*. ISSN: 1554-3528. DOI: 10.3758/s13428-021-01566-w. URL: https://doi.org/10.3758/s13428-021-01566-w.

Camacho, Alberto et al. (July 2019). "LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, pp. 6065–6073. DOI: 10.24963/ijcai.2019/840. URL: https://doi.org/10.24963/ijcai.2019/840.

Carroll, Micah et al. (2019). "On the Utility of Learning about Humans for Human-AI Coordination". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al., pp. 5175–5186. URL: https://proceedings.neurips.cc/paper/2019/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

Chen, Boyuan et al. (2022). "Open-vocabulary queryable scene representations for real world planning". In: *arXiv preprint arXiv:2209.09874*.

Chen, William et al. (2025). *Training Strategies for Efficient Embodied Reasoning*. arXiv: 2505.08243 [cs.RO].

Chou, Glen et al. (2020). "Explaining Multi-stage Tasks by Learning Temporal Logic Formulas from Suboptimal Demonstrations". In: *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020*. Ed. by Marc Toussaint et al. DOI: 10.15607/RSS.2020.XVI.097. URL: https://doi.org/10.15607/RSS.2020.XVI.097.

De Giacomo, Giuseppe et al. (Sept. 2020). "Temporal Logic Monitoring Rewards via Transducers". In: *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 860–870. DOI: 10.24963/kr.2020/89. URL: https://doi.org/10.24963/kr.2020/89.

De la Higuera, Colin (2010). *Grammatical inference: learning automata and grammars*. Cambridge University Press.

Du, Yuqing et al. (2023). *Guiding Pretraining in Reinforcement Learning with Large Language Models*. arXiv: 2302.06692 [cs.LG].

Fainekos, Georgios E. et al. (2009). "Robustness of temporal logic specifications for continuous-time signals". In: *Theor. Comput. Sci.* 410.42, pp. 4262–4291. DOI: 10.1016/J.TCS.2009.06.021. URL: https://doi.org/10.1016/j.tcs.2009.06.021.

Fang, Kuan et al. (2024). "MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting". In: *Robotics: Science and Systems (RSS)*.

Feng, Lei et al. (2022). "Multi-level credit assignment for cooperative multi-agent reinforcement learning". In: *Applied Sciences* 12.14, p. 6938.

Ferret, Johan et al. (2019). "Self-attentional credit assignment for transfer in reinforcement learning". In: *arXiv preprint arXiv:1907.08027*.

Fu, Jie et al. (2014). "Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints". In: *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*. Ed. by Dieter Fox et al. DOI: 10.15607/RSS.2014.X.039. URL: http://www.roboticsproceedings.org/rss10/p39.html.

Fu, Letian et al. (2024). "In-context imitation learning via next-token prediction". In: *arXiv preprint arXiv:2408.15980*.

Gao, Jensen et al. (2025). "A Taxonomy for Evaluating Generalist Robot Policies". In: *CoRR* abs/2503.01238. DOI: 10.48550/ARXIV.2503.01238. arXiv: 2503.01238. URL: https://doi.org/10.48550/arXiv.2503.01238.

Gemini Robotics Team et al. (2025). *Gemini Robotics: Bringing AI into the Physical World*. arXiv: 2503.20020 [cs.RO].

Gibson, James J (2014). *The Ecological Approach to Visual Perception: Classic Edition*. Psychology Press.

Glossop, Catherine et al. (2025). "CAST: Counterfactual Labels Improve Instruction Following in Vision-Language-Action Models". In: *arXiv preprint arXiv:2508.13446*.

Gou, Zhibin et al. (2023). "Critic: Large language models can self-correct with tool-interactive critiquing". In: *arXiv preprint arXiv:2305.11738*.

Gundana, David et al. (2021). "Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks". In: *IEEE Robotics Autom. Lett.* 6.2, pp. 3687–3694. DOI: 10.1109/LRA.2021.3064220. URL: https://doi.org/10.1109/LRA.2021.3064220.

Haarnoja, Tuomas et al. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

Hadfield-Menell, Dylan et al. (2016). "Cooperative Inverse Reinforcement Learning". In: *NeurIPS*, pp. 3909–3917.

Hahn, Ernst Moritz et al. (2013). "Lazy probabilistic model checking without determinisation". In: *arXiv preprint arXiv:1311.2928*.

Hasanbeig, Mohammadhosein et al. (2018). *Logically-Constrained Reinforcement Learning*. DOI: 10.48550/ARXIV.1801.08099. URL: https://arxiv.org/abs/1801.08099.

Hasanbeig, Mohammadhosein et al. (2020). "Deep reinforcement learning with temporal logics". In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, pp. 1–22.

Hasselt, Hado van et al. (2021). "Expected eligibility traces". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 11, pp. 9997–10005.

Heule, Marijn et al. (2010). "Exact DFA Identification Using SAT Solvers". In: *ICGI*.

Holladay, Rachel et al. (June 2016). "Active Comparison Based Learning Incorporating User Uncertainty and Noise". In: *Proceedings of RSS Workshop on Model Learning for Human-Robot Communication*.

Hsiung, Eric et al. (2023). "Learning Reward Machines through Preference Queries over Sequences". In: *CoRR* abs/2308.09301. DOI: 10.48550/ARXIV.2308.09301. arXiv: 2308.09301. URL: https://doi.org/10.48550/arXiv.2308.09301.

Hu, Yujing et al. (2020). "Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/b710915795b9e9c02cf10d6d2bdb688c-Abstract.html.

Huang, Wenlong et al. (2022). "Inner Monologue: Embodied Reasoning through Planning with Language Models". In: *arXiv preprint arXiv:2207.05608*.

Huang, Wenlong et al. (2023). "Voxposer: Composable 3D Value Maps for Robotic Manipulation with Language Models". In: *arXiv preprint arXiv:2307.05973*.

Hurley, Isabelle et al. (2024). "STL: Still Tricky Logic (for System Validation, Even When Showing Your Work)". In: *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. Ed. by Amir Globersons et al.

Icarte, Rodrigo Toro et al. (2022). "Reward machines: Exploiting reward function structure in reinforcement learning". In: *Journal of Artificial Intelligence Research* 73, pp. 173–208.

Ichter, Brian et al. (2022). "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*. Ed. by Karen Liu et al. Vol. 205. Proceedings of Machine Learning Research. PMLR, pp. 287–318. URL: https://proceedings.mlr.press/v205/ichter23a.html.

Ji, Jiaming et al. (2023). "Safety-Gymnasium: A Unified Safe Reinforcement Learning Benchmark". In: *arXiv preprint arXiv:2310.12567*.

Jiang, Zhenyu et al. (2025). "Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning". In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 16923–16930.

Jothimurugan, Kishor et al. (2019). "A composable specification language for reinforcement learning tasks". In: *Advances in Neural Information Processing Systems* 32.

Karimadini, Mohammad et al. (Jan. 2011). "Cooperative Tasking for Deterministic Specification Automata". In: *Asian Journal of Control* 18. DOI: 10.1002/asjc.1300.

Katehakis, Michael N et al. (1987). "The multi-armed bandit problem: decomposition and computation". In: *Mathematics of Operations Research* 12.2, pp. 262–268.

Kearns, Michael J. et al. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

Khazatsky, Alexander et al. (2024). "DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset". In: *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*. Ed. by Dana Kulic et al. DOI: 10.15607/RSS.2024.XX.120. URL: https://doi.org/10.15607/RSS.2024.XX.120.

Kim, Moo Jin et al. (2024). "OpenVLA: An Open-Source Vision-Language-Action Model". In: *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*. Ed. by Pulkit Agrawal et al. Vol. 270. Proceedings of Machine Learning Research. PMLR, pp. 2679–2713. URL: https://proceedings.mlr.press/v270/kim25c.html.

Krasowski, Hanna et al. (2023). "Safe Reinforcement Learning with Probabilistic Guarantees Satisfying Temporal Logic Specifications in Continuous Action Spaces". In: *62nd IEEE Conference on Decision and Control, CDC 2023, Singapore, December 13-15, 2023*. IEEE, pp. 4372–4378. DOI: 10.1109/CDC49753.2023.10383601. URL: https://doi.org/10.1109/CDC49753.2023.10383601.

Křetínskỳ, Jan et al. (2018). "Owl: a library for ω-words, automata, and LTL". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer, pp. 543–550.

Lauffer, Niklas et al. (2022). "Learning Deterministic Finite Automata Decompositions from Examples and Demonstrations". In: *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*. Ed. by Alberto Griggio et al. IEEE, pp. 1–6. DOI: 10.34727/2022/ISBN.978-3-85448-053-2_39. URL: https://doi.org/10.34727/2022/isbn.978-3-85448-053-2%5C_39.

Lavaei, Abolfazl et al. (2020). "Formal Controller Synthesis for Continuous-Space MDPs via Model-Free Reinforcement Learning". In: *11th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2020, Sydney, Australia, April 21-25, 2020*. IEEE, pp. 98–107. DOI: 10.1109/ICCPS48487.2020.00017. URL: https://doi.org/10.1109/ICCPS48487.2020.00017.

Le, Hoang et al. (2019). "Batch policy learning under constraints". In: *International Conference on Machine Learning*. PMLR, pp. 3703–3712.

Lewis, Patrick et al. (2020). "Retrieval-augmented generation for knowledge-intensive NLP tasks". In: *Advances in neural information processing systems* 33, pp. 9459–9474.

Li, Andrew C. et al. (2024). "Reward Machines for Deep RL in Noisy and Uncertain Environments". In: *CoRR* abs/2406.00120. DOI: 10.48550/ARXIV.2406.00120. arXiv: 2406.00120. URL: https://doi.org/10.48550/arXiv.2406.00120.

Li, Xiao et al. (2017). "Reinforcement learning with temporal logic rewards". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3834–3839.

Liang, Jacky et al. (2023). "Code as Policies: Language Model Programs for Embodied Control". In: *IEEE International Conference on Robotics and Automation*. IEEE, pp. 9493–9500.

Lin, Fanqi et al. (2025). "OneTwoVLA: A Unified Vision-Language-Action Model with Adaptive Reasoning". In: *CoRR* abs/2505.11917. DOI: 10.48550/ARXIV.2505.11917. arXiv: 2505.11917. URL: https://doi.org/10.48550/arXiv.2505.11917.

Ma, Yecheng Jason et al. (2025). "Vision Language Models are In-Context Value Learners". In: *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. URL: https://openreview.net/forum?id=friHAl5ofG.

Madaan, Aman et al. (2023). "Self-refine: Iterative refinement with self-feedback". In: *Advances in Neural Information Processing Systems* 36, pp. 46534–46594.

Maler, Oded et al. (2004). "Monitoring Temporal Properties of Continuous Signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*. Ed. by Yassine Lakhnech et al. Vol. 3253. Lecture Notes in Computer Science. Springer, pp. 152–166. DOI: 10.1007/978-3-540-30206-3_12. URL: https://doi.org/10.1007/978-3-540-30206-3%5C_12.

Mealy, George H (1955). "A method for synthesizing sequential circuits". In: *The Bell System Technical Journal* 34.5, pp. 1045–1079.

Moeller, Mark et al. (2023). "Automata Learning with an Incomplete Teacher". In: *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*. Ed. by Karim Ali et al. Vol. 263. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 21:1–21:30. DOI: 10.4230/LIPICS.ECOOP.2023.21. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2023.21.

Murray, Richard M et al. (2017). *A mathematical introduction to robotic manipulation*. CRC press.

Muškardin, Edi et al. (Mar. 2022). "AALpy: an active automata learning library". In: *Innovations in Systems and Software Engineering* 18, pp. 1–10. DOI: 10.1007/s11334-022-00449-3.

Nakamoto, Mitsuhiko et al. (2024). "Steering your generalists: Improving robotic foundation models via value guidance". In: *arXiv preprint arXiv:2410.13816*.

Neary, Cyrus et al. (2021). "Reward Machines for Cooperative Multi-Agent Reinforcement Learning". In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. Ed. by Frank Dignum et al. ACM, pp. 934–942. DOI: 10.5555/3463952.3464063. URL: https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p934.pdf.

Ng, Andrew Y. et al. (2000). "Algorithms for Inverse Reinforcement Learning". In: *ICML*. Morgan Kaufmann, pp. 663–670.

Nguyen, Duc Thien et al. (2017). "Policy gradient with value function approximation for collective multiagent planning". In: *Advances in neural information processing systems* 30.

O'Neill, Abby et al. (2024). "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6892–6903.

Octo Model Team et al. (2023). *Octo: An Open-Source Generalist Robot Policy*. https://octo-models.github.io.

OpenAI et al. (2025). "GPT-5 System Card". In: *OpenAI Blog*.

Oroojlooy, Afshin et al. (2023). "A review of cooperative multi-agent deep reinforcement learning". In: *Appl. Intell.* 53.11, pp. 13677–13722. DOI: 10.1007/S10489-022-04105-Y. URL: https://doi.org/10.1007/s10489-022-04105-y.

Ouyang, Yutao et al. (2024). "Long-horizon locomotion and manipulation on a quadrupedal robot with large language models". In: *arXiv preprint arXiv:2404.05291*.

Palan, Malayandi et al. (2019). "Learning Reward Functions by Integrating Human Demonstrations and Preferences". In: *Robotics: Science and Systems*.

Pan, Jiayi et al. (2023). "Data-Efficient Learning of Natural Language to Linear Temporal Logic Translators for Robot Task Specification". In: *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, pp. 11554–11561. DOI: 10.1109/ICRA48891.2023.10161125. URL: https://doi.org/10.1109/ICRA48891.2023.10161125.

Phelps, Andrew S. et al. (Jan. 2015). "Pairwise comparison versus Likert scale for biomedical image assessment". In: *AJR. American journal of roentgenology* 204.1, pp. 8–14. ISSN: 1546-3141. DOI: 10.2214/AJR.14.13022.

Physical Intelligence et al. (2025). $\pi_{0.5}$*: a Vision-Language-Action Model with Open-World Generalization*. arXiv: 2504.16054 [cs.LG].

Pnueli, Amir (1977). "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee, pp. 46–57.

Qiu, Wenjie et al. (2023). "Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/7b35a69f434b5eb07ed1b1ef16ace52c-Abstract-Conference.html.

Radford, Alec et al. (2019). "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* 1.8, p. 9.

Radosavovic, Ilija et al. (2024). "Humanoid locomotion as next token prediction". In: *Advances in neural information processing systems* 37, pp. 79307–79324.

Rashid, Tabish et al. (2020). "Monotonic value function factorisation for deep multi-agent reinforcement learning". In: *Journal of Machine Learning Research* 21.178, pp. 1–51.

Riedmiller, Martin et al. (2018). "Learning by playing solving sparse reward tasks from scratch". In: *International conference on machine learning*. PMLR, pp. 4344–4353.

Rutherford, Alexander et al. (2023). "JaxMARL: Multi-Agent RL Environments in JAX". In: *arXiv preprint arXiv:2311.10090*.

Sadigh, Dorsa et al. (2014). "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications". In: *53rd IEEE Conference on Decision and Control*. IEEE, pp. 1091–1096.

Sadigh, Dorsa et al. (2017). "Active Preference-Based Learning of Reward Functions". In: *RSS*.

Schillinger, Philipp et al. (2016). "Decomposition of Finite LTL Specifications for Efficient Multi-agent Planning". In: *Distributed Autonomous Robotic Systems, The 13th International Symposium, DARS 2016, Natural History Museum, London, UK, November 7-9, 2016*. Ed. by Roderich Groß et al. Vol. 6. Springer Proceedings in Advanced Robotics. Springer, pp. 253–267. DOI: 10.1007/978-3-319-73008-0_18. URL: https://doi.org/10.1007/978-3-319-73008-0%5C_18.

— (2018). "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems". In: *Int. J. Robotics Res.* 37.7, pp. 818–838. DOI: 10.1177/0278364918774135. URL: https://doi.org/10.1177/0278364918774135.

Schulman, John et al. (2017). "Proximal Policy Optimization Algorithms". In: *CoRR*. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

Shah, Ameesh et al. (2025a). "Learning Formal Specifications from Membership and Preference Queries". In: *International Conference on Neuro-symbolic Systems, 28-30 May 2025, University of Pennsylvania, Philadelphia, Pennsylvania, USA*. Ed. by George J. Pappas et al. Vol. 288. Proceedings of Machine Learning Research. PMLR, pp. 365–383. URL: https://proceedings.mlr.press/v288/shah25a.html.

Shah, Ameesh et al. (2024). "Deep Policy Optimization with Temporal Logic Constraints". In: *CoRR* abs/2404.11578. DOI: 10.48550/ARXIV.2404.11578. arXiv: 2404.11578. URL: https://doi.org/10.48550/arXiv.2404.11578.

Shah, Ameesh et al. (2025b). "Learning Affordances at Inference-Time for Vision-Language-Action Models". In: *CoRR* abs/2510.19752. DOI: 10.48550/ARXIV.2510.19752. arXiv: 2510.19752. URL: https://doi.org/10.48550/arXiv.2510.19752.

Shah, Ameesh et al. (2025c). "Learning Symbolic Task Decompositions for Multi-Agent Teams". In: *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025, Detroit, MI, USA, May 19-23, 2025*. Ed. by Sanmay Das et al. International Foundation for Autonomous Agents and Multiagent Systems / ACM, pp. 1904–1913. DOI: 10.5555/3709347.3743827. URL: https://dl.acm.org/doi/10.5555/3709347.3743827.

Shah, Ameesh et al. (2025d). "LTL-Constrained Policy Optimization with Cycle Experience Replay". In: *Trans. Mach. Learn. Res.* 2025. URL: https://openreview.net/forum?id=gxUp2d4JTw.

Shi, Lucy Xiaoyang et al. (2025). "Hi robot: Open-ended instruction following with hierarchical vision-language-action models". In: *arXiv preprint arXiv:2502.19417*.

Shinn, Noah et al. (2023). "Reflexion: Language agents with verbal reinforcement learning". In: *Advances in Neural Information Processing Systems* 36, pp. 8634–8652.

Sickert, Salomon et al. (2016). "Limit-Deterministic Büchi Automata for Linear Temporal Logic". In: *Computer Aided Verification*. Ed. by Swarat Chaudhuri et al. Cham: Springer International Publishing, pp. 312–332. ISBN: 978-3-319-41540-6.

Smith, Laura et al. (2025). "Steer: Flexible robotic manipulation via dense language grounding". In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 16517–16524.

Smith, Sophia et al. (2023). "Automatic Decomposition of Reward Machines for Decentralized Multiagent Reinforcement Learning". In: *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 5423–5430.

Sun, Changyin et al. (2020). "Reinforcement learning with task decomposition for cooperative multiagent systems". In: *IEEE transactions on neural networks and learning systems* 32.5, pp. 2054–2065.

Sunehag, Peter et al. (2017). "Value-decomposition networks for cooperative multi-agent learning". In: *arXiv preprint arXiv:1706.05296*.

Sutton, Richard S. et al. (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.

Sutton, Richard Stuart (1984). *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst.

Sverdlik, William et al. (1992). "Dynamic Version Spaces in Machine Learning". In: *ICTAI*. IEEE CS, pp. 308–315.

Tomita, Masaru (1982). "Learning of construction of finite automata from examples using hill-climbing : RR: Regular set Recognizer". In.

Tong, Yongqi et al. (2024). "Can LLMs learn from previous mistakes? investigating LLMs' errors to boost for reasoning". In: *arXiv preprint arXiv:2403.20046*.

Toro Icarte, Rodrigo et al. (May 2022). "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning". In: *J. Artif. Int. Res.* 73. ISSN: 1076-9757. DOI: 10.1613/jair.1.12440. URL: https://doi.org/10.1613/jair.1.12440.

Ulyantsev, Vladimir et al. (2015). "BFS-Based Symmetry Breaking Predicates for DFA Identification". In: *LATA*. Vol. 8977. LNCS. Springer, pp. 611–622.

Vaezipoor, Pashootan et al. (2021). "LTL2Action: Generalizing LTL Instructions for Multi-Task RL". In: *Proceedings of the 38th International Conference on Machine Learning, ICML*. Vol. 139. Proceedings of Machine Learning Research, pp. 10497–10508. URL: http://proceedings.mlr.press/v139/vaezipoor21a.html.

Vazquez-Chanlatte, Marcell et al. (2018). *Learning Task Specifications from Demonstrations*. arXiv: 1710.03875 [cs.LG].

Voloshin, Cameron et al. (2022). "Policy Optimization with Linear Temporal Logic Constraints". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. URL: https://openreview.net/forum?id=yZcPRIZEwOG.

Voloshin, Cameron et al. (2023). "Eventual Discounting Temporal Logic Counterfactual Experience Replay". In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by Andreas Krause et al. Vol. 202. Proceedings

of Machine Learning Research. PMLR, pp. 35137–35150. URL: https://proceedings.mlr.press/v202/voloshin23a.html.

Vosylius, Vitalis et al. (2024). "Instant policy: In-context imitation learning via graph diffusion". In: *arXiv preprint arXiv:2411.12633*.

Walke, Homer Rich et al. (2023). "BridgeData V2: A Dataset for Robot Learning at Scale". In: *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*. Ed. by Jie Tan et al. Vol. 229. Proceedings of Machine Learning Research. PMLR, pp. 1723–1736. URL: https://proceedings.mlr.press/v229/walke23a.html.

Wang, Chuanzheng et al. (2020a). "Continuous motion planning with temporal logic specifications using deep neural networks". In: *arXiv preprint arXiv:2004.02610*.

Wang, Guanzhi et al. (2023). "Voyager: An Open-Ended Embodied Agent with Large Language Models". In: *arXiv preprint arXiv:2305.16291*.

Wang, Jianhao et al. (2021). "Towards understanding cooperative multi-agent q-learning with value factorization". In: *Advances in Neural Information Processing Systems* 34, pp. 29142–29155.

Wang, Tonghan et al. (2020b). "Rode: Learning roles to decompose multi-agent tasks". In: *arXiv preprint arXiv:2010.01523*.

Webster, Matt et al. (2020). "A corroborative approach to verification and validation of human-robot teams". In: *Int. J. Robotics Res.* 39.1. DOI: 10.1177/0278364919883338. URL: https://doi.org/10.1177/0278364919883338.

Wei, Jason et al. (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems* 35, pp. 24824–24837.

Wilson, Aaron et al. (2012). "A Bayesian Approach for Policy Learning from Trajectory Preference Queries". In: *NIPS*, pp. 1142–1150.

Xiao, Xuan et al. (2025). "Robot learning in the era of foundation models: A survey". In: *Neurocomputing*, p. 129963.

Xu, Yichong et al. (2017). *Noise-Tolerant Interactive Learning from Pairwise Comparisons*. arXiv: 1704.05820 [stat.ML].

Xu, Yichong et al. (2020a). *Thresholding Bandit Problem with Both Duels and Pulls*. arXiv: 1910.06368 [cs.LG].

Xu, Zhe et al. (2020b). "Joint Inference of Reward Machines and Policies for Reinforcement Learning". In: *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*. Ed. by J. Christopher Beck et al. AAAI Press, pp. 590–598. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6756.

Yalcinkaya, Beyazit et al. (n.d.). "Automata Conditioned Reinforcement Learning with Experience Replay". In: *NeurIPS 2023 Workshop on Goal-Conditioned Reinforcement Learning*.

Yang, Cambridge et al. (2022). "On the (In)Tractability of Reinforcement Learning for LTL Objectives". In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Ed. by Lud De Raedt. Main Track. International Joint Con-

ferences on Artificial Intelligence Org., pp. 3650–3658. DOI: 10.24963/ijcai.2022/507. URL: https://doi.org/10.24963/ijcai.2022/507.

Yifru, Lunet et al. (2023). "Joint Learning of Policy with Unknown Temporal Constraints for Safe Reinforcement Learning". In: *CoRR* abs/2305.00576. DOI: 10.48550/arXiv.2305.00576. arXiv: 2305.00576. URL: https://doi.org/10.48550/arXiv.2305.00576.

Zawalski, Michał et al. (2024). *Robotic Control via Embodied Chain-of-Thought Reasoning*. arXiv: 2407.08693 [cs.RO].

Zeng, Andy et al. (2022). *Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language*. arXiv: 2204.00598 [cs.CV].

Zhao, Tony Z. et al. (2023). "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware". In: *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*. Ed. by Kostas E. Bekris et al. DOI: 10.15607/RSS.2023.XIX.016. URL: https://doi.org/10.15607/RSS.2023.XIX.016.

Zhou, Ce et al. (2025). "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt". In: *International Journal of Machine Learning and Cybernetics* 16.12, pp. 9851–9915.

Zhou, Meng et al. (2020). "Learning implicit credit assignment for cooperative multi-agent reinforcement learning". In: *NeurIPS* 33, pp. 11853–11864.