ROBOTS THAT INTROSPECT: IMPROVING DEPLOYMENT-TIME PERFORMANCE FOR LONG-HORIZON PLANNING UNDER UNCERTAINTY

by

Abhishek Paudel
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____ Dr. Gregory J. Stein, Dissertation Director

_____ Dr. Xuesu Xiao, Committee Member

_____ Dr. Ziyu Yao, Committee Member

_____ Dr. Ali K. Raz, Committee Member

_____ Dr. David S. Rosenblum, Department Chair

Date: _____ Fall Semester 2025
George Mason University
Fairfax, VA

Robots that Introspect: Improving Deployment-Time Performance for Long-Horizon Planning under Uncertainty

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Abhishek Paudel
Master of Science
George Mason University, 2023
Bachelor of Engineering
Tribhuvan University, 2017

Director: Dr. Gregory J. Stein, Assistant Professor
Department of Computer Science

Fall Semester 2025
George Mason University
Fairfax, VA

Dedication

To my parents...

Acknowledgments

"The Road goes ever on and on,
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can."

— J.R.R. Tolkien, *The Lord of the Rings*

My graduate school journey has been long, winding, and transformative—marked by curiosity, challenge, growth, and quiet joy. What began as a distant dream in a country far from home gradually became a lived experience, filled with challenges, discoveries, and lessons that extend far beyond the pages that follow. Looking back, it is not only the research itself that stands out but also the people, moments, and memories that gave meaning to this journey.

First and foremost, I am deeply grateful to my advisor, Prof. Gregory J. Stein, for his unwavering guidance, support, and encouragement throughout my doctoral journey. When I began my PhD, I had little experience with the rigor of research, critical thinking, or scholarly writing. Under Greg's compassionate and nurturing mentorship, I learned not only how to structure my work and approach research thoughtfully but also how to think critically and communicate effectively. There were times when self-doubt or existential worries about my work weighed heavily on me, yet coming out of a meeting with Greg was always quite a magical experience—my worries would fade, and I would leave motivated, confident, and reassured that I was doing valuable work. His patience, insight, trust, and genuine care have shaped not only my research but also the person I have become as a researcher and as an individual. I will continue to be inspired by his dedication, curiosity, and integrity, which have always motivated me to strive for the same standards in my work and in my life.

I am equally grateful to my committee members, Prof. Xuesu Xiao, Prof. Ziyu Yao, and Prof. Ali K. Raz, whose thoughtful feedback and guidance have helped me refine my ideas and strengthen my research. They each have a genuine passion for research that has inspired me in unique ways. In addition to their guidance, I am also thankful for their time, patience, and encouragement during the later years of my PhD, which helped me broaden my perspective and grow as a researcher.

My labmates have been an integral part of this journey. Abhish, not only my labmate but also my dear *bhai*, has been my close companion both in and out of the lab. Together, we have laughed and cried, built and broken things, worked hand in hand—even quite literally with one of us on the keyboard and the other one on the mouse—and, most importantly, encouraged each other through challenges and successes alike, making both research and life immensely enjoyable. Having him beside me—whether we were deep in research, lost in some hobby project, watching movies, or simply chatting about life—gave me the motivation and

energy to embrace and persevere through each day of this journey. I feel truly fortunate and deeply grateful for his presence and support along the way. I am also thankful to Roshan, Raihan, Ridwan, Arnab, Yimeng, and Bui; their presence in the lab made it more lively, and together we created countless fun memories, including trips to ICRA and IROS conferences. The lab was more than just a workspace—it became a place of learning, laughter, and camaraderie, and I will always cherish the moments shared with these remarkable peers. I am equally grateful to Max, Benned, and Prof. George Konidaris at Brown University for the invaluable opportunities to collaborate, learn, and exchange ideas, which greatly enriched my research experience. I am also thankful to Naman, whose mentorship since our meeting at CoRL has helped me navigate the broader research and career landscape.

I am profoundly thankful to Sarala *didi* and her family, who welcomed me when I first came to the US—a country that felt completely foreign, where I knew no one and had little idea how to find my footing. Their warmth and generosity made me feel at home and supported during the early, uncertain phase of my journey. I am also grateful to my fellow Nepali friends at Mason, whose companionship and shared experiences made my time here joyful and memorable. Special thanks to Prabin and Anuj, who I have known for more than a decade now. As housemates, Prabin and I have shared countless wonderful meals, movies, and conversations; and with Anuj, the three of us have shared fond memories hanging out, traveling, and listening to music—*tum tak* plays in the background. I am also thankful to Sulabh, Angeela, Divesh, Daking, Preksha, Rajendra, Bikram, Gaurab, Denish, Amit, Sameep, Surendra, Sachita, Aakriti, and Manshi for their friendship, road trips, house parties, and late-night chats that ranged from utter nonsense to profoundly philosophical. While I cannot name everyone here, I hope you know that if you are reading this, you are in my thoughts and appreciation.

To my siblings and cousins—Archana, Aarati, Deeya, Pratistha, Jagannath, Sansrit, Subrat, and Ananya—thank you for being a constant presence in my life despite the distance. With our regular video chats, it never feels like I am far from you. Our conversations about life, movies, books, travel, philosophy, relationships, career, politics, and family gossip have brought me joy and comfort, making my PhD journey feel deeply connected to home and far less lonely. I am especially indebted to Jagannath, my *go-to guy* for anything and everything in life, who has stood by me intellectually and emotionally through both the highs and lows of my journey. I cannot imagine having navigated some of the toughest periods in my life without him by my side to lean on. I am also grateful to all my *joint family* members—*sandai*, *Rama auntie*, *thuldidi*, *sandidi*, *sanu uncle*, and *Pushpa auntie*—whose love, blessings, and good wishes have reached me from thousands of miles away.

Finally, I owe my deepest gratitude to my parents. My *mamu* and *baba* have worked tirelessly all their lives and have made innumerable sacrifices to give me the freedom to dream big and the opportunity to follow those dreams. I will eternally be indebted to them, and I owe all my accomplishments to their unconditional love, sacrifices, support, and trust. This dissertation is as much theirs as it is mine.

Funding Acknowledgment: This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 2232733.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	application programming interface
LLM	large language model
LSP	learning over subgoals planning
MAB	multi-armed bandit
MCTS	Monte Carlo tree search
MDP	Markov decision process
PDDL	planning domain definition language
POMDP	partially observable Markov decision process
RL	reinforcement learning
RRT	rapidly-exploring random tree
SLAM	simultaneous localization and mapping
UCB	upper confidence bound
VLM	vision language model

Abstract

ROBOTS THAT INTROSPECT: IMPROVING DEPLOYMENT-TIME PERFORMANCE FOR LONG-HORIZON PLANNING UNDER UNCERTAINTY

Abhishek Paudel

George Mason University, 2025

Dissertation Director: Dr. Gregory J. Stein

Reliably deploying autonomous robots for long-horizon tasks like cleaning, cooking or delivery remains a fundamental challenge due to the pervasive nature of uncertainty and incomplete knowledge in real-world environments. For robots to perform well in these scenarios, they must not only make effective decisions under uncertainty but also be capable of continuous self-assessment and improvement during deployment. In this dissertation, we propose introspection for robots as a key idea that enables a robot to reason about how well it is doing and whether it could have done better so as to improve its behavior during deployment for long-horizon planning problems.

We present novel techniques that leverage this introspection ability to address the challenges of data-efficiency when selecting the best policies during deployment in unknown environments. Specifically, we introduce offline alt-policy replay to reuse data collected during a trial to estimate the performance bounds of many alternative planning strategies simultaneously. We show that our introspection approach allows a robot to quickly select the best-performing policies from a family of pre-trained policies during deployment, resulting in improved performance in unknown environments for long-horizon goal-directed navigation tasks with guarantees on asymptotic sub-linear regret. In scenarios where the

robot's pre-trained policies may not perform well, and the robot must learn or adapt its policies during deployment for effective performance, we additionally leverage our introspection approach to reason about non-stationary policies that are being continually learned or adapted during deployment, enabling the selection of the best-performing ones as soon as their performance improves and resulting in improved behavior during deployment in previously unseen and potentially out-of-distribution environments. Finally, we demonstrate the utility of such introspection in a novel LLM-informed model-based planning framework for object search, where the robot introspects the performance of different prompts and LLMs used to guide its search behavior in partially known environments, enabling effective object search performance during deployment despite uncertainty.

Collectively, this dissertation advances the vision of introspective robots that are capable of monitoring, evaluating, and improving their own behavior during deployment. By enabling robots to reason about their past behaviors, anticipate alternative outcomes, and adapt to uncertainty, this work takes a step toward self-improving autonomy in complex, real-world environments.

Chapter 1: Introduction

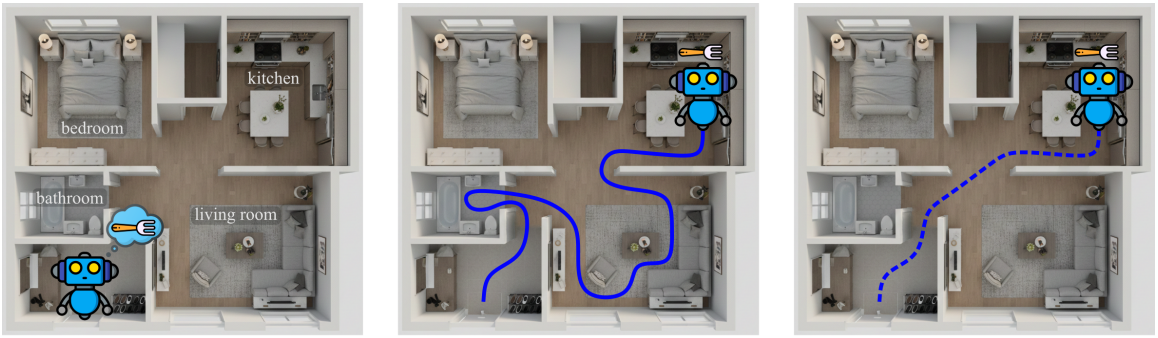
As we welcome the next generation of service robots into our homes and offices to assist us in a wide variety of tasks like cleaning, making breakfast, moving things around or other routine chores, they must not only be capable of performing these tasks effectively and reliably but also be able to learn and improve over time while continuously accessing how good or bad they are at doing these tasks. Humans demonstrate remarkable abilities to reason about how good we are at doing certain things and identify ways to improve our skills over time. We can *introspect* about our past behaviors and learn from past mistakes so as to get better in our abilities to do our tasks over time. Our ability to introspect allows us to reason about how well we did in the past and imagine what would have happened if we had done certain things differently. Such *introspection* allows us to course correct our behaviors and get better at doing things over time while enabling us to avoid things that might result in poor outcomes.

Robots, on the other hand, do not intrinsically possess such introspection abilities. While advances in machine learning have enabled robots to learn from prior experiences and get better over time, such robots when deployed in new scenarios that are drastically different from those seen before tend to perform poorly, as a result of distribution shift. Worse yet, these robots would have no way to identify their poor performance so as to either course correct their behavior or wait until their behavior has been improved in such scenarios so as to minimize failure cases or avoid poor outcomes. If we could imbue robots with abilities to introspect, it would be possible to reason about good and bad behaviors without the robot having go through them first-hand to realize their impacts.

The idea of introspection is not new in the robotics literature, and many approaches exist that try to imbue robots with abilities for introspection (Fox et al., 2006; Marshall et al., 2008; Liang et al., 2024). Adjacent to introspection are the approaches for *runtime*

monitoring that aim to evaluate the deployment-time reliability of learning-guided robot behavior (Rahman et al., 2021; Mallozzi et al., 2019; Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017) to decide whether one should continue using learning or resort to backup strategies in case of degraded performance. Most such approaches, however, focus on tackling issues associated with shortcomings of local perception or aim to monitor performance in terms of the underlying learned model's raw predictions as opposed to the overall task performance (Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017), thus limiting their effectiveness in evaluating the goodness of long-horizon behavior. As such, there is a need for broader introspection framework that can capture the aspects of long-horizon planning under uncertainty and enable a robot to not only identify poor behaviors but also recover from them and improve over time during deployment in previously unseen scenarios.

To illustrate what introspection might look like in the domain of planning under uncertainty, consider the scenario as demonstrated in Fig. 1.1. The robot is tasked to find a fork in a household setting. However, the robot does not yet have full knowledge about the environment. It knows that there is a bedroom, a kitchen, a living room and a bathroom, however, it doesn't know which of these rooms has a fork. As such, it would have to reason about where it should travel to so as to find the fork quickly despite missing or incomplete knowledge. With a strategy S_1, the robot always goes to the room that is nearest to it so as to find a fork. This strategy, as shown in Fig. 1.1(b) results in the robot first traveling to the bathroom, then the living room, and finally to the kitchen where it finds a fork. While the robot was able to complete the task successfully, we as humans can clearly see that the robot could have done a better job. For example, with the benefit of hindsight that the fork is in the kitchen, if the robot had instead traveled to the kitchen first, it would have found the fork much quicker. Our ability for such introspection given the benefit of hindsight can be imbued in a robot by simulating what the robot would have done if it had instead followed a different strategy S_2. Demonstrated in Fig. 1.1(c), using the knowledge obtained after completing the task with strategy S_1, we can simulate what the robot would



(a) Robot is tasked to find a fork.

(b) Path traveled by the robot to find a fork with strategy S_1.

(c) Simulated path with strategy S_2 after finding the fork.

Figure 1.1: **Motivating Example**: (a) A robot is tasked to find a fork in an environment, and (b) uses strategy S_1 to find the fork. (c) With the benefit of hindsight, it can then leverage the knowledge gathered to reason about how another strategy S_2 would have performed.

have done with a different strategy S_2 (which directly guides the robot to kitchen), and infer that it would have found the fork much quickly. Such introspection can be used to pick better behaviors—or avoid poor ones—and improve performance during deployment.

In this dissertation, we introduce representations and techniques that enable a robot to introspect its behavior while performing long-horizon tasks in environments that may not be fully known to the robot. Such introspection enables the robot to improve its deployment-time performance even when its behaviors are informed by learning-based methods which might be brittle to distribution shifts or even minor environment changes. Specifically, this dissertation focuses on demonstrating the effectiveness of such introspection in the context of robot planning under uncertainty where uncertainties arise from factors such as missing or incomplete knowledge about the environment where the robot is deployed in.

The main theme of this dissertation is to enable autonomous robots to introspect about their own behavior—to reason about how well they are performing and whether alternative behaviors could have led to better outcomes—so as to improve their deployment-time performance for long-horizon planning under uncertainty. In particular, this dissertation proposes *introspection* as a key framework that allows robots to assess and improve their

decision-making during deployment in unknown or partially known environments. Through this unifying theme, this dissertation explores a series of problems centered on enhancing effectiveness, reliability, and adaptability of robots in long-horizon tasks:

Data-efficient policy selection: How can robots introspectively reuse experience from a single deployment to reason about the performance of multiple alternative behaviors and efficiently select the best ones?

Learning and adaptation during deployment: How can introspection enable robots to recognize if and when their policies that are continually being learned or adapted online have improved and to quickly select from these evolving policies to enable improved deployment-time performance in novel environments?

LLM-informed robot planning: How can introspection be applied to broader and emerging areas of large language model (LLM)-informed planning where the robot's behavior may depend heavily on the choices of prompts and the LLM models themselves?

Together, these threads are unified by the central goal of developing introspective robots that can monitor, evaluate, and improve their behavior during deployment, bridging the gap between learning-informed robot planning and self-improving autonomous systems in the context of long-horizon planning under uncertainty. The following thesis statement summarizes the key insights of this dissertation.

Thesis Statement: *Models that allow robots to simulate or imagine alternative behaviors without having to deploy them are key to enabling introspection and deployment-time behavior selection for long-horizon planning under uncertainty, thereby achieving fast and reliable performance improvements—even when learning is used to inform the robot's behavior.*

The rest of the dissertation is organized as follows. We first discuss some background topics in the space of robot planning and decision making (Chapter 2). In Chapter 3, we introduce the techniques for introspection and demonstrate that such introspection allows the robot to quickly identify the best behavior from a family of predetermined behaviors

and thereby resulting in improved navigation performance when deployed in a wide variety of partially-mapped environments. In Chapter 4, we extend the idea of introspection further to scenarios where the robot is continually learning multiple strategies during deployment and has reason about which of these continually evolving strategies should be deployed for best performance in new unknown environments. In Chapter 5, we introduce a novel model-based planning abstraction for object search in partially-known environments that integrates the commonsense world knowledge of LLMs with classical planning framework, and also propose an approach for deployment-time selection of prompts and LLMs, enabling the robot to improve its object search performance during deployment. Finally, in Chapter 6, we conclude with a brief summary of the contributions made in this dissertation and discuss potential future works.

We give an overview of our contributions and publications in the rest of this chapter.

1.1 Contributions

The contributions of this dissertation are as follows:

- We introduce a novel technique for *introspection* to reason about alternative robot behaviors and their performances in the context of navigation under uncertainty. We demonstrate that such introspection can be leveraged to pick between behaviors during deployment (Chapter 3).

- We formalize the problem of *policy selection* as an instance of multi-armed bandit (MAB) and present a novel approach that leverages introspection for fast and data-efficient policy selection. We show that our policy selection approach has guarantees on asymptotic sub-linear regret while quickly converging to the best behaviors during deployment (Chapter 3).

- We demonstrate the effectiveness of introspection in the context of selecting between non-stationary policies which are continually being learned or adapted while the robot is deployed in environments that it may or may not have seen before (Chapter 4).

- We present a novel approach for LLM-informed model-based high-level planning for object search in partially-known environments that integrates predictions about uncertainty from LLMs and known traversal costs from the occupancy map (Chapter 5).
- Leveraging the introspection framework introduced in Chapter 3, we demonstrate fast deployment-time selection of best prompts and LLMs from a family of candidate prompts and LLMs used for guiding the robot behavior in object search tasks in partially-known environments (Chapter 5).

1.2 Publications

The contributions presented in this dissertation have been published in Paudel and Stein (2023); Paudel et al. (2024); Hossain et al. (2024); Paudel and Stein (2025); Paudel (2025), as listed below.

- Abhishek Paudel and Gregory J. Stein. Data-efficient policy selection for navigation in partial maps via subgoal-based abstraction. In *International Conference on Intelligent Robots and Systems (IROS)*, 2023.
- Abhishek Paudel, Xuesu Xiao, and Gregory J. Stein. Multi-strategy deployment-time learning and adaptation for navigation under uncertainty. In *Conference on Robot Learning (CoRL)*, 2024.
- Shahriar Hossain, Abhishek Paudel, and Gregory J. Stein. Enhancing object search by augmenting planning with predictions from large language models. In *CoRL Workshop on Learning Effective Abstractions for Planning*, 2024.
- Abhishek Paudel and Gregory J. Stein. Deployment-time selection of prompts for LLM-informed object search in partially-known environments. In *ICRA Workshop on Foundation Models and Neuro-Symbolic AI for Robotics*, 2025.
- Abhishek Paudel. Introspection for long-horizon robot planning under uncertainty. In *ICRA Doctoral Consortium*, 2025.

The following are the contributions presented in this dissertation that are under review as Paudel et al. (2026).

- Abhishek Paudel, Abhish Khanal, Raihan I. Arnob, Shahriar Hossain and Gregory J. Stein. Object search in partially-known environments via LLM-informed model-based planning and prompt selection. In *International Conference on Intelligent Robots and Systems (IROS)*, 2026.

Other manuscripts that I have contributed to and have been accepted for publication (Khanal et al., 2026) are listed as follows.

- Abhish Khanal, Abhishek Paudel, Hung Pham and Gregory J. Stein. Multi-robot learning-informed task planning under uncertainty. In *International Conference on Robotics and Automation (ICRA)*, 2026.

Following are the pre-prints that I have contributed to during my PhD (Paudel et al., 2021; Paudel, 2021, 2022a,b; Bhandari and Paudel, 2024; Arnob et al., 2026).

- Abhishek Paudel, Roshan Dhakal, and Sakshat Bhattarai. Room classification on floor plan graphs using graph neural networks. *arXiv preprint arXiv:108.05947*, 2021.

- Abhishek Paudel. Sophisticated students in boston mechanism and gale-shapley algorithm for school choice problem. *arXiv preprint arXiv:2108.05951*, 2021.

- Abhishek Paudel. Learning for robot decision making under distribution shift: A survey. *arXiv preprint arXiv:2203.07558*, 2022.

- Abhishek Paudel. Motion primitives based path planning with rapidly-exploring random tree. *arXiv preprint arXiv:2210.15784*, 2022.

- Prabin Bhandari and Abhishek Paudel. Using contextual information for sentence-level morpheme segmentation. *arXiv preprint arXiv:2403.15436*, 2024.

- Raihan I. Arnob, Max Merlin, Abhishek Paudel, Benned Hedegaard, George Konidaris and Gregory J. Stein. Effective task planning with missing objects using learning-informed object search. *arXiv preprint arXiv:2602.11468*, 2026.

Chapter 2: Background

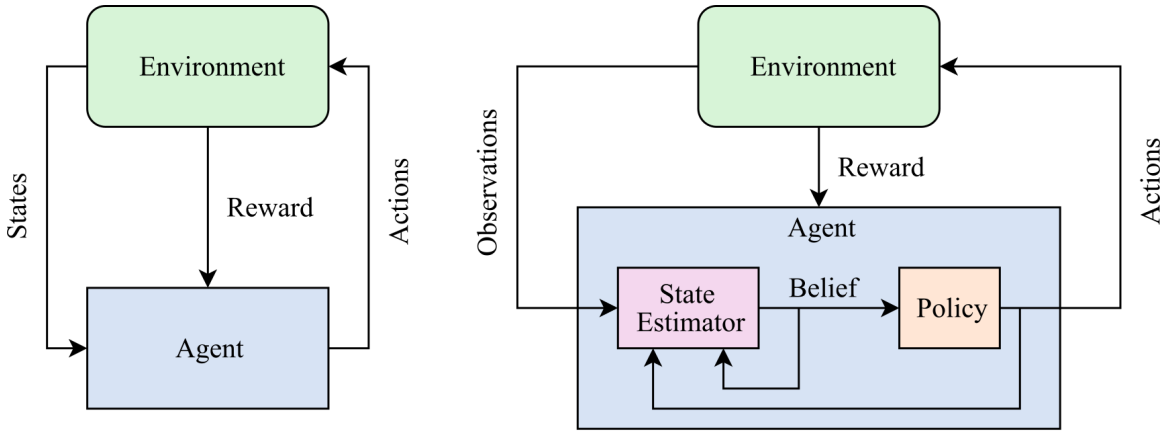
In this chapter, we discuss the topics and ideas that one should be familiar with to follow the rest of the thesis, with a particular focus on the foundations of planning and decision-making in robotics.

2.1 Markov Decision Process

A general framework for sequential decision making in fully observable, stochastic environments is provided by Markov decision process (MDP) (Sutton and Barto, 2018). An MDP consists of the following:

- A set of states, S
- A set of actions, A
- State transition probabilities, $P(s', s, a)$
- Reward function, $R(s)$

At each time step, the agent can fully observe its state and can take an action that transitions it to a new state as per the state transition probabilities with the agent receiving some reward in the process. The goal is to solve for a policy π that specifies what the agent should do at every possible state in order to maximize the expected total reward. A policy that maximizes the total reward that an agent can accrue in expectation is called an optimal policy. Value iteration and policy iteration are two algorithms that can be used to solve for an optimal policy given an MDP. Value iteration involves alternating between policy update based on the values and value computation based on the policy. A value function



(a) MDP (b) POMDP

Figure 2.1: Illustration of agent-environment interaction in (a) Markov decision process (MDP) and (b) Partially observable Markov decision process (POMDP)

for a given policy is defined over states to quantify how good it is to be in a particular state. The value of a state can be calculated using the Bellman equation as shown in Eq. (2.1).

$$V^{\pi*}(s) = R(s) + \gamma \max_{a \in A(s)} \sum P(s', s, a) V^{\pi*}(s') \tag{2.1}$$

Here, γ denotes the discount factor and is a real number between 0 and 1 that controls how much should future rewards be prioritized over the current reward. One can also represent values as a function of action in addition to state, which is known as Q-value or action-value. The corresponding Bellman equation for a Q-value is,

$$Q^{\pi*}(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a' \in A(s)} Q(s', a') \tag{2.2}$$

Value iteration assumes the equality in Eqs. (2.1) and (2.2) as an assignment to find optimal value function which is then used to solve for optimal policy. Policy iteration differs from value iteration in that it solves for value function at every iteration without keeping track of values from the last iteration and updates the policies using newly computed

values. Depending upon how much into the future the reasoning is considered, the horizon for sequential decision making could be finite or infinite. Finite horizon problems assume that the agent has a fixed number of time steps over which it can maximize its rewards, whereas infinite horizon problems do not have a fixed number of time steps predefined and can vary or even be infinite in principle.

2.2 Partially Observable Markov Decision Process

Markov decision process (MDP) assumes that the agent has complete knowledge of its state. However, in many real world problems, this assumption does not hold and the agent has to deal with state uncertainties in addition to action uncertainties. Partially observable Markov decision process (POMDP) allows for dealing with state uncertainties by formulating belief over states (Kaelbling et al., 1998). This means that the agent has to reason over every possible state it could be in at any given time making the problem more difficult to solve. Algorithms like value iteration that are used to solve an MDP become computationally intractable even for POMDPs with small state space. In the context of robotic systems, the problem of planning or navigating in unknown environments is generally formulated as a POMDP.

2.3 Reinforcement Learning

Although many problems can be formulated as an MDP (or a POMDP), the underlying transition probabilities and rewards may not always be directly accessible to the agent in many scenarios of interest. This means that the agent has to actually interact with the environment and learn through trial-and-error what it should do in order to maximize the return. Reinforcement learning (RL) provides a framework to solve such problems in scenarios where the agent does not have direct access to transition probabilities and/or rewards (Sutton and Barto, 2018). With reinforcement learning, the agent has to use

observed rewards to learn a policy that maximizes the expected total reward. This can be done using two kinds of approaches: model-based approach and model-free approach.

Model-based approaches focus on first learning the transition probabilities and rewards (combinedly known as the *model*) based on the agent's interaction with the environment and use this model to learn the optimal policy. The main advantage of model-based approaches is that they allow the agent to plan by reasoning about what would happen for a range of possible choices and thus make decisions based on available options. On the other hand, model-free approaches aim to solve for the optimal policy without first estimating the transition probabilities and rewards. Model-free approaches estimate value function or optimal policy from a series of interactions between the agent and the environment.

2.4 Multi-Armed Bandit

A multi-armed bandit (MAB) is a classic reinforcement learning problem in which there is a slot machine with n arms (bandits) and each arm has an independent reward distribution that is hidden from the agent pulling the arms (Sutton and Barto, 2018). A multi-armed bandit problem can be seen as an instance of MDP having only one state with actions that each correspond to pulling an arm from among n arms. When the agent pulls an arm, it gets some reward based on the underlying reward distribution of the arm. The goal is to maximize the total reward accumulated by pulling the arms. If the agent had the knowledge of the underlying reward distributions of each arm, it could always pull the arm with the highest expected reward. However, since the underlying reward distributions are not known to the agent, a common strategy is to first identify the arm with the highest expected reward through trial and error. This requires that the agent take into consideration the exploration and exploitation trade-off while pulling the arms so as to maximize the accumulated rewards while simultaneously learning to identify the most rewarding arm.

2.5 Goal-Directed Navigation

Goal-directed navigation is a fundamental problem in robotics, where a robot must move from a start location to a desired goal while avoiding obstacles and efficiently planning its path. In realistic scenarios, the robot often operates in environments that are partially known or completely unknown, requiring the integration of perception, planning, and decision-making under uncertainty.

Goal-directed navigation can be framed as a sequential decision-making problem. When the environment is fully known, the problem reduces to computing an optimal path, which can be solved using classical planning methods such as rapidly-exploring random tree (RRT) (LaValle, 1998; Karaman and Frazzoli, 2011) or A* (Hart et al., 1968). In stochastic environments, this formulation naturally extends to an MDP, where states correspond to the environment map and the robot's location, actions correspond to motion commands, and rewards are typically structured to encourage reaching the goal efficiently while penalizing collisions or unsafe states.

When the robot has incomplete knowledge about its environment or uncertain state estimates, the navigation problem is more accurately modeled as a POMDP. In practice, the robot must maintain a belief not only over its own pose but also over the map of the environment; this joint estimation is typically handled by simultaneous localization and mapping (SLAM). SLAM provides the evolving belief state—over both pose and map—that a POMDP planner relies on to make informed decisions under uncertainty. However, solving POMDPs exactly is computationally intractable for large-scale navigation problems; therefore, approximate methods are often used, such as belief-space planning, Monte Carlo tree search (MCTS), and policy search in parameterized policy spaces.

In addition to classical and POMDP-based approaches, reinforcement learning (RL) has been increasingly applied to goal-directed navigation. In model-free RL, the robot learns a navigation policy directly from interactions with the environment without requiring an explicit map or transition model. Model-based RL, on the other hand, allows the robot to predict the outcomes of actions and plan efficiently over multiple time steps. A key

challenge in RL-based navigation is balancing exploration (gathering information about the environment) and exploitation (moving toward the goal efficiently), which is reminiscent of the multi-armed bandit problem discussed previously.

Goal-directed navigation also often leverages hierarchical structures, where high-level planning determines waypoints or subgoals, and low-level controllers handle local obstacle avoidance and motion execution. Such hierarchical approaches improve efficiency and scalability, especially in large and dynamic environments.

Chapter 3: Data-Efficient Policy Selection for Navigation in Partial Maps via Subgoal-Based Abstraction

3.1 Introduction

In this chapter, we address the first component of our dissertation—the development of introspection framework that allows robots to reason about how alternative behaviors could have performed during deployment. Specifically, we focus on the problem of navigation in partially mapped environments that the robot may not have seen before and introduce *offline alt-policy replay* as a practical realization of introspection. This technique enables a robot to reuse data from a single trial to estimate how well other policies might have performed, enabling selection of best behaviors while avoiding poor ones and hence improving deployment-time performance for long-horizon navigation under uncertainty. The work presented in this chapter has been published in Paudel and Stein (2023).

We enable a robot to quickly identify the best performing policy from a set of policies when it is deployed in partially-mapped environments to navigate to an unseen goal. Goal-directed navigation in partially-mapped environments is an important capability for autonomous robots. A robot tasked with navigating to a faraway goal in an unknown office building will need to build the map as it travels, plan through unknown regions of the map, and decide how to act once that space is revealed. Efficient navigation in such scenarios requires that the robot be able to reason about the impacts of its actions far into the future, commonly known as the *long-horizon planning* problem, which can be formulated as a POMDP (Kaelbling et al., 1998). Solving such POMDPs is computationally intractable, and therefore many existing approaches for planning in such scenarios leverage learning to guide the robot's behavior (Stein et al., 2018; Richter et al., 2014; Wayne et al., 2018; Zhu et al., 2017b; Kahn et al., 2018; Mirowski et al., 2016).

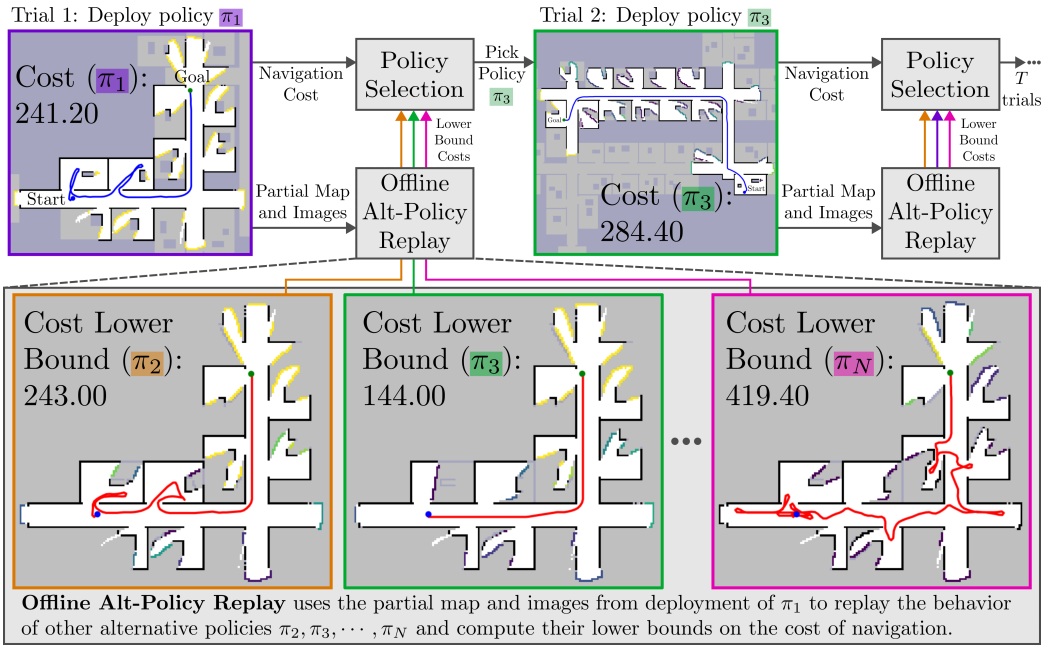


Figure 3.1: **Overview of our approach for data-efficient policy selection for navigation in partial maps.** Our approach relies upon *offline alt-policy replay*, a procedure to compute lower bounds of navigation costs for alternate policies after deployment, bounds used to constrain selection.

When navigating through a partial map, learning often relies on sensor observations collected onboard the robot (e.g., images and laser scans) to make predictions about the goodness of the robot's actions, thus guiding its behavior towards actions that would have resulted in effective performance in its training environments. However, in general, the robot may have access to a *family* of such learning-informed policies, each trained in a different environment, and must select one from these to guide its behavior during deployment. After planning with one of these policies, the robot must use this policy's cumulative performance to determine whether it should select that policy again, or switch to another in hopes of reducing navigation cost: an instance of *model selection* applied to *policy selection*.

Model selection approaches (Lee et al., 2021; Reisinger et al., 2008; Lattimore and Szepesvári, 2020; Lai et al., 1985; Gittins, 1979; Thompson, 1933; Kuleshov and Precup, 2014; Mannor and Tsitsiklis, 2004) aim to identify the best performing models from a set of

pre-defined behaviors by deploying each policy multiple times and evaluating their performance. Bandit algorithms (Lai et al., 1985; Thompson, 1933; Gittins, 1979), for example, trade off between *exploitation* (choosing the learning-informed policy that has performed best so far) and *exploration* to pick another policy that *could* improve performance. Upper confidence bound (UCB) bandits (Lai et al., 1985) compute statistical bounds on potential performance for each available policy based on the observed performance of each and how often they have been deployed. However, bandit algorithms are often black-box, and so tightening these bounds often requires the robot to go through multiple planning attempts and incur poor behavior before such policies can be ruled out. Owing to the expense associated with lengthy (and poor-performing) navigation trials, data efficiency is critical for such selection approaches to be practically useful.

Instead, we reuse data collected during a trial to evaluate how well other alternative policies could have performed and use this evaluation to tighten the bounds on policy selection: a procedure for *introspection* we refer to as *offline alt-policy replay*, illustrated in Fig. 3.1. Based on the information collected during each of its trials, the robot, without collecting additional information from the environment, replays how each of its other alternative policies *could* have performed. Our offline alt-policy replay approach makes an optimistic assumption about space not seen during the navigation trial (i.e., that the robot will spend only a minimal amount of time in unseen space) allowing us to compute a strict lower bound on the possible performance of each policy. This information constrains bandit selection—exploration should never select a policy that could not have improved performance during deployment so far—and improves data efficiency of selection.

Our offline alt-policy replay approach depends upon having a planning strategy for which an accurate bound on its would-be performance can be determined via only the limited information collected during the robot's trials so far. Not all planning strategies are amenable to such replay, including model-free approaches trained via deep reinforcement learning (Mallozzi et al., 2019; Kulhánek et al., 2019), which can be brittle to changes in their inputs (Henderson et al., 2018; Huang et al., 2017; Xiang et al., 2018). It is a

key insight of this work that the recent model-based, learning-augmented learning over subgoals planning (LSP) of Stein et al. (2018) is well-suited for this purpose. Under the LSP abstraction, learning is used only to make robot-pose-agnostic predictions about statistics of unseen space: e.g., the likelihood that a particular region of space will lead to the unseen goal. Since its approach to learning is robust to changes in the robot's location, offline replay of LSP policies yields accurate lower bounds on cost that can be used to constrain bandit-like selection.

In this work, we develop a procedure for data-efficient policy selection called *offline alt-policy replay*, which leverages the information collected during navigation to evaluate how well other alternative policies could have performed and thereby constrain bandit-like selection of a set of Learning over Subgoals Planners (Stein et al., 2018). Our robot is deployed in simulated maze and office-like environments with multiple learning-informed policies: each relies on the learning over subgoals planning (LSP) abstraction and is trained in a different environment. We demonstrate that our approach is able to quickly reduce the average navigation cost within much fewer trials compared to baseline upper confidence bound (UCB) bandit approach which generally takes longer to converge and has higher cumulative regret than our approach. Our results validate that our approach reliably performs better than the UCB bandit and is often able to perform selection much more quickly.

3.2 Related Work

Planning under Uncertainty POMDPs (Kaelbling et al., 1998; Littman et al., 1995) provide is a general framework to represent navigation and exploration under uncertainty (Candido and Hutchinson, 2011; Kurniawati et al., 2011). However, they are computationally intractable to solve, and so learning is used for planning (Stein et al., 2018; Richter et al., 2014; Wayne et al., 2018; Zhu et al., 2017b; Kahn et al., 2018; Mirowski et al., 2016). Deep reinforcement learning approaches are also widely used for planning in partially-mapped

environments (Gupta et al., 2017; Zhang et al., 2017; Tai et al., 2017), but they are often limited to short-horizon planning.

Model Selection Online model selection approaches in reinforcement learning (Lee et al., 2021; Reisinger et al., 2008; Ghosh and Chowdhury, 2022; Pacchiano et al., 2020) generalize bandits and aim to identify best models from a set of available models. Bandit algorithms (Lai et al., 1985; Thompson, 1933; Gittins, 1979), which are often black-box, trade off between exploitation and exploration requiring the robot to go through multiple trials with poor behavior before identifying a better policy.

Runtime Monitoring So-called *runtime monitoring* approaches aim to evaluate the deployment-time reliability of learning-guided robot behavior (Rahman et al., 2021; Mallozzi et al., 2019; Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017) to decide whether one should continue using learning or resort to backup strategies in case of degraded performance. Most such approaches, however, focus on tackling issues associated with shortcomings of local perception or aim to monitor performance in terms of the underlying learned model's raw predictions as opposed to the overall task performance (Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017), thus limiting their effectiveness in evaluating the goodness of long-horizon behavior.

3.3 Problem Formulation

3.3.1 Goal-directed Navigation in Partial Maps

Our robot is placed in a partially-mapped environment and tasked to reach a point-goal in unseen space in minimum expected distance. We formulate this problem as a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998). The belief state b_t captures the partially-observed map and the robot's pose at time t. The robot is equipped with a planar laser scanner, and so it is capable of reliably mapping its surroundings in places it has already seen and determining its location with high precision, a subset of POMDPs recently coined as a *locally* observable Markov decision process (Merlin et al., 2020). Robot

behavior is guided by a policy π that specifies the action the robot should take from the belief b_t, stored as a partial occupancy map m_t and the set of visual observations collected by the robot. During deployment, performance is measured as the average distance traveled to reach the goal across a number of trials; a trial is a single traversal from start to goal in a previously-unseen map.

3.3.2 Policy Selection during Deployment

We consider that the robot has access to a set of policies $\mathcal{P} = \{\pi_1, \pi_2, \cdots, \pi_N\}$, and its objective is to pick the policy that minimizes the expected cost during deployment,

$$\pi^* = \underset{\pi \in \mathcal{P}}{\operatorname{argmin}} \, \mathbb{E}[C(\pi)] \tag{3.1}$$

where $\mathbb{E}[C(\pi)]$ is the expected cost of policy $C(\pi)$ during deployment. However, in general we will not have direct access to the expected cost and must instead estimate it during deployment via execution of multiple trials.

3.3.3 Limits of Black Box Policy Selection for Planning in a Partial Map

Black box policy selection methods—e.g., many bandit algorithms—aim to select policies by balancing minimization of the average cost observed so far and exploration, so as to occasionally select policies that have the potential to improve performance. For trial $k+1$, the upper confidence bound (UCB)[1] (Lai et al., 1985; Sutton and Barto, 2018) bandit algorithm selects the next policy $\pi^{(k+1)}$ according to

$$\pi^{(k+1)} = \underset{\pi \in \mathcal{P}}{\operatorname{argmin}} \left[\bar{C}_k(\pi) - c\sqrt{\frac{\ln k}{n_k(\pi)}} \right] \tag{3.2}$$

[1]Eq. (3.2) uses a lower confidence bound (LCB) instead of a UCB, since our performance estimates are represented as *costs* instead of *rewards* and is therefore minimized. As such, we use the more common term UCB to mean the approach in general rather than the upper bound itself.

where $\bar{C}_k(\pi)$ is the average cost over trials 1–k in which policy π was selected, $n_k(\pi)$ is the number of times policy π was selected (up to trial k), and $c > 0$ is a parameter controlling the balance between exploration and exploitation.

However, for navigation under uncertainty, each trial is expensive to execute, and the results of these trials depend on the environment map and so often have high variance. Policy selection via Eq. (3.2) is therefore often problematically slow to converge, limiting the utility of this approach in practice. Narrowing the confidence bounds on the expected performance of each policy requires deploying them, often multiple times, and therefore the robot would potentially need to go through repeated attempts of poor performing behavior before such policies can be ruled out. Instead, we aim to improve the data efficiency of selection via a *white-box selection strategy* that uses offline alt-policy replay to compute bounds on how well each policy *could have done* in the trials so far, and uses these bounds to constrain selection.

3.4 Overview: Data-Efficient Policy Selection via Offline Alt-Policy Replay

If we are to accelerate policy selection, we must be able to quickly tighten the bounds on expected performance for each policy, prioritizing selection of the most promising policies with fewer trials. As such, we seek to constrain policy selection by determining how well an alternative policy *could have performed* if it had instead been in charge. While we cannot re-deploy the robot to repeat the same trial with another policy, information collected during the trial can be used to scrutinize alternative behaviors and determine a lower bound on their performance even without deployment.

We seek to use *deployment-time offline alt-policy replay* to compute a lower bound on the performance of policies that did not control the robot's behavior during the trial. Using the information collected by policy π during its trial, we can replay how every other alternative policy $\pi' \in \mathcal{P} \setminus \pi$ could have performed. Our offline alt-policy replay makes optimistic

assumptions about space not seen during the navigation trial (i.e., that the robot will only spend a minimal amount of time in unseen space), allowing us to compute a strict lower bound on the mean performance of each policy so far (see Sec. 3.7.3, 3.7.4). We use this lower bound on the mean $\bar{C}^{\text{lb}}$ to constrain UCB bandit selection, so that policies that could not have improved performance are not selected. Our *Constrained* UCB Bandit algorithm selects according to

$$\pi^{(\text{k}+1)} = \underset{\pi \in \mathcal{P}}{\operatorname{argmin}} \left[\max \left(\bar{C}_k^{\text{lb}}(\pi), \bar{C}_k(\pi) - c\sqrt{\frac{\ln k}{n_k(\pi)}} \right) \right] \tag{3.3}$$

Intuitively, Eq. (3.3) always picks the tighter (i.e., higher) of the lower bounds between (i) $\bar{C}_k^{\text{lb}}$, the lower bound computed via offline alt-policy replay, and (ii) bound computed by UCB algorithm as shown in Eq. (3.2) and then minimizes over this bound to pick a policy for next trial.

3.5 Offline Policy Replay Requires a Planning Approach Robust to Vantage Point Change

Scrutinizing alternative behavior to determine the lower bound on cost $\text{C}_k^{\text{lb}}(\pi)$ needed for selection via our constrained UCB bandit algorithm, Eq. (3.3), requires that we can perform offline alt-policy replay of robot behavior under an alternative policy without actually deploying the robot. In general, replaying a policy offline requires an ability to generate observations from poses the robot may not have visited, which for many learning-informed planning strategies in this domain will not accurately reflect how the policy would have behaved if it had been in control of the robot. Many approaches to vision-informed navigation under uncertainty, particularly those relying on deep reinforcement learning (Kulhánek et al., 2019; Mirowski et al., 2016), require observations (images) from poses and vantage

points not visited during the original trial and can be brittle to even small changes (Henderson et al., 2018), and so replay of such policies is unlikely to yield an accurate lower bound on cost. As such, we instead require an approach to planning that is robust to changes in viewpoint and a kind of policy that is robust to minor changes in robot pose and corresponding observations, and can reliably reach the goal even in environments where learning informs poor behavior.

3.6 Preliminaries: The Learning over Subgoals Model-Based Planning Abstraction

It is a key insight that the learning over subgoals planning (LSP) of Stein et al. (2018) is well-suited for offline alt-policy replay while being suitable for long horizon planning in partially-mapped environments. LSP is a high level planning framework in which navigation in a partially-mapped environment is formulated as a partially observable Markov decision process (POMDP). In this abstraction, *subgoals* represent the robot's high-level actions of navigating to a frontier: i.e. a boundary between free and unseen region in the map. Each action a_t corresponding to a subgoal has three properties associated with it: likelihood that the subgoal leads to the goal $P_S(a_t)$, expected cost of reaching the goal via the subgoal $R_S(a_t)$, and expected cost of failure or exploration in case the subgoal doesn't lead to the goal $R_E(a_t)$. A neural network, parameterized by θ, is trained via supervised learning to estimate the subgoal properties $\mathcal{R}_\theta = \{P_{S,\theta}, R_{S,\theta}, R_{E,\theta}\}$ based on image observations centered at the subgoal. A factored form of Bellman equation shown in Eq. (3.4) is used to calculate the expected cost of each action,

$$Q_\theta(b_t, a_t) = D(b_t, a_t) + P_{S,\theta}(a_t)R_{S,\theta}(a_t) + (1 - P_{S,\theta}(a_t)) \left[R_{E,\theta}(a_t) + \min_{a \in \mathcal{A}(b_t) \setminus a_t} Q_\theta(b_t, a) \right] \tag{3.4}$$

where $D(b_t, a_t)$ is the known cost of navigating to a frontier. The robot's policy is then to minimize expected cost:

$$\pi_\theta(b_t) = \underset{a \in \mathcal{A}(b_t)}{\operatorname{argmin}} Q_\theta(b_t, a) \tag{3.5}$$

Since learning is used only to make robot-pose-agnostic predictions of subgoal properties using panoramic image oriented so as to face the subgoal, LSP is robust to changes in the robot's location and corresponding image observations, and is therefore suitable for offline alt-policy replay.

3.6.1 Planning via Learning over Subgoals Planning

At each time step, the robot selects a high-level action a_t via Eq. (3.4) that specifies the subgoal towards which the robot will navigate. Upon selecting a high-level action, the robot computes a cost grid over the observed map via A* (Hart et al., 1968) and selects a low-level (short-horizon) motion primitive that makes the most progress towards the subgoal. The robot (i) executes this primitive action, (ii) updates the partial map via laser scanner observations, (iii) recomputes the subgoals (and, if necessary the subgoal properties P_S, R_S, and R_E from panoramic images collected onboard the robot), and finally (iv) replans via Eq. (3.4). As planning via LSP continually seeks out unexplored space until the goal is reached, it is guaranteed to reach the goal if a feasible path exists, even when learning informs poor behavior.

3.6.2 Network Architecture and Training

The neural network takes as input a 128×512 RGB panoramic image centered on a subgoal, the relative subgoal location and the relative goal location. Our neural network architecture and training procedure closely follow that of Bradley et al. (2021). The input image is encoded by first passing through 4 convolutional layers and then concatenated with the features representing the relative locations of subgoal and goal. These concatenated features

go through 9 convolutional layers and then 5 fully connected layers to output 3 subgoal properties: P_S, R_S, and R_E.

To generate training data, we conduct hundreds of trials in previously unseen maps in which the robot navigates from start to goal via a non-learned heuristic-driven planner, collecting images of the environment at each step. The target labels for P_S, R_S, and R_E corresponding to these images for observed subgoals are computed using the underlying known map. Labels for P_S correspond to whether or not a path to goal exists via a subgoal. Labels for the costs correspond to the travel distance to reach the goal through unknown space if the goal can be reached via a subgoal (for R_S) and to a heuristic cost approximating the distance robot would need to travel before meeting a dead end if the goal cannot be reached via a subgoal (for R_E). Further details on data collection and training can be found in Stein et al. (2018) and Bradley et al. (2021).

3.7 Approach: Offline Alt-Policy Replay via Learning over Subgoals

Our approach leverages information collected during a trial to perform offline replay of policies. During offline alt-policy replay, we make optimistic or simply-connected assumptions about the unseen space to compute an approximate lower bound cost for these policies which is then used to constrain UCB bandit to accelerate policy selection.

3.7.1 Information Collection during a Trial

During a trial (a single navigation from start to goal), we record information needed for the offline replay of other policies. For a trial k, the policy $\pi^{(k)}$ determines behavior, producing a *record* $\mathcal{Z}_k$ that includes a list of tuples (X_t, I_t) where $X_t = (x_t, y_t, \phi_t)$ is the robot pose at time t and I_t is a panoramic image collected by the robot at time t. The record $\mathcal{Z}_k$ also includes the partially-known occupancy grid map m_{final} obtained after reaching the

goal. The record $\mathcal{Z}_k = \{(X_t, I_t)_{t \in 1 \cdots M}, m_{\text{final}}\}$ is then used to perform offline replay of robot behavior guided by every other policy.

3.7.2 Offline Alt-Policy Replay Overview

Upon completion of a trial (e.g., trial k), the robot seeks to replay behavior of all other LSP-policies $\pi' \in \mathcal{P} \setminus \pi^{(k)}$ using the record $\mathcal{Z}_k$ it collected during the trial. Offline alt-policy replay proceeds similarly to online planning via LSP, as described in Sec. 3.6.1: at every replay time step τ, the robot updates its map using a simulated laser scan, computed by ray-casting into the 2D partial map m_{final}, and subsequently recomputes the available subgoals, each corresponding to a boundary between free and unknown space.

To compute the subgoal properties P_S, R_S, R_E associated with exploration into a particular region of unseen space, the robot requires panoramic images to be fed into its neural network (Sec. 3.6.2). Though images will not be available at every point in the environment during offline alt-policy replay, subgoal properties are instead estimated from nearby images stored in $\mathcal{Z}_k$. Specifically, we find the pose X_{nearest} from $\mathcal{Z}_k$ nearest to the current (replay) pose X_τ that can see the subgoal of interest and then use its associated image I_{nearest} to estimate the subgoal properties. Using the estimated subgoal properties, the robot selects a high-level action via Eq. (3.4), executes a low-level motion primitive to navigate towards the selected subgoal-action, and this process repeats. Critically, if we replay the deployed policy $\pi^{(k)}$ using its own record $\mathcal{Z}_k$, we recover its behavior exactly.

3.7.3 Computing an Approximate Lower Bound Cost

As the robot navigates to goal during offline alt-policy replay, it might attempt to enter space not seen by the robot planning with policy $\pi^{(k)}$ and thus unknown in the partial map m_{final}. Since this space is not known, we cannot know precisely what will happen in this unseen space and so we instead make conservative assumptions about what *could* happen: we assume that the robot either (i) reaches the goal in the shortest possible distance through unseen space or (ii) is immediately turned around and must pursue a different route to the

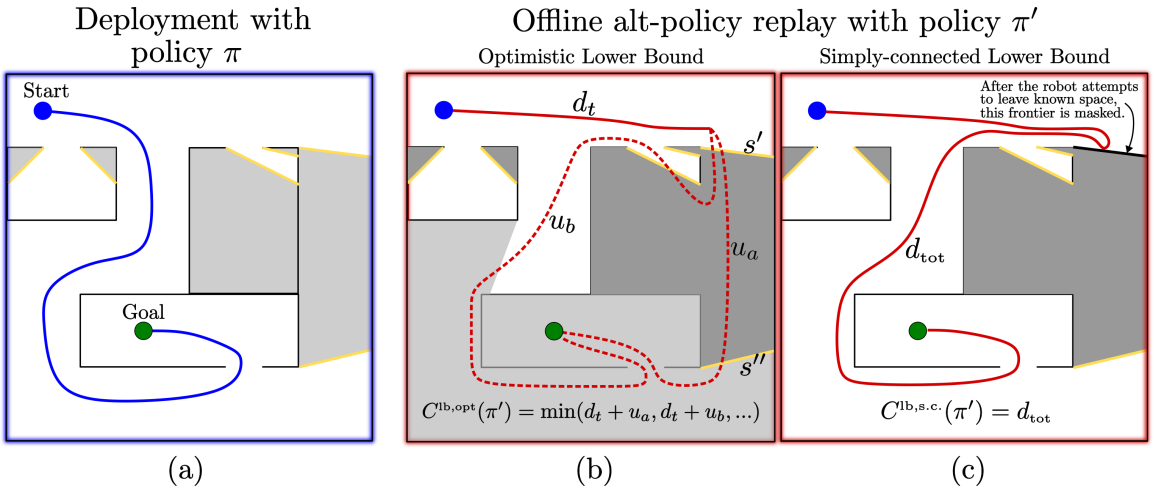


Figure 3.2: **Lower Bound Cost Approximation**: (a) Policy π guides the robot in a trial. (b) During offline alt-policy replay, policy π' attempts to leave known space via subgoal s' to try to reach goal with proposed path u_a. The minimum of such paths obtained during replay gives the optimistic lower bound of policy π'. (c) With simply-connected assumption, the lower bound is the net distance travelled under policy π' during offline replay.

goal. Based on what we know about unseen space—e.g., whether or not the environment is known to be simply-connected or the likelihood that there exists an undiscovered shortcut to the goal—we can make different assumptions and use them to compute lower bounds on the cost. Here (and visualized in Fig. 3.2) we discuss these different assumptions, their implications, and when each is appropriate.

Optimistic Cost Lower Bound If we have no prior information about the structure of unseen space, we make an *optimistic* assumption about unseen space: that all unseen space is free and could therefore provide a potential route to the goal. Under this assumption, whenever the actions from offline-replayed policy leads the robot to leave known space (via a subgoal s'), we assume the robot *could* have reached the goal via the shortest possible path in m_{final} that passes through s'. Fig. 3.2(b) shows two examples of such alternative paths. Upon attempting to leave known space, we compute the optimistic path cost, mask that particular frontier (so that the robot must continue exploration through known space),

and proceed with navigation. We generate such optimistic path costs every time the offline-replayed robot attempts to leave known space. The shortest of these paths is the optimistic lower bound cost $C_k^{\text{lb,opt}}(\pi')$ for a replayed policy π' in trial k.

Simply-Connected Cost Lower Bound If we were to know in advance that the environments were simply-connected, we could come up with a tighter lower bound on the cost, since the shortest path to the goal should exist within known space in m_{final} and routes that leave observed space will not reach the goal. During offline-replayed navigation, whenever the robot attempts to leave known space, the boundary through which it aims to leave is masked as an obstacle before it is allowed to do so, forcing it to turn back and seek an alternate route to the goal, Fig. 3.2(c). The total distance traveled to reach the goal is the simply-connected lower bound cost $C_k^{\text{lb,s.c.}}(\pi')$ for a replayed policy π'.

Weighted Approximate Cost Lower Bound In many environments, the optimistic lower bound is not particularly tight, and so does not help to accelerate selection, yet if the environment is not known to be simply connected, the simply connected lower bound may be too high and not a lower bound on cost. Often, we can use our prior understanding of an environment to determine the likelihood that alternative shortcuts to the goal could exist in space the robot did not see during its trial. We introduce a parameter p_{short} that denotes the likelihood of the existence of a shorter path to goal. This parameter is used to compute a third *approximate* lower bound, defined as the weighted sum of the optimistic and simply-connected bounds:

$$C_{k,p}^{\text{lb,wgt}}(\pi') = p_{\text{short}} C_k^{\text{lb,opt}}(\pi') + (1 - p_{\text{short}}) C_k^{\text{lb,s.c.}}(\pi') \tag{3.6}$$

While the weighted approximate lower bound cost may not strictly be a lower bound, and so may violate guarantees of asymptotic regret bounds afforded by UCB bandit selection, we will show that this approximate lower bound cost helps to achieve good empirical performance (Sec. 3.8.2).

3.7.4 Combining Observed and Replayed Lower Bound Costs

As trials proceed, we compute an approximate lower bound on the mean $\bar{C}^{\text{lb}}$, that combines both averaged performance $\bar{C}_k(\pi)$ of policy π until trial k and the mean replayed lower bound $\bar{C}^{\text{lb,rep}}$, the average cost computed from offline alt-policy replay via one of the lower bounds defined in Sec. 3.7.3:

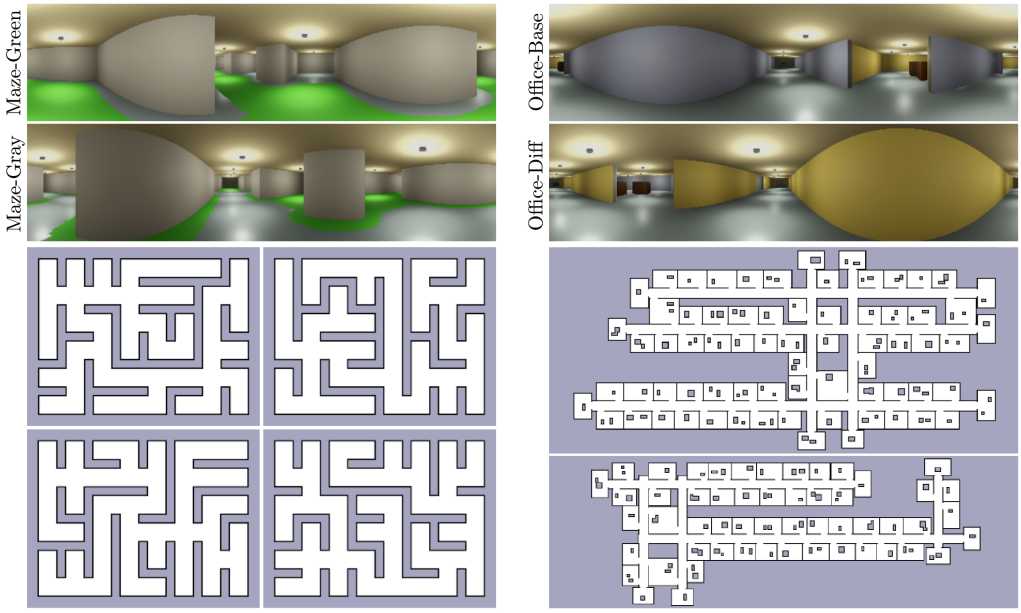
$$\bar{C}_k^{\text{lb}}(\pi) = \frac{n_k(\pi)\bar{C}_k(\pi) + n_k^{\text{rep}}(\pi)\bar{C}^{\text{lb,rep}}}{n_k(\pi) + n_k^{\text{rep}}(\pi)} \tag{3.7}$$

where $n_k(\pi)$ and $n_k^{\text{rep}}(\pi)$ are the number of times each policy has been deployed and offline-replayed, respectively.

Constrained policy selection based on modified UCB bandit formulation discussed in Sec. 3.4 leverages this lower bound $\bar{C}_k^{\text{lb}}(\pi)$ to quickly identify the best-performing policy.

3.8 Experimental Results

We perform experiments in simulated maze and office-like environments (Fig. 3.3), evaluating our constrained policy selection approach, Const-UCB, Eq. (3.3) against baseline UCB bandit selection, Eq. (3.2). We conduct 200 deployments, each consists of 100 navigation trials, each in a distinct, procedurally-generated map not yet seen by the robot. At the outset of each deployment, the robot starts with a randomly selected policy. The policy for subsequent trials is selected via our Const-UCB approach or the UCB bandit. Both our Const-UCB approach and the baseline UCB bandit approach use an exploration parameter $c = 100$ in all experiments, empirically chosen on a held out test set to achieve good performance on the baseline UCB bandit approach. We note that the results are not particularly sensitive to changes in c. We additionally show results of deploying each policy individually in the absence of policy selection to illustrate the necessity of deployment-time selection for good performance across environments.



(a) Maze Environments (b) Office Environments

Figure 3.3: **The Simulated Environments.** Robot-view panoramic images from simulation environments (top two rows) and samples of maps from (a) maze (b) office-like environments. All our experiments are conducted in simulated environments rendered using the Unity game engine.

3.8.1 Maze-centric Results

Our simulated maze environments consist of randomly generated simply-connected mazes in which the robot needs to navigate from start to unseen goal. Each generated map is unique with randomized start and goal poses. To study the versatility and effectiveness of our approach, we design three variations of the maze environments and train an LSP-based

Table 3.1: Average navigation cost for each policy in maze-centric environments without policy selection

	Maze-Green	Maze-Gray	Maze-Random
Non-learned	206.05	194.37	<u>177.31</u>
LSP-Maze-Green	<u>150.39</u>	483.20	557.99
LSP-Maze-Gray	618.87	<u>154.03</u>	418.98
LSP-Maze-Random	231.71	238.22	180.23

	Deployment Environment	Maze-Green			Maze-Gray			Maze-Random		
	Num of Trials (k)	$k = 10$	$k = 40$	$k = 100$	$k = 10$	$k = 40$	$k = 100$	$k = 10$	$k = 40$	$k = 100$
Average Nav. Cost (mean)	UCB-Bandit (baseline)	250.04▲	190.08♦	170.88■	230.66▲	187.71♦	172.10■	290.89▲	216.87♦	194.20■
	Const-UCB:$C^{\text{lb,opt}}$ (ours)	175.38▲	159.14♦	155.08■	173.96▲	163.50♦	158.81■	209.44▲	186.21♦	181.20■
	Const-UCB:$C^{\text{lb,s.c.}}$ (ours)	**163.56**▲	**153.12**♦	**152.13**■	**165.62**▲	**158.06**♦	**155.94**■	**197.30**▲	**182.14**♦	**179.66**■
	Best Single Policy[2]	150.39	150.39	150.39	154.03	154.03	154.03	177.31	177.31	177.31
Cumul. Regret (mean)	UCB-Bandit (baseline)	1327.0△	3105.9◇	4761.4□	994.1△	2444.8◇	3879.4□	1472.0△	3391.9◇	4892.1□
	Const-UCB:$C^{\text{lb,opt}}$ (ours)	369.1△	813.4◇	1180.5□	278.2△	698.9◇	1076.9□	417.8△	918.6◇	1249.1□
	Const-UCB:$C^{\text{lb,s.c.}}$ (ours)	**259.2**△	**446.6**◇	**573.3**□	**208.4**△	**419.1**◇	**565.0**□	**309.5**△	**600.6**◇	**786.9**□

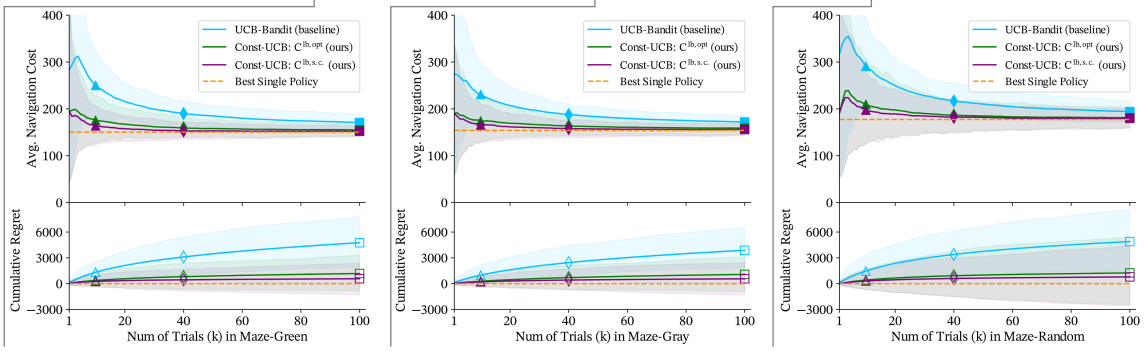


Figure 3.4: **Average Navigation Cost (mean) and Cumulative Regret (mean) for deployments in maze environments, Fig. 3.3(a).** Each deployment consists of 100 randomized navigation trials, each in a previously unseen maze. Mean cost and regret are computed across 200 randomized deployments. For our approach Const-UCB, we show results with optimistic $C^{\text{lb,opt}}$ and simply connected $C^{\text{lb,s.c.}}$ lower bounds as discussed in Sec. 3.7.3. The solid lines denote the mean, and the shaded regions show 10^{th} to 90^{th} percentile. The symbols: triangle, diamond and square denote average cost (filled) and cumulative regret (unfilled) at 10^{th}, 40^{th} and 100^{th} trial respectively in both the table and the plot for each environment.

policy (Sec. 3.6) in each. Shown in Fig. 3.3, the three maze environment variations are as follows:

Maze-Green The floor is generally gray, but a green path on the ground connects the start to goal. The green color is thus a signal an experienced agent should recognize as leading towards the unseen goal.

Maze-Gray Similar to the Maze-Green environment, yet the color of the floor and path are flipped: the floor is green with a gray path to the goal. Thus, this environment should mislead policies trained in Maze-Green.

[2]For each environment, "Best Single Policy" refers to the policy which incurs minimum cost when deployed in that environment. The costs incurred by Best Single Policy are underlined in Table 3.1 and 3.2.

Maze-Random The green path on gray floor is placed randomly and is not a reliable route to goal.

We train a LSP-based policy in each—yielding LSP-Maze-Green, LSP-Maze-Gray, and LSP-Maze-Random—following the procedure in Sec. 3.6.2. Each LSP-based policy is trained on 500 randomly generated mazes. Each maze map during a navigation trial is also distinct and is not a part of the training set. We also deploy a non-learned optimistic baseline policy that our approach can select; the non-learned policy, instead of using a neural network, uses optimistic heuristics about the environment to compute the subgoal properties, namely that each subgoal could lead to the unseen goal: i.e., $P_S = 1$.

We evaluate policy selection approach across 100 trials in each maze variation. Each trial consists of a randomly generated maze not present during the training of corresponding policies. Policy selection evaluation for each trial[3] is aggregated over 200 randomized deployments to compute statistics; we compute the mean and upper and lower 10^{th}-percentile and show results in Fig. 3.4. We show average navigation cost and cumulative regret accrued until k^{th} trial for each approach. UCB-Bandit is the baseline policy selection approach using Eq. (3.2), and Const-UCB corresponds to our constrained policy selection approach using Eq. (3.3).

The results in Fig. 3.4 show that in all environments, our constrained UCB (Const-UCB) bandit approach accelerates policy selection, reducing average navigation costs within far fewer trials than is possible with the UCB-Bandit alone. Consequently, our Const-UCB approach accumulates significantly lower mean cumulative regret compared to UCB bandit: 88% lower in Maze-Green, 85% lower in Maze-Gray and 84% lower in Maze-Random owing to quickly ruling out poor performing policies. The maze environments are constructed so as to be simply-connected and selection with the simply-connected lower bound $C^{\text{lb,s.c.}}$ achieves the best performance. Even if we did not know in advance that the environments

[3]For computational efficiency when computing statistics, each deployment randomly samples a subset of 100 distinct evaluation scenarios from a set of 150, run in advance of model selection for each policy.

	Deployment Environment	Maze-Green			Office-Base			Office-Diff		
	Num of Trials (k)	$k = 10$	$k = 40$	$k = 100$	$k = 10$	$k = 40$	$k = 100$	$k = 10$	$k = 40$	$k = 100$
Average Navigation Cost (mean)	UCB-Bandit (baseline)	213.90▲	184.62◆	168.87■	648.02▲	503.31◆	470.43■	638.02▲	508.72◆	480.22■
	Const-UCB:$C^{\text{lb,opt}}$ (ours)	174.03▲	158.88◆	155.03■	642.21▲	505.00◆	469.48■	622.17▲	508.17◆	475.46■
	Const-UCB:$C^{\text{lb,wgt}}_{\text{p=0.5}}$ (ours)	165.65▲	154.23◆	152.63■	467.63▲	451.30◆	446.58■	**476.77**▲	**456.97**◆	**452.43**■
	Const-UCB:$C^{\text{lb,s.c.}}$ (ours)	**163.01**▲	**152.91**◆	**152.03**■	**454.22**▲	**446.93**◆	**444.77**■	483.54▲	458.35◆	454.27■
	Best Single Policy[2]	150.39	150.39	150.39	442.95	442.95	442.95	424.84	424.84	424.84
Cumul. Regret (mean)	UCB-Bandit (baseline)	736.7△	2138.7◇	3603.1□	3180.7△	6186.5◇	8495.4□	3389.6△	6978.6◇	10948.8□
	Const-UCB:$C^{\text{lb,opt}}$ (ours)	332.2△	760.7◇	1121.3□	2003.2△	5053.6◇	7309.7□	2374.3△	5848.0◇	9615.9□
	Const-UCB:$C^{\text{lb,wgt}}_{\text{p=0.5}}$ (ours)	274.9△	504.0◇	674.8□	222.1△	568.5◇	847.5□	**719.6**△	**1783.2**◇	**3599.2**□
	Const-UCB:$C^{\text{lb,s.c.}}$ (ours)	**249.8**△	**428.9**◇	**546.4**□	**74.5**△	**230.9**◇	**327.4**□	749.3△	1887.6◇	3737.3□

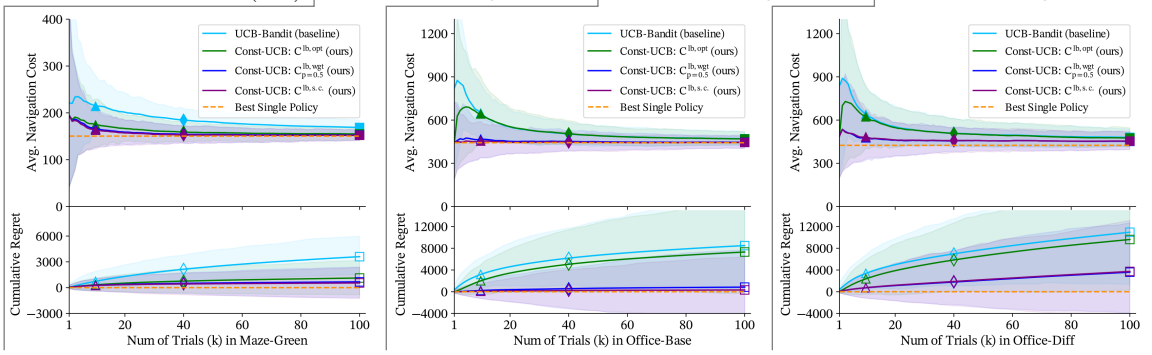


Figure 3.5: **Average Navigation Cost (mean) and Cumulative Regret (mean) for deployments in office-centric environments, Fig. 3.3(b).** Each deployment consists of 100 randomized navigation trials, each in a previously unseen map. Mean cost and regret are computed across 200 randomized deployments. For our approach Const-UCB, we show results with optimistic $C^{\text{lb,opt}}$, weighted $C^{\text{lb,wgt}}$ and simply connected $C^{\text{lb,s.c.}}$ lower bounds as discussed in Sec. 3.7.3. The solid lines denote the mean, and the shaded regions show 10^{th} to 90^{th} percentile. The symbols: triangle, diamond and square denote average cost (filled) and cumulative regret (unfilled) at 10^{th}, 40^{th} and 100^{th} trial respectively in both the table and the plot for each environment.

were simply-connected, our selection procedure using the optimistic lower bound $C^{\text{lb,opt}}$ outperforms baseline UCB bandit, demonstrating the utility of our approach.

3.8.2 Office-centric Results

We simulate navigation in procedurally-generated office-like environments consisting of randomly generated interconnected hallways with rooms meant to look like offices with furniture-like clutter. Each generated map is unique with randomized start and goal poses. Fig. 3.3 shows example offices from our visual simulator. We design and experiment in two variants of the office environments:

Table 3.2: Average navigation cost for each policy in office-centric environments without policy selection

	Maze-Green	Office-Base	Office-Diff
Non-learned	206.05	448.60	476.88
LSP-Maze-Green	<u>150.39</u>	1667.70	1759.76
LSP-Office-Base	258.76	<u>442.95</u>	641.88
LSP-Office-Diff	306.45	942.62	<u>424.84</u>

Office-Base The hallway walls are painted with gray while room walls are yellow.

Office-Diff The wall colors for hallways and rooms are swapped to look visually different from Office-Base.

Two LSP-based policies, LSP-Office-Base and LSP-Office-Diff, are trained in their corresponding environments following the procedure in Sec. 3.6.2. Similar to maze experiments, each LSP-based policy in office environments is trained in 500 randomly generated offices while deploying on a different set of maps. In addition to these two policies, we deploy an optimistic non-learned policy and the LSP-Maze-Green policy (see Sec. 3.8.1 for both), the latter of which is trained in Maze-Green, showing cross-environment policy selection and thus the flexibility of our approach. Deployment and statistics generation follow the same procedure as described for maze experiments (Sec. 3.8.1).

The results in Fig. 3.5 show that our constrained UCB (Const-UCB) bandit approach reduces the average navigation cost within fewer trials compared to the baseline UCB bandit approach in all of our experiments, more quickly converging closer to the costs of the best performing policy, and never under-performs the bandit. Consequently, our Const-UCB approach accumulates significantly lower mean cumulative regret compared to UCB bandit: 85% lower in Maze-Green (with $C^{\text{lb,s.c.}}$ bound), 96% lower in Office-Base (with $C^{\text{lb,s.c.}}$ bound) and 67% lower in Office-Diff (with $C^{\text{lb,wgt}}_{\text{p=0.5}}$ bound) owing to quickly ruling out poor performing policies.

Our office environment is not simply connected, and so only the optimistic lower bound $C^{\text{lb,opt}}$ maintains guarantees on long-term convergence; the least tight of the bounds, it

only non-trivially improves performance (i.e., constrains selection) when the office-trained policies are deployed in the maze environment. However, even some assumptions about the structure of unseen space are helpful for more strongly constraining selection. Though they may violate *theoretical* guarantees on asymptotic performance in general, we show that using the weighted lower bound cost $C^{\text{lb,wgt}}_{\text{p=0.5}}$, with a $p = 50\%$ likelihood a shorter alternative path exists, and the non-simply-connected assumptions result in improvements over the unconstrained UCB bandit in both office environments.

3.9 Contributions

We present a data-efficient policy selection approach that leverages learning over subgoals planning (LSP)-enabled offline alt-policy replay to compute a lower bound on the performance of policies based on the partial map and images collected from the environment during navigation, and use bandit-like method to identify the best-performing policy quickly. Our approach enables the learning-guided robot to reduce average navigation cost in a wide variety of partially-mapped environments by picking only those policies that are known to perform better or have the potential to do so and thereby significantly reducing the cumulative regret compared to the baseline UCB bandit.

Chapter 4: Multi-Strategy Deployment-Time Learning and Adaptation for Navigation under Uncertainty

4.1 Introduction

Building upon the central thesis of enabling autonomous robots to introspect and improve their deployment-time performance, this chapter extends the concept of introspection to scenarios where robots must learn and adapt during deployment. We investigate how a robot can reason about multiple concurrently evolving policies—each being learned or adapted online—and introspectively determine which policy has improved and should be deployed for effective performance in novel, unknown environments. This exploration advances the theme of self-improving robotic systems central to this dissertation. The work presented in this chapter has been published in Paudel et al. (2024).

Consider a robot deployed in an unknown environment and tasked to reach an unseen point goal. The robot leverages learning to make decisions about where to go to most quickly find the goal. However, depending upon how different the training and deployment environments are, learning may not always inform good behavior and the robot may demonstrate poor performance during deployment. Thus, to perform well in a wide variety of environments, the robot must improve its behavior *during deployment*, via strategies such as visual domain adaptation or online policy learning. Problematically, there is no *one-size-fits-all* approach to deployment-time learning or adaptation: each such strategy is typically only suitable for addressing a subset of the types of changes a robot may encounter during deployment or are slow to converge, risking poor performance during a potentially lengthy or error-prone learning phase and limiting most such approaches in general.

If robots in this goal-directed navigation scenario are to perform well when deployed in arbitrary unfamiliar environments, they could ideally run many strategies for adaptation

and online learning *in parallel* and rely on *run-time monitoring* to select the best-performing policy or strategy. Bandit algorithms (Lai et al., 1985; Thompson, 1933; Gittins, 1979) are potentially useful, yet are prohibitively slow to converge in this setting. Other existing work in the space of run-time monitoring (Rahman et al., 2021; Mallozzi et al., 2019; Zhou et al., 2019; Daftry et al., 2016; Liu et al., 2024) or policy selection for reinforcement learning (Lee et al., 2021; Reisinger et al., 2008; Ghosh and Chowdhury, 2022; Pacchiano et al., 2020) are similarly limited to short time horizons or fully-known environments and are not straightforwardly applicable. Moreover, such strategies typically presume that the policies are static, posing a risk when the policies improve during deployment: deciding to *not select* a policy that was deemed to perform poorly early on—yet has improved considerably since then—results in poor overall performance. Instead, we require an approach that can reliably select between a family of continually changing (*non-stationary*) policies being learned or adapted during deployment.

In this work, we present an approach for data-efficient and reliable policy selection capable of choosing the best-performing policies from an ensemble of policies, even when many of these policies are being learned or adapted during deployment (Fig. 4.1). We leverage insights from the recent work by Paudel and Stein (2023) whose *offline alt-policy replay* uses data collected by the robot from a navigation trial to simulate how another policy would have performed, yielding lower bounds on its performance that constrain and improve policy selection. As this approach does not consider that policies may change during deployment, the rolling estimates of the performance of each policy quickly diverge from their true expected performance as old performance estimates become stale. Instead, we leverage *offline alt-policy replay* to refresh old data and so compute up-to-date bounds on the would-be performance of each policy, allowing us to quickly select the best performing policy even as the robot's policies improve during deployment.

We deploy our approach in simulated maze-like environments where visual cues indicate promising routes to the unseen goal. Via our approach, our robot is deployed with six

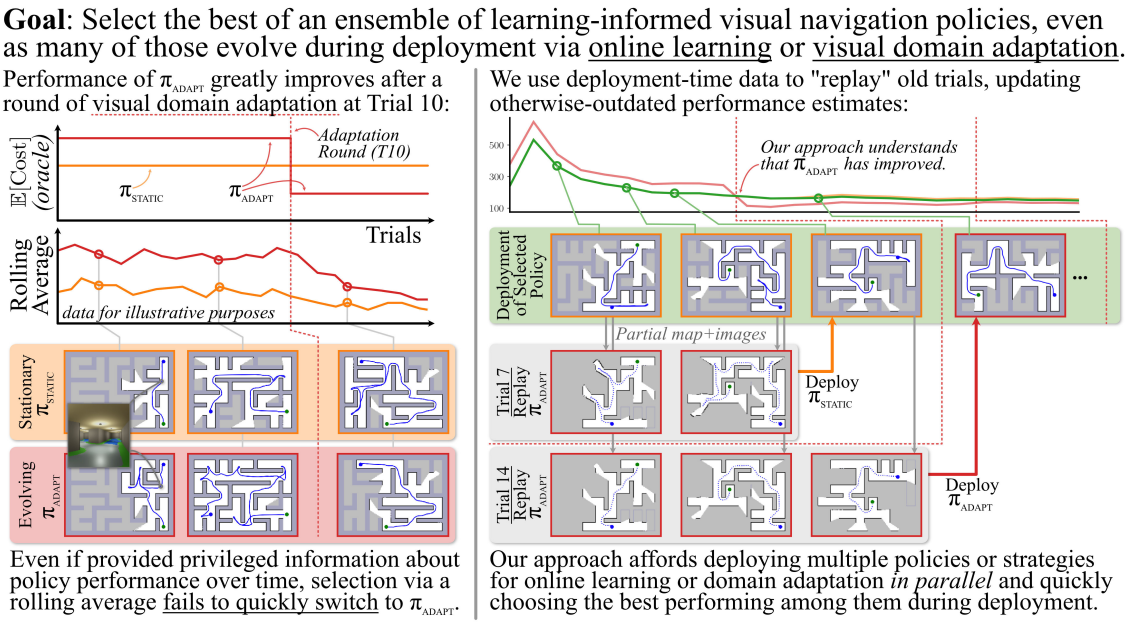


Figure 4.1: Overview of our approach for fast selection between non-stationary policies.

policies—a static pre-trained policy, a non-learned baseline policy, two policies being continually adapted via visual domain adaptation, and two policies being trained from scratch using deployment-time data—and quickly chooses the best performing ones among these when deployed. Notably, our approach correctly avoids the non-stationary policies early in deployment before they learn to understand the environment yet quickly switches to them once they improve, a capability that allows our approach to outperform all single-policy strategies and existing policy selection baselines.

4.2 Related Work

Domain Adaptation Advances in visual domain adaptation (Ganin et al., 2016; Long et al., 2015; Bousmalis et al., 2017; Hoffman et al., 2018; Taigman et al., 2017; Zhu et al., 2017a) have made it possible to adapt systems trained in one domain to a different domain during deployment. Visual domain adaptation approaches have been used to compensate

for differences in the robot's visual observations between the training and deployment environments (Bousmalis et al., 2018; Stein and Roy, 2018; Wulfmeier et al., 2017; Palazzo et al., 2020; Bradley and Bagnell, 2009). However, visual domain adaptation approaches, such as those that leverage CycleGAN (Zhu et al., 2017a), are not always suitable for improving performance, especially when domains are drastically dissimilar, requiring careful consideration for their uses on robotic systems where reliability is of critical importance.

Deployment-Time Adaptation The idea of adapting robots during deployment has been explored widely in the robotics literature. Many approaches focus on short-horizon robot behavior such as adapting low-level motor controls or locomotion in diverse terrains (Kumar et al., 2021; Zhu et al., 2023; Chen et al., 2023; Smith et al., 2022; Hansen et al., 2021; Nagabandi et al., 2019). As such, they are not concerned with long-horizon behavior where the robot should consider the impacts of its immediate actions far into the future, the focus of this work.

Runtime Monitoring Runtime monitoring approaches aim to evaluate the reliability of learned models used to guide the robot's behavior during deployment (Rahman et al., 2021; Mallozzi et al., 2019; Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017; Liu et al., 2024) and decide whether a fallback policy should instead be used. However, these approaches focus primarily on evaluating certain subsystems (e.g., the perception system) (Mallozzi et al., 2019; Liu et al., 2024) or scrutinizing a learned model's raw output instead of the robot's task performance (Zhou et al., 2019; Daftry et al., 2016; Richter and Roy, 2017), thus limiting their applicability for evaluating the robot's long-horizon behavior for navigation in partially-mapped environments.

Policy Selection can be thought of as model selection applied to choosing between robot behaviors (Paudel and Stein, 2023). In reinforcement learning, model selection (Lee et al., 2021; Reisinger et al., 2008; Ghosh and Chowdhury, 2022; Pacchiano et al., 2020) is often treated as a generalized form of multi-armed bandit problem (Sutton and Barto, 2018). Using bandit algorithms (Lai et al., 1985; Thompson, 1933; Gittins, 1979)—which are often black-box—for policy selection requires trade off between exploitation and exploration and

hence the robot has to potentially go through multiple trials with poor behavior before a better policy can be identified. White-box policy selection (Paudel and Stein, 2023) has shown recent promise in data efficiency, yet so far only consider stationary policy selection, ill-suited for policies that evolve during deployment.

Adaptation with Meta-Reinforcement Learning Many problems in robotics have leveraged meta-reinforcement learning (meta-RL) approaches for learning robot policies that can be adapted to new tasks or scenarios (Nagabandi et al., 2019; Arndt et al., 2020; Ajay et al., 2022; Schoettler et al., 2020; Shin et al., 2022). While meta-RL is most effective when provided access during training to a distribution over environments to which the deployment-time environments may belong, our work is geared more towards reliably identifying and deploying existing general-purpose online training and domain adaptation tools meant to handle systematic differences between training and deployment-time environments. As such, our approach allows for integrating a wide range of existing learning/adaptation strategies as demonstrated in our experiments with a mix of different policies—something that meta-RL is not geared towards. Additionally, our approach makes no assumptions about similarities between training and deployment environments while still benefiting from asymptotic performance guarantees.

4.3 Preliminaries

For robots to perform well across a variety of unfamiliar environments, they must have the ability to simultaneously apply multiple deployment-time learning and domain adaptation techniques in parallel, so that they may choose the best-performing of them when deployed. We model this scenario as an instance of *policy selection*. While the policies themselves are *non-stationary*—they change over time as the robot is deployed and collects data—we must first discuss the fundamentals of goal-directed navigation in partially-mapped environments and strategies for *stationary* (static) policy selection in this domain, needed to understand our approach.

Goal-Directed Navigation in Partially-Mapped Environments For each trial, our robot is placed in a partially-mapped environment and tasked to reach an unseen point-goal in minimum expected cost, measured in units of distance. Planning under uncertainty is often formulated as a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998). As planning via the POMDP model directly is computationally intractable, many planning strategies in this domain rely on learning to anticipate what may lie in unseen space and thus inform good behavior. The robot's policy π consumes a belief state b_t—including the robot pose q_t, the partial map m_t, and visual observations collected onboard the robot o_t—and returns a primitive action a_t specifying the robot's behavior. For all policies in this work, images are used to inform learning, which makes predictions about unseen space to guide the robot more quickly to the unseen goal.

Policy Selection over Multi-Trial Deployments A single *deployment* consists of a sequence of trials, each a single traversal from start to goal in a previously-unseen map. We consider the scenario in which the robot is deployed with an ensemble of policies $\Pi = \{\pi_1, \pi_2, \cdots, \pi_N\}$, each defining a different behavior. The objective of policy selection is to determine, during a deployment, which of those policies performs best in the deployment-time environments:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}} \, \mathbb{E}[C(\pi)] \ , \tag{4.1}$$

where $\mathbb{E}[C(\pi)]$ is the expected cost of policy π in the deployment environment distribution. In practice, the expected cost is approximated by deploying the policy multiple times and averaging the performance: $\mathbb{E}[C(\pi)] \approx \bar{C}(\pi)$, the average cost of the trials in which policy π was deployed.

4.3.1 Black-box Policy Selection via Multi-Armed Bandits

Selecting the policy that minimizes $\bar{C}(\pi)$ is unwise in general, since the randomness inherent in the deployments means that the best-performing policy may be incorrectly ruled out early on and never selected again to improve the estimate of $\mathbb{E}[C(\pi)]$ with no recourse to

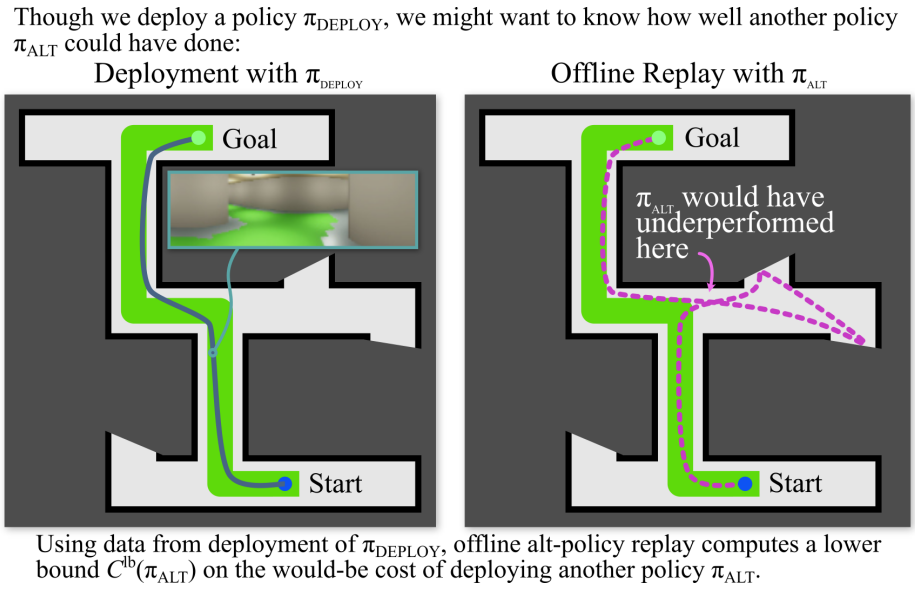


Figure 4.2: Overview of Offline Replay

recover. Instead, the UCB multi-armed bandit (Lai et al., 1985) balances exploitation with exploration, incentivizing selection of policies that have been chosen less often to improve the estimate of $\mathbb{E}[C(\pi)]$. For trial $k+1$, UCB bandit selection specifies one choose policy $\pi^{(k+1)}$ according to

$$\pi^{(k+1)} = \underset{\pi \in \Pi}{\operatorname{argmin}} \left[\bar{C}_k(\pi) - c\sqrt{\frac{\ln k}{n_k(\pi)}} \right], \tag{4.2}$$

where $\bar{C}_k(\pi)$ is the average cost over trials 1-through-k in which policy π was selected, $n_k(\pi)$ is the number of times policy π was selected through trial k, and $c > 0$ is a parameter controlling the rate of exploration. In practice, each trial of goal-directed navigation is expensive and the resulting cost samples have high variance. Thus, black-box policy selection approaches such as this are problematically slow to converge, making them impractical for policy selection in this domain despite their desirable guarantees on asymptotic sub-linear regret.

4.3.2 Data-Efficient Policy Selection via Offline Alt-Policy Replay

A central challenge of black-box model selection approaches is their data inefficiency, as each deployment yields only a single data point for a single policy. If it were instead possible to use the images and partial map collected during a trial to determine how well other policies could have performed—a bound on performance—we could use it to inform policy selection and so accelerate convergence to the best-performing policy. Paudel and Stein (2023) present an approach they term *offline alt-policy replay* in which data collected during deployment of policy π is used to simulate the would-be behavior of another policy π' placed in the same environment.

Under this approach, deployment of policy π during trial k yields a *record* $\mathcal{Z}_k$ of all its poses and observations of the environment. This partial snapshot of the environment is used to simulate how another policy π' would have acted, letting it navigate and reveal the environment and receive the image observations it needs to make predictions about unseen space that inform its behavior. Offline replay produces a lower bound on the performance of π' had it instead been deployed: $C_k^{\text{lb}}(\pi') = \textsc{OfflineReplay}(\pi', \mathcal{Z}_k) \leq C_k(\pi')$. Thus, each trial yields information about the performance of *all policies* $\pi \in \Pi$. The average lower bound after k trials, $\bar{C}_k^{\text{lb}}(\pi')$, constrains bandit-like policy selection:

$$\pi^{(\text{k}+1)} = \underset{\pi \in \Pi}{\operatorname{argmin}} \left[\max \left(\bar{C}_k^{\text{lb}}(\pi), \bar{C}_k(\pi) - c\sqrt{\frac{\ln k}{n_k(\pi)}} \right) \right] \tag{4.3}$$

Offline alt-policy replay results in dramatically faster convergence to the best-performing policy while still preserving guarantees on asymptotic sub-linear regret. However, like the bandit algorithm before it, this approach presumes that the policies themselves are stationary, and so is not well-suited to select between policies being learned or adapted online, the focus of this work.

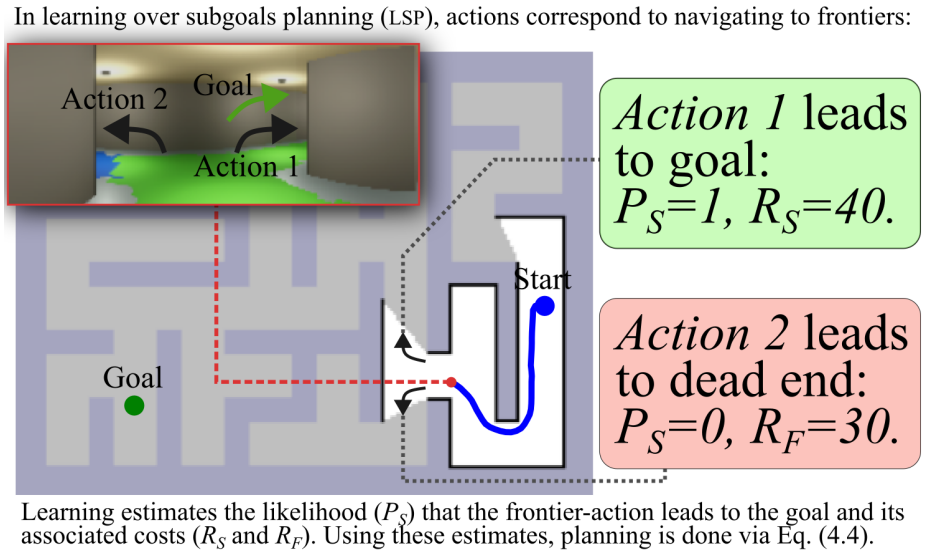


Figure 4.3: Overview of learning over subgoals planning (LSP).

4.3.3 Learning over Subgoals Planning: A Learning-Augmented Model-Based Planning Abstraction

A key enabler of offline alt-policy reply is the requirement that planning be done via a policy amenable to counterfactual reasoning—i.e., *What would policy π' have done if it had instead been placed in this situation?*—with which the behavior of policy π' can be simulated. As such, our policies are based on the learning-informed model-based learning over subgoals planning (LSP) by Stein et al. (2018), which satisfies this requirement.

Learning over subgoals planning (LSP) is a high-level planning framework for learning-informed model-based navigation in a partially-mapped environments. Under this abstraction, *subgoals* correspond to frontiers, each a boundary between free and unknown space; high-level actions a_t correspond to navigation to a subgoal and then exploration beyond to attempt to reach the unseen goal. The robot uses a learned model to make predictions about unseen space beyond each subgoal, information it uses to decide where to reveal next. The learned *subgoal property estimator*, a neural network $\mathcal{N}_\theta$ parameterized by θ and trained via supervised learning, consumes panoramic images in the vicinity of each subgoal to estimate the likelihood that each subgoal-action will successfully reach the goal $P_{S,\theta}$ and

the expected costs associated with success $R_{S,\theta}$ and failure $R_{F,\theta}$ to reach the goal in unseen space. Upon failing to reach the goal via a high-level action a_t, the robot must select another action $a \in \mathcal{A} \setminus a_t$. Planning is model-based, with the expected cost of a high-level action defined by a Bellman Equation:

$$Q_\theta(b_t, a_t) = D(b_t, a_t) + P_{S,\theta}(a_t)R_{S,\theta}(a_t) + (1 - P_{S,\theta}(a_t)) \left[R_{F,\theta}(a_t) + \min_{a \in \mathcal{A}(b_t) \setminus a_t} Q_\theta(\tilde{b}'_t, a) \right] \tag{4.4}$$

where b_t is the robot's belief, $D(b_t, a_t)$ is the travel cost to reach the subgoal a_t via known space, and $\tilde{b}'_t$ is the approximate updated belief, which reflects that the robot has moved to the subgoal a_t. Planning seeks to find the action a_t that minimizes expected cost: $\pi_\theta(b_t) = \text{argmin}_a \, Q_\theta(b_t, a)$; the robot replans whenever the map is updated, proceeding until the goal is reached.

As our approach builds upon the insights of offline replay, all navigation policies in this work rely on the learning over subgoals planning (LSP) abstraction. Planning for each is done via Eq. (4.4), and so policy selection in this context can be thought of as choosing the network model $\mathcal{N}_\theta$ whose predictions about unseen space $\{P_{S,\theta}, R_{S,\theta}, R_{F,\theta}\}$ result in the best performance when deployed.

4.4 Multi-Strategy Deployment-Time Learning and Adaptation

4.4.1 Problem Formulation: Non-stationary Policy Selection

We seek to achieve minimum-expected-cost performance for navigation in partially-revealed environments. Our robot is deployed with an ensemble of learning-informed policies—many of which are being learned or adapted after each trial—and seeks to pick the best performing strategy during deployment. We formulate this problem as an instance of *non-stationary policy selection*. A deployment is a sequence of T trials, each a navigation from start to

goal in a previously unseen map. As many of the policies are learned or adapted and so evolve over time, we add an additional subscript k to denote the policy after trial k. Thus, before trial $k+1$, the robot has access to N policies $\Pi_k = \{\pi_{1,k}, \pi_{2,k}, \ldots, \pi_{N,k}\}$ and seeks to pick the one that minimizes expected cost via

$$\pi^{(k+1)} = \underset{\pi_{n,k} \in \Pi_k}{\operatorname{argmin}} \mathbb{E}[C(\pi_{n,k})] \ . \tag{4.5}$$

Though Eq. (4.5) resembles Eq. (4.1), computing the expected cost of policies in Eq. (4.5) is challenging because the policies themselves are continually being updated via learning or adaptation during deployment. The policy selection strategies discussed in Section 4.3 (Eq. (4.2) & (4.3)) use a rolling average to estimate $\mathbb{E}[C(\pi_{n,k})]$, which quickly diverges from the true estimate for policies that improve via deployment-time training or adaptation, reducing the robot's performance. Instead, if selection is to quickly and reliably converge to the best performing policy, there is a need for an approach that can compute accurate bounds on the performance of each policy $\pi_{n,k}$ even as they improve.

4.4.2 Approach: Selection over Non-Stationary Policies being Continuously Learned or Adapted During Deployment

We present an approach that performs data efficient policy selection over a set of policies that are continually learned or adapted during deployment. Our policy selection approach chooses policies based on the selection strategy of Eq. (4.3), yet rather than using *rolling averages* of the lower-bound costs $\bar{C}_k^{lb}$ and the deployment costs $\bar{C}_k$ that fail to consider the evolving nature of the robot's policies, we instead rely upon OfflineReplay to refresh the estimates of the robot's performance.

Computing Up-to-Date Bounds on Performance Whenever one of the robot's policies is updated, via domain adaptation or learning with data it collects during deployment, its behavior may have changed. Thus, $C_k(\pi_{n,k}) \neq C_k(\pi_{n,k-1}) \neq \cdots \neq C_k(\pi_{n,1})$ in general and so using a rolling average of performance will result in poor selection performance and miss

out on choosing policies that may have dramatically improved. Just as offline alt-policy replay can be used to determine the would-be performance of alternate policies after each trial, we can leverage this approach to revisit *old trials* to determine how well the robot's updated policies would have performed. We use the data the robot collects until trial k (the records $\{\mathcal{Z}_i\}_{i=1,\cdots,k}$) to replay how each of the robot's updated policies would have behaved in older trials, letting us get a much more accurate estimate of that policy's expected performance and performance bounds. For each updated policy $\pi_{\mathrm{n,k}} \in \Pi_k$, the updated lower bound performance is computed by replaying its behavior for all trials:

$$\bar{C}_k^{\mathrm{lb}}(\pi_{\mathrm{n,k}}) \approx \frac{1}{k} \sum_{i=1}^{k} \textsc{OfflineReplay}(\pi_{\mathrm{n,k}}, \mathcal{Z}_i) \tag{4.6}$$

The average cost of deployed policies $\bar{C}_k(\pi_{\mathrm{n,k}})$ is recomputed similarly. We note that our problem setting envisions that robots will not be in constant operation and can perform much of this computation for training or re-evaluation while idle and waiting for its next navigation objective.

Policy Selection using Updated Performance Estimates We use the selection approach of Eq. (4.3) to perform policy selection, yet use our *updated* estimates of the performance of the robot's policies—lower-bound costs $\bar{C}_k^{\mathrm{lb}}$ and the deployment costs $\bar{C}_k$ computed via Eq. (4.6)—in place of their rolling averages. After each trial k, the robot initiates a procedure to learn or adapt its non-stationary policies using the data it has so far collected $\{\mathcal{Z}_i\}_{i=1,\cdots,k}$. Before trial $k+1$, it computes updated performance estimates for each of its updated policies $\pi_{\mathrm{n,k}} \in \Pi_k$ via Eq. (4.6) and selects the policy to deploy via Eq. (4.3). Since data from new trials are continually added to record $\mathcal{Z}$ for training and offline replay, these policies will asymptotically converge to static policies after many trials, and we preserve bandit-like guarantees on sub-linear asymptotic regret.

4.4.3 Deploying an Ensemble of Policies

Facilitated by our approach, we deploy our robot with an ensemble of policies; some are unchanging (e.g., the robot's pre-trained policy) and others evolve and improve during deployment as more data is collected—e.g., those that rely on visual domain adaptation or are trained from scratch. Policies in the ensemble are chosen so as to capture potentially distinct scenarios during deployment. Each such policy depends on the learning over subgoals planning (LSP) approach discussed in Section 4.3.3, and so policy selection and domain adaptation in this context can be thought of as simultaneously improving and choosing between feed-forward models that make predictions about unseen space. Here, we discuss the different strategies employed by each policy our robot is deployed with.

The Pre-Trained Learning-Informed Policy [Stationary] π_{PreTrain} The robot is equipped with a static policy trained in advance of deployment, a LSP policy well-suited to good performance in the training environments. If we were to know in advance that the deployment-time environments matched the training environments, we would expect this policy to perform well and so no policy selection would be necessary. However, this work considers the more general case wherein the deployment environments may differ non-trivially compared to those seen during training, leading to poor performance when relying on π_{PreTrain}.

The Non-Learned Optimistic Policy [Stationary] $\pi_{\text{NonLearned}}$ We also deploy the robot with an *optimistic* planning strategy: a common non-learned strategy in which unseen space is assumed to be unoccupied. The non-learned optimistic policy can be thought of as a special case of LSP, in which $P_S \equiv 1$ and $R_S, R_F \equiv 0$, so that the robot simply selects the shortest path through the partial map to the unseen goal, replanning when necessary as unseen space is revealed. This strategy of optimism under uncertainty is unlikely to outperform π_{PreTrain} when the deployment environments match those seen during training. However, when deployed in an unfamiliar environment that differs significantly from the training environment, the learned model underpinning π_{PreTrain} may be misled by visual features it cannot properly understand, making $\pi_{\text{NonLearned}}$ a reasonable backup strategy.

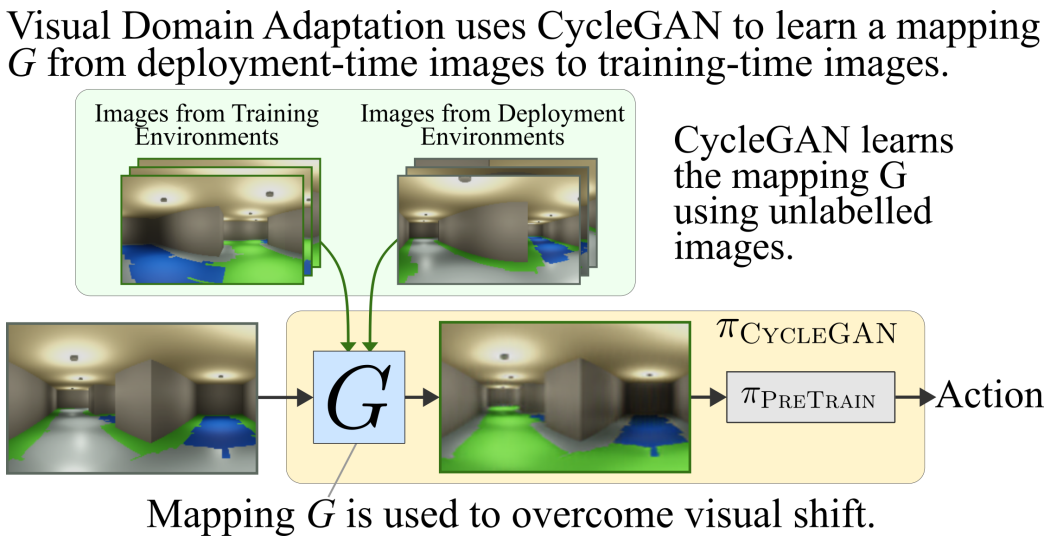


Figure 4.4: Adaptation of a pre-trained policy using visual domain adaptation.

Visual Domain Adaptation via CycleGAN [Non-Stationary] π_{CycleGAN} Our robot will use images it collects during deployment to perform *visual domain adaptation* via the CycleGAN algorithm (Zhu et al., 2017a; Stein and Roy, 2018). Using unlabeled images from the training and deployment environments, CycleGAN learns a mapping from one to the other, as shown in Fig. 4.4. This mapping is used as a preprocessing step: deployment-time images are made to look like training-time images and then fed to the robot's pre-trained policy π_{PreTrain}. The resulting policy π_{CycleGAN} can thus compensate for visual dissimilarities between training and deployment, and so can improve performance when such changes are what limits the robot's behavior.

However, visual domain adaptation *cannot* perform well in all deployment environments, something policy selection must judge during deployment. For our experiments, we train two models, one for 50 epochs $\pi_{\text{CycleGAN}_{50}}$ and one for 100 epochs $\pi_{\text{CycleGAN}_{100}}$, and allow policy selection to choose between them. Additional details of the CycleGAN approach and training can be found in Section 4.4.4.

Training from Scratch during Deployment [Non-Stationary] π_{Scratch} Using data it collects during deployment, the robot can additionally train an LSP-style policy *from scratch*

while deployed. For LSP, the robot's learned model estimates for each subgoal (associated with a boundary between free and unseen space that may lead to the unseen goal) the likelihood P_S that the subgoal successfully leads to the goal and the expected costs associated with reaching the goal R_S or needing to explore and turn back R_F. As the robot navigates, it can label image data using the partial map to train an LSP model: routes the robot discovered to reach the goal correspond to $P_S = 1$ and dead-ends to $P_S = 0$. Ambiguous routes—potential routes to the goal the robot did not explore during deployment—can also be included in the training data under either an *optimistic* prior, i.e., where all ambiguous routes are labeled with $P_S = 1$, or a *conservative* prior, i.e., an assumption that unexplored space is simply-connected, so that $P_S = 0$. During deployment, we train one such policy for each of the optimistic prior $\pi_{\text{ScratchOpt}}$ and conservative prior $\pi_{\text{ScratchCon}}$. While these policies will be invariably slow-to-converge towards good performance, they have the potential to successfully improve performance even in environments where visual domain adaptation is not well-suited. We use 5-fold cross validation (see Section 4.4.4) to determine an unbiased estimate of the updated lower bound on performance and deploy a policy trained on all trials.

4.4.4 Additional Implementation Details

CycleGAN Implementation and Training

We use the official PyTorch implementation of CycleGAN provided by Zhu et al. (2017a) on GitHub[1]. As mentioned in Section 4.4.3, we train two models: one for 50 epoch and another for 100 epochs. We use the default parameters for training except that we use a batch size of 8 and no learning rate scheduling is done. We use learning rate of 0.0002 with Adam optimizer. The images are of size 512 × 128 and we perform default resizing and cropping as a preprocessing step. 1261 images from 10 distinct maps in Maze-Green (where π_{PreTrain} is trained) are set aside as target domain images, and we sample 1300 images from the source domain (either Maze-Gray or Maze-Blue) collected during deployment for

[1] `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`

training the CycleGAN model. Using only 1300 images from deployment-time environments additionally helps to get an unbiased estimate of performance when replaying CycleGAN-adapted policy π_{CycleGAN} on older trials from which the images were collected.

Learning over Subgoals Planning: Subgoal Property Estimator Network Implementation and Training

As discussed in Section 4.3.3, all learning-informed policies rely on the learning over subgoals planning (LSP) abstraction for planning, a model-based approach that relies on a learned model to estimate *subgoal properties*: statistics of unseen space associated with each of the robots temporally-extended high-level actions to explore unseen space. The subgoal property estimator network $\mathcal{N}_\theta$ corresponding to π_{PreTrain} is trained with data-collected in 500 distinct maps in Maze-Green where the robot navigates using the $\pi_{\text{NonLearned}}$ policy. Data labelling procedure is similar to the one described for π_{Scratch} (Sec. 4.4.3) except that at training time the underlying map is known and so can be used to generate ground truth labels for subgoal properties P_S, R_S and R_F corresponding to all subgoals.

The subgoal property estimator network $\mathcal{N}_\theta$ is trained via supervised learning using the data collected during an offline training phase. Our neural network architecture and training procedure resemble that of Paudel and Stein (2023). The network takes as input a 512×128 panoramic image centered on a subgoal, relative distance to the subgoal and relative distance to the goal. The image is encoded by passing through 4 convolutional layers and then concatenated with features corresponding to relative distances to subgoal and goal after which the concatenated features are passed through 9 convolutional layers and finally 5 fully connected layers to output 3 subgoal properties P_S, R_S and R_F. We use a learning rate of 0.002 with a decay factor of 0.5 every epoch and train for 8 epochs with Adam optimizer. We use cross-entropy loss for learning logits associated with P_S and L2 loss for learning R_S and R_F. The deployment-time training of subgoal property estimators for π_{Scratch} follows a similar architecture and procedure.

Offline Alt-Policy Replay Details

As discussed in Sec. 4.3.2, we use offline alt-policy replay to replay the behavior of a policy without deploying it. To replay the behavior of a policy π', we leverage the record $\mathcal{Z}_k$ collected during trial k under a deployed policy π. At every time step during offline alt-policy replay, the robot leverages the final partial map m_{final} observed in trial k to simulate the laser scan and updates its observed map as the robot moves. The frontiers—boundaries between free and unknown space—revealed in the observed map corresponds to the subgoal-actions that the robot can take to explore the region. To get the robot-view panoramic image corresponding to a subgoal-action, we retrieve existing image in record $\mathcal{Z}_k$ that is closest to and in line of sight to the subgoal and recenter it at the subgoal. This image is used to estimate the subgoal properties P_S, R_S and R_F using a neural network corresponding to π' which is then used to compute the next high-level action using Eq. (4.4). The robot then simulates the low-level motion primitive to move towards the selected subgoal and this process is repeated. At any point during simulated navigation, if the robot attempts to enter a region that is unknown in the final partial map m_{final} via a frontier, we mask that frontier and force the robot to pick a different subgoal-action. The net distance travelled to reach the goal via this procedure is the replay cost of policy π'.

Cross Validation for Reevaluating Older Trials

As mentioned in Sec. 4.4.3, we use 5-fold cross validation to get an unbiased estimate of the performance of updated policies. Since our policies trained during deployment from scratch (π_{SCRATCH}) are based on the same data in record $\mathcal{Z}$ that is also used for offline alt-policy replay to reevaluate older trials after updating the policies, such cross-validation approach overcomes the risk of overestimation of performance during replay due to data leak. With 5-fold cross validation, 5 policies are trained, each on the data from four-fifth of older trials, and replayed on the remaining one-fifth of the trials to get the revised performance estimates for all older trials. Finally, a new policy is trained on data from all previous trials and made available for the robot to choose from in the next trial.

4.5 Experimental Results

We demonstrate the effectiveness of our approach in simulated maze-like environments, procedurally generated so that each navigation trial sees a unique map. Each deployment consists of 50 navigation trials, each a traversal from start to goal through a previously-unseen map. As described in Section 4.4.3, the robot has an ensemble of six policies to choose from during deployment: $\Pi = \{\pi_{\text{NonLearned}}, \pi_{\text{PreTrain}}, \pi_{\text{CycleGAN}_{50}}, \pi_{\text{CycleGAN}_{100}}, \pi_{\text{ScratchOpt}}, \pi_{\text{ScratchCon}}\}$; to save computation, the non-stationary policies are updated via learning or adaptation only every 10 trials. In addition to our approach for non-stationary policy selection NonStationaryReplay, we include results with ucbBandit and RollingReplay (Paudel and Stein, 2023) policy selection, which performs selection via Eq. (4.3). We compare our NonStationaryReplay approach with ucbBandit and RollingReplay baselines both of which assume that the policies are stationary. While it would have been more suitable to compare our approach with non-stationary policy selection approaches, existing potential baselines in this area either assume policies are stationary or leverage knowledge we do not have access to in the context of robot navigation, e.g. underlying distribution of a policy's navigation cost. As such, ours is the first to propose such non-stationary policy selection algorithm for robot navigation under uncertainty, and hence our baselines only include the most suitable stationary policy selection approaches from the literature.

We deploy in three maze variants, each with differing (often conflicting) visual cues that signal routes to the unseen goal as shown in Fig. 4.5. Results (Fig. 4.6) show average performance and cumulative regret of each selection strategy over time. We additionally show (Fig. 4.6 bottom) our NonStationaryReplay's estimate of the cost lower bound $\bar{C}^{\text{lb}}$ over time, lending insight into its selection process.

Deployment in Maze-Green: *A green path signals routes to the goal, the floor is gray, and a blue path leads to dead-ends.* The stationary π_{PreTrain} is trained offline in held-out maps from this environment and so is the best. While the ucbBandit converges slowly, both

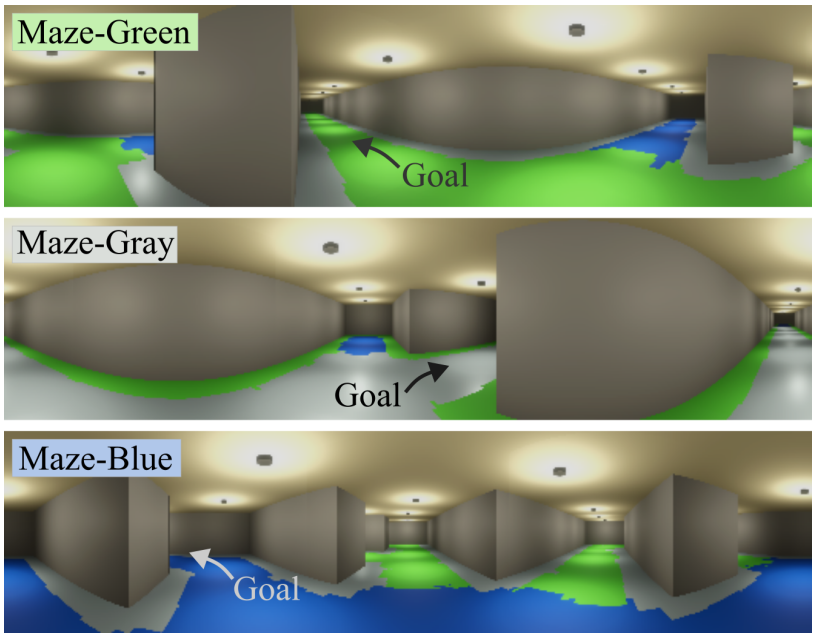


Figure 4.5: The Simulated Maze Environments

NonStationaryReplay and RollingReplay expectedly select π_{PreTrain} quickly, achieving near-optimal performance.

Deployment in Maze-Gray: *A gray path signals routes to goal, the floor is green, and the blue path leads to dead ends.* Misled by the green flooring, π_{PreTrain} performs poorly and so both NonStationaryReplay and RollingReplay, initially select the non-learned $\pi_{\text{NonLearned}}$. However, only our approach considers that policies may evolve during deployment. After 10 trials, the π_{CycleGAN} policies improve via visual domain adaptation and our approach quickly switches to them, achieving 57% lower cumulative regret compared to RollingReplay. Notably, selecting π_{CycleGAN} before it improves would *reduce* performance; thus, our NonStationaryReplay *outperforms all single-policy selection strategies.*

Deployment in Maze-Blue: *A blue path on the ground signals routes to goal, the floor is gray, and the green path leads to dead ends.* π_{CycleGAN} cannot resolve the visual dissimilarities between training and deployment even after many trials, and so the policies trained from scratch π_{Scratch} are the most promising approaches despite their slow convergence. After 30 trials, $\pi_{\text{ScratchCon}}$ improves sufficiently and our NonStationaryReplay selects it, resulting

	Deployment Environment	UCB Bandit	Rolling Replay	NonStationary Replay (ours)	Our Improvement vs. UCB	Our Improvement vs. Rolling
Avg. Cost	Maze-Green	187.9	**161.6**	**161.6**	14.0%	0.0%
	Maze-Gray	183.7	189.3	**165.0**	10.2%	12.8%
	Maze-Blue	243.5	202.3	**196.5**	19.3%	2.9%
Cumul. Regret	Maze-Green	1375.2	**61.2**	**61.2**	95.6%	0.0%
	Maze-Gray	1848.6	2131.2	**916.2**	50.4%	57.0%
	Maze-Blue	2955.6	898.2	**606.6**	79.5%	32.5%

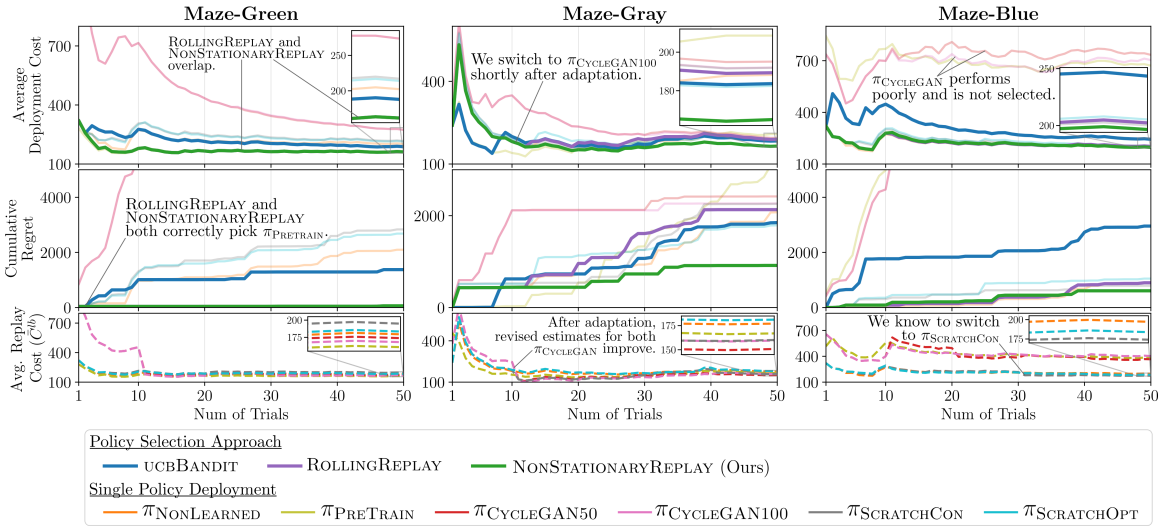


Figure 4.6: Our NonStationaryReplay policy selection approach outperforms both ucbBandit and RollingReplay (Paudel and Stein, 2023) baselines in both Average Navigation Cost and Cumulative Regret.

in 32% lower regret compared to RollingReplay, a number that would continue to grow with further trials.

4.5.1 Discussion of Trends in Results

Adaptation of π_{CycleGAN} in Maze-Gray and Maze-Blue As is mentioned in Section 4.5, Maze-Blue has blue path signaling routes to goal and green path leading to dead ends, and is deliberately designed to be indistinguishable from Maze-Green based only on visual observations in absence of the information about goal location. To clarify, CycleGAN does learn a reasonable visual mapping between the two environments: it learns an approximate identity mapping (see images corresponding to Maze-Blue in Fig. 4.7) from the visual

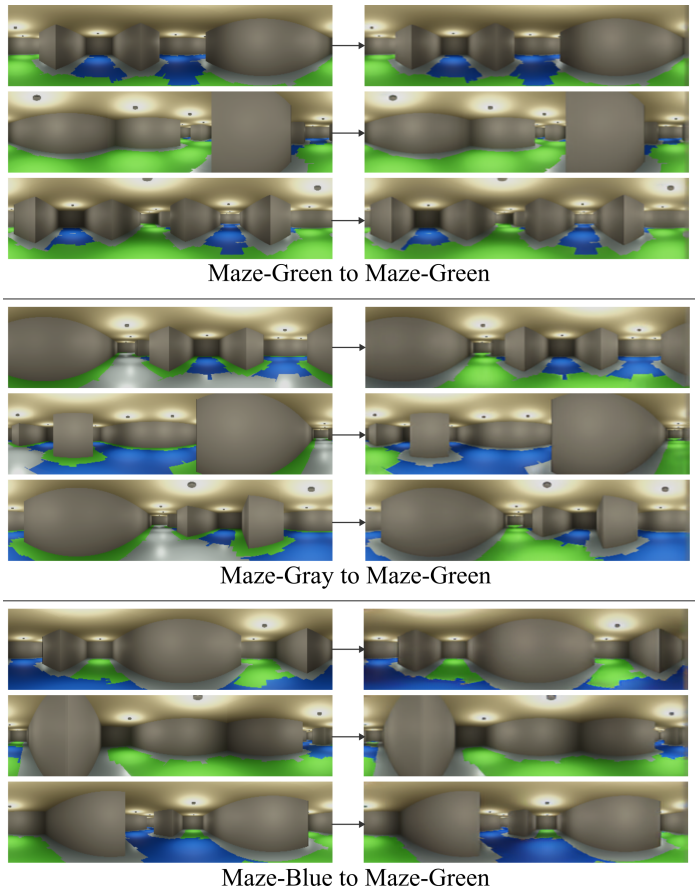


Figure 4.7: Images transformed from deployment-time environments (input) to look like training-time environments (output) with CycleGAN-based visual domain adaptation. The CycleGAN models trained after 40th trial are used to generate the images.

observations from each. However, we note that this mapping is not helpful for resolving the best path to the goal since the robot is still drawn to follow the green path, which no longer leads in the direction of the unseen goal. Hence, the performance of π_{CycleGAN} is poor in Maze-Blue and say that "visual domain adaptation fails" to properly adapt. On the other hand, in Maze-Gray, the color of the floor and goal-routes are swapped compared to Maze-Green. CycleGAN learns a mapping between the two that "swaps back" the colors of floor and path, a mapping that then allows the robot to correctly identify the best path to the goal, resulting in improved performance of π_{CycleGAN} in Maze-Gray.

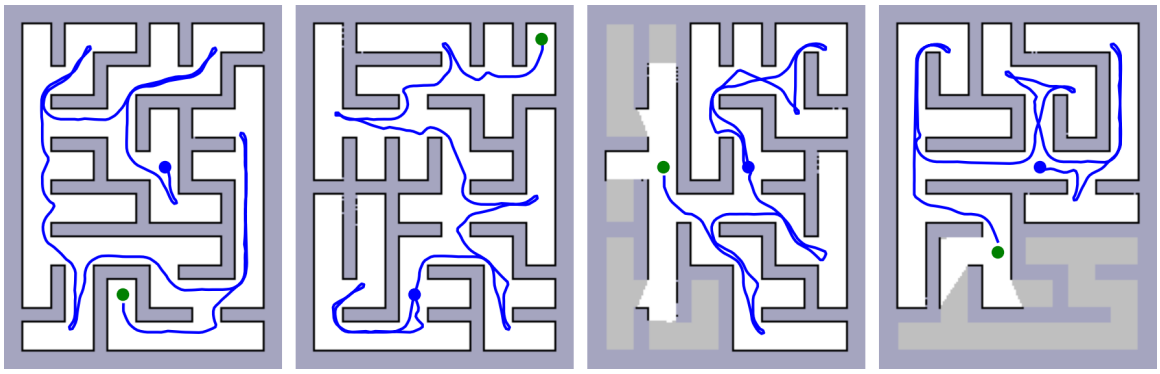


Figure 4.8: Sample Trajectories for π_{PRETRAIN} in Maze-Blue

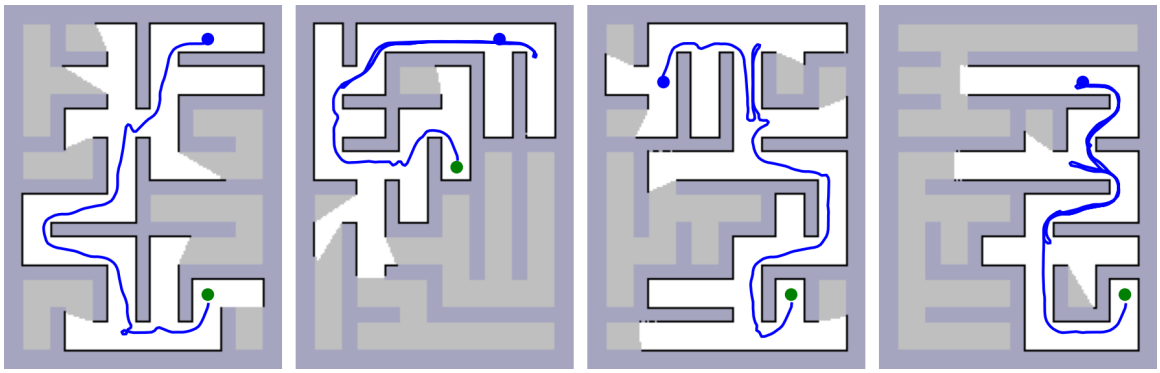


Figure 4.9: Sample Trajectories for π_{PRETRAIN} in Maze-Gray

Performance Differences of π_{PreTrain} in Maze-Blue and Maze-Gray The π_{PRETRAIN} policy trained in Maze-Green would have learned to follow the green path to find the goal and avoid the blue path to dead ends. As such, deploying it in Maze-Blue where green path leads to dead ends and blue path leads to goal (as illustrated in the respective environment's figures in Section 4.5) would often mislead π_{PRETRAIN} to navigate towards the dead ends increasing the cost to find the goal (see example trajectories in these environments as shown in Figs. 4.8 and 4.9). In Maze-Gray, this phenomenon is less severe since the dead ends are still blue and are often avoided by π_{PRETRAIN}.

Performance Similarities of π_{PreTrain} and π_{CycleGAN} in Maze-Blue As discussed in the aforementioned paragraph, the visual observations from Maze-Blue and Maze-Green

Table 4.1: Results of ablation with removing best learning/adaptation strategies corresponding to Maze-Gray and Maze-Blue

	Environment/ Ablation	UCB Bandit	Rolling Replay	NonStationary Replay (ours)	Our Improvement vs. UCB	Our Improvement vs. Rolling
Average Cost	Maze-Gray (w/o π_{CycleGAN})	190.8	197.1	**176.3**	7.6%	10.56%
	Maze-Blue (w/o π_{Scratch})	234.6	**202.3**	**202.3**	13.8%	0.0%
Cumul. Regret	Maze-Gray (w/o π_{CycleGAN})	1917.0	2232.0	**1191.6**	37.8%	46.6%
	Maze-Blue (w/o π_{Scratch})	2053.8	**441.0**	**441.0**	78.5%	0.0%

are similar in absence of the knowledge about goal. This causes CycleGAN generator in π_{CycleGAN} to effectively learn an identity mapping (see images for Maze-Blue in Fig. 4.7) and therefore π_{CycleGAN} policies are effectively equivalent to π_{PreTrain} (see Fig. 4.4 on how π_{PreTrain} is used by π_{CycleGAN}) resulting in very similar performances over trials in Fig. 4.6.

4.5.2 Ablation Studies

We study the effect of removing the most effective learning/adaptation strategies from Maze-Gray and Maze-Green and see how the performance varies. Specifically, we remove CycleGAN-based policies ($\pi_{\text{CycleGAN}_{50}}$ and $\pi_{\text{CycleGAN}_{100}}$) from Maze-Gray and policies trained from scratch ($\pi_{\text{ScratchCon}}$ and $\pi_{\text{ScratchOpt}}$) from Maze-Blue. The results are shown in Table 4.1. We observe that in both environments, removing the corresponding best learning/adaptation strategies leads our NonStationaryReplay approach still outperforming or on-par with baselines. We also observe that the average navigation cost for our NonStationaryReplay approach increases when the best learning/adaptation strategies are removed compared to those in Fig. 4.6 where these strategies were included.

4.5.3 Compute Platform and Execution Time

We ran our experiments on Intel i9 CPU with NVIDIA RTX A6000 GPU. The most computational intensive tasks in our experiments are training the CycleGAN and the neural networks. While deploying the robot with our approach, training a policy from scratch (8 epochs using data from 50 maps) would take around 5 minutes, and training a CycleGAN (100 epochs with about 1300 images in each domain) would take around 3 hours. By contrast, performing offline alt-policy replay for a single policy on a single typical trial takes about 20 seconds.

4.6 Discussion of Limitations

Although our policy selection approach is not particularly computationally intensive on its own, each strategy for learning/adaptation relies on training a deep neural network, and so requires considerable computation as the number of non-stationary policies grows. Moreover, our policy selection method relies on policies amenable to counterfactual reasoning—of which learning over subgoals planning (LSP) is one—and so may not be directly applicable for selection between model-free navigation methods, limiting its broad utility.

Our approach scales linearly in terms of both the number of policies and the number of trials: the addition of a new trial requires that offline alt-policy be run for the policies that were not deployed and, as policies are retrained, each old trial is re-evaluated. Such replay isn't computationally intensive—replaying a policy in a single typical trial takes roughly 20 seconds—and hence scales well over larger number of trials and policies. Additionally, our policy selection approach is not particularly computationally demanding on its own, but that the underlying strategies for training or adapting policies online (e.g., via CycleGAN) can be expensive, costs that are not specific to our policy selection approach, and so should not be seen as a limitation specific to our policy selection approach.

In real-world settings, the computation associated with training policies and selecting between them will not preclude using policy selection via our approach; specifically, our

problem setting envisions that robots will not be in constant operation and can perform much of this computation and policy selection while idle and waiting for its next navigation objective. Thus, such computation associated with our approach would not necessarily impede navigation performance. For very lengthy deployments spanning thousands of trials, indeed our approach might run into practical limitations since determining an accurate estimate of the robot's performance requires replaying *all* previous trials. In such scenarios, one could consider only selectively replaying specific trials or randomly sampling a subset of trials to replay to save on computation while preserving our asymptotic performance guarantees.

4.7 Contributions

We present an approach to monitor and quickly select between learning-informed navigation policies, many of which are being continuously learned or adapted during deployment. Our approach facilitates deploying multiple-such strategies for learning and visual domain adaptation in parallel, allowing our robot to choose the best among them over time as demonstrated in simulated visual maze-like environments in which we outperform state-of-the-art selection strategies by a considerable margin.

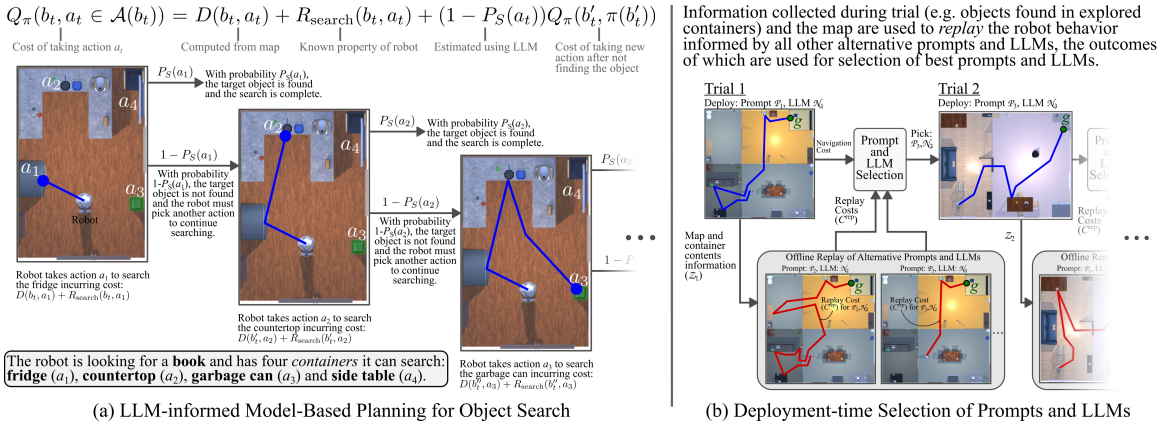
Chapter 5: Object Search in Partially-Known Environments via LLM-informed Model-based Planning and Prompt Selection

5.1 Introduction

In this chapter, we apply the framework for introspection developed in Chapter 3 to the emerging domain of large language model (LLM)-informed planning, where robot behavior is informed by LLMs. We present a novel model-based planning abstraction for object search in partially known environments that integrates the commonsense world knowledge of LLMs with classical planning. Furthermore, leveraging our introspection approach, we introduce a method for *prompt and model selection*—enabling robots to assess and improve their performance during deployment by reasoning about how alternative prompts and LLMs could have influenced their behavior, thereby reinforcing the central theme of introspection-driven improvement under uncertainty proposed in this dissertation. The work presented in this chapter has been published in Hossain et al. (2024) and Paudel and Stein (2025) and is also under review in Paudel et al. (2026).

We consider the problem of *object search* in partially-known household environments, in which a robot is tasked to find an object of interest and can use LLMs to inform the robot's behavior. Effective object search in these scenarios often requires (i) considering the impacts of the robot's immediate actions far into the future and (ii) an ability for the robot to continuously self-evaluate during deployment to improve itself over time.

Model-based planning with a high-level action abstraction is a common decision-making framework for embodied intelligence, since it not only enables deciding what action the robot should do next but also enables reasoning farther into the future about the long-term value of



(a) LLM-informed Model-Based Planning for Object Search

(b) Deployment-time Selection of Prompts and LLMs

Figure 5.1: Our high-level action abstraction, in which actions correspond to searching available containers to look for target object, enables (a) model-based planning in which an LLM informs the likelihood P_S of finding target object in a container and, (b) fast deployment-time selection of prompts and LLMs for effective object search performance. Illustration in (a) demonstrates the search problem implicit in the Bellman equation, which requires building a search tree to find the search policy that minimizes the expected cost.

a particular course of action. While many existing approaches that leverage LLMs for object search tasks use a high-level action abstraction in which the robot's actions corresponds to exploration of spaces that might contain the target object (Dorbala et al., 2023; Zhou et al., 2023; Yu et al., 2023; Arjun et al., 2024; Ge et al., 2024; Rajvanshi et al., 2024), they often prompt the LLMs to directly decide what action the robot should pick next (Zhou et al., 2023; Dorbala et al., 2023; Yu et al., 2023) without using a plannning framework. As such, LLMs can struggle to achieve good performance on many planning tasks (Valmeekam et al., 2022, 2023; Kambhampati, 2024; Kambhampati et al., 2024), particularly those necessary for embodied intelligence, including object search. Moreover, it is well-established that LLMs perform poorly on quantitative reasoning tasks (Lewkowycz et al., 2022; Arora et al., 2023; Boye and Moell, 2025; Rahman and Mishra, 2025), a limitation that translates to difficulties in choosing the *best performing* of a family of potentially suitable actions, resulting in greedy or myopic behavior that model-based reasoning is designed to avoid. However, it is not straightforward to integrate an LLM with a model-based planner and thus unclear how to

best benefit from both model-based reasoning and the useful commonsense understanding an LLM can provide.

In addition, the performance on object search tasks depends strongly on the choice of prompting strategy—e.g., the prompt text, description of robot's environment, in-context examples, etc.—and LLM model, since choosing different prompts or LLMs can result in varied performance when deployed, particularly when the deployment-time environments differ from those that were considered when designing such prompts. As such, selecting only a single prompting strategy or LLM in advance will not always elicit the best deployment-time performance. Instead, the robot should be able to choose from more than one prompting strategies or LLMs and evaluate each of those to pick the best ones during deployment. However, the process of deploying and repeatedly trying out prompting strategies or LLMs until a clear winner emerges can be problematically time consuming in general, requiring many trials to choose between them. Recent work in the space of point-goal navigation (Paudel and Stein, 2023) presents *offline alt-policy replay*, in which model-based counterfactual reasoning can be used to afford choosing the best of a family of learning-informed navigation policies, a strategy we seek to leverage for prompt and LLM selection.

To achieve effective object search performance, we therefore require a model-based approach that both informs and is informed by an LLM: with which we can plan using the commonsense world knowledge of LLMs and also introspect during deployment so as to quickly allow the system to select the best performing prompting strategy or LLM. It is a key insight of this work that a model-based planning framework for LLM-informed object search in partially-known environments can be built upon the same high-level action abstraction used in similar approaches designed for learning-informed planning under uncertainty (Stein et al., 2018). In addition, the same high-level action abstraction also affords *offline replay* (Paudel and Stein, 2023), and so can facilitate deployment-time evaluation of prompts and LLMs for object search tasks—the outcomes of which can then be used to quickly select the best performing prompts and LLMs for object search tasks.

In this work, we present an LLM-informed model-based planning framework for object search in partially-known environments, and an accompanying approach for deployment-time selection of the best prompts and LLMs for such LLM-guided object search tasks (Fig. 5.1). Our model-based planning framework leverages a frequently-used high-level action abstraction where the robot's actions correspond to revealing unsearched/unopened containers or furniture to look for a target object and leverages an LLM to make predictions about statistics of uncertainty—namely, likelihood of finding an object of interest in a location—to *inform*, rather than *replace*, model-based object search. Further leveraging this abstraction, we enable *fast deployment-time selection* of prompts and LLMs, a capability unique in this domain, by leveraging the *offline replay* approach of Paudel and Stein (2023). Our contributions are as follows:

- We identify a high-level action abstraction as a key enabler of both *model-based planning* with LLMs and *introspection*, with which such LLM-based systems can self-evaluate deployment-time behaviors.
- We present a novel approach for LLM-informed model-based high-level planning for object search in partially-known environments that integrates predictions about uncertainty from LLMs and known traversal costs from the occupancy map.
- Leveraging the action abstraction upon which planning relies, we demonstrate fast bandit-like selection of best prompts and LLMs from a family of candidate prompts and LLMs used for guiding the robot behavior in object search tasks.

Experiments in simulated ProcTHOR environments demonstrate that our LLM-informed model-based planning framework for object search outperforms LLM-based baselines that directly ask an LLM what action to pick and optimistic baselines, resulting up to 11.8% and 39.2% improvements respectively. In addition, our prompt selection approach enables quick selection of best prompts and LLMs from a family of prompts and LLMs, resulting in 6.5% lower average cost and 33.8% lower average cumulative regret over baseline UCB bandit selection. Real-robot experiments with a LoCoBot robot in an apartment building show

improvements for both our LLM-informed model-based planning approach and our prompt selection approach.

5.2 Related Work

LLMs and VLMs for Object Search Many recent works have explored the use of LLMs and vision language models (VLMs) and for object search tasks (Dorbala et al., 2023; Zhou et al., 2023; Yu et al., 2023; Arjun et al., 2024; Ge et al., 2024; Rajvanshi et al., 2024). These works use LLMs or VLMs for their commonsense world knowledge to decide where to search (Zhou et al., 2023; Dorbala et al., 2023; Yu et al., 2023). However, as they do not use a planning framework, they are often fairly myopic in their search strategies. Our work focuses on leveraging the commonsense knowledge from LLMs to *inform* rather than *replace* planning to enable reasoning about long-horizon impacts of robot's search actions: a capability important to achieve good performance for object search in partially-known environments.

LLMs and Planning LLMs have been widely used as planners (Irpan et al., 2022; Rajvanshi et al., 2024; Silver et al., 2022; Song et al., 2023; Wu et al., 2024) or to augment planning (Guan et al., 2023; Hazra et al., 2024; Zhao et al., 2024; Liu et al., 2023a; Nayak et al., 2024; Zhang et al., 2025; Ling et al., 2025). LLMs have been recently used to directly solve task planning problems in planning domain definition language (PDDL), but their abilities to generate feasible or correct solutions are brittle (Valmeekam et al., 2022, 2023). Some recent approaches therefore aim to integrate classical planning methods with LLMs (Guan et al., 2023; Hazra et al., 2024; Liu et al., 2023a; Nayak et al., 2024; Zhang et al., 2025; Ling et al., 2025). Our work is specifically focused on extracting statistics of high-level exploratory actions from LLMs to inform a model-based planning framework for object search.

Prompt Selection Prompt selection, which falls under a broader area of prompt engineering (Sahoo et al., 2024; Liu et al., 2023b), deals with selecting the prompts that achieve

the best LLM performance on downstream tasks (Yang et al., 2023). While there are approaches that aim to select the best prompts from predesigned templates (Liao et al., 2022; Liu et al., 2023b; Sorensen et al., 2022; Yang et al., 2023), these approaches focus on selecting prompts that gets the best responses from LLMs on various benchmarks and hence are not suitable for deployment-time selection of prompts in LLM-informed object search tasks, the focus of this work.

5.3 Problem Formulation

Object Search in Partially-Known Environments Our robot is tasked to find a target object g in a household environment in minimum expected cost, measured in terms of distance traveled. The environment consists of rooms, containers and objects. *Containers* are entities in the environment that can contain other objects: bed, dresser, countertop, etc. The containers are located in different rooms in the household environment. The belief state $b_t = \{m_t, q_t\}$ consists of the map m_t—with *a priori* known locations of rooms and containers but what objects exist in the containers are not known—and the robot pose q_t, both at time t. The robot must navigate to containers and search them to look for the target object. Unexplored containers form the robot's action space $\mathcal{A}$ and the robot's policy π maps the belief state b_t to a container search action $a_t \in \mathcal{A}(b_t)$. Our search policies are informed by LLMs and so depend upon the choice of LLMs and prompts used to query the LLMs.

We presume that the robot has access to a low-level navigation planner and controller that can be used to move about and interact with the environment. As such, the aim of our planner is to determine the sequence of container search actions that minimizes the expected cost of finding the target object. The performance of the robot during deployment is measured as the average distance traveled by the robot to find the target object across a sequence of trials, where each trial is held in a distinct map to find an object sampled uniformly at random from the environment.

Prompt Selection We consider that the robot's policy has access to multiple prompt templates and LLMs each represented as $\theta = (\mathcal{P}, \mathcal{N})$ where $\mathcal{P}$ denotes prompt template and $\mathcal{N}$ denotes LLM. As such, the robot has access to a family of search policies $\Pi = \{\pi_{\theta_1}, \pi_{\theta_2}, \cdots, \pi_{\theta_N}\}$ each with a unique prompt-LLM pair. The objective of prompt selection is to pick the policy with a prompt-LLM pair θ whose corresponding search actions result in minimum expected cost of finding target objects during deployment over multiple trials in distinct partially-known environments:

$$\pi_\theta^* = \underset{\pi_\theta \in \Pi}{\operatorname{argmin}} \, \mathbb{E}[C(\pi_\theta)] \tag{5.1}$$

where $\mathbb{E}[C(\pi_\theta)]$ is the expected cost incurred by the robot upon using policy π_θ with a prompt-LLM pair θ during deployment. This problem can be formulated as a multi-armed bandit problem (Sutton and Barto, 2018), solved via black-box selection algorithms like UCB (Lai et al., 1985) using Eq. (5.2):

$$\pi_\theta^{(\mathrm{k}+1)} = \underset{\pi_\theta \in \Pi}{\operatorname{argmin}} \left[\bar{C}_k(\pi_\theta) - c\sqrt{\frac{\ln k}{n_k(\pi_\theta)}} \right] \tag{5.2}$$

where $\bar{C}_k(\pi_\theta)$ is the average cost over trials 1-through-k in which policy π_θ with prompt-LLM pair θ was selected, $n_k(\pi_\theta)$ is the number of times policy π_θ was selected until trial k, and $c > 0$ is a parameter controlling the balance between exploration and exploitation. However, such approaches can be slow to converge, requiring the robot to go through multiple trials of poor performance before the best policies can be identified. White-box approaches can accelerate selection (Paudel and Stein, 2023; Paudel et al., 2024), but rely on planning abstractions that support counterfactual reasoning about robot behavior. It is our insight that LLM-informed planning strategies can be made compatible with such approaches and so can afford prompt and LLM selection in this setting.

5.4 LLM-informed Model-based Planning for Object Search

Here, we introduce our approach for LLM-informed model-based planning for object search before we discuss prompt selection in Section 5.5.

We want an approach that can leverage the commonsense world knowledge of LLMs to *inform* model-based planning rather than using LLMs to replace planning altogether. To achieve this, we introduce an approach for LLM-informed model-based object search, in which we seek to perform model-based planning wherein planning is augmented by the predictions generated by an LLM about the object locations.

Our approach takes inspiration from the learning over subgoals planning (LSP) approach of Stein et al. (2018). Their approach, designed around the aim of effective long-horizon point-goal navigation, is centered around using learning to estimate statistics associated with temporally extended actions for exploration; a learned model, trained in environments similar to those the robot sees when deployed, estimates the goodness of each such exploratory action and the likelihood that exploring the space that the action corresponds to will reach the unseen goal.

Our LLM-informed model-based object search framework adopts a similar planning abstraction. In this framework, each high-level action corresponds to searching the *containers*, which are entities in the environment that contain other objects: bed, dresser, countertop, etc. A search policy π specifies the sequence of search actions the robot intends to take to find the target object. Each such search action $a_t \in \mathcal{A}(b_t)$ has an immediate cost of first traveling to the container—corresponding to a distance $D(b_t, a_t)$ computed via A* from the occupancy grid—and then searching the container for the target object, which has a (known) search cost $R_{\text{search}}(b_t, a_t)$. With a probability P_S, the container contains the target object and so the corresponding search action successfully finds the object. Otherwise, with probability $1 - P_S$, searching continues in other containers after picking another container search action (Fig. 5.1(a)). The expected cost of a search action a_t under policy π is

computed using a Bellman equation:

$$Q_\pi(b_t, a_t \in \mathcal{A}(b_t)) = D(b_t, a_t) + R_{\text{search}}(b_t, a_t) + (1 - P_S(a_t))Q_\pi(b'_t, \pi(b'_t)) \tag{5.3}$$

The robot's policy $\pi(b_t) \equiv \text{argmin}_a\, Q_\pi(b_t, a \in \mathcal{A}(b_t))$ can be used to compute a search plan: the sequence of actions that minimizes the expected cost via Eq. (5.3) to find the target object. To speed up planning, we limit the action space using heuristics to choose up to eight containers with high likelihoods P_S and low travel costs D and select among them the container with lowest expected cost, incorporating additional containers as the robot moves and searches. It should be noted that our planning strategy is complete since, explored containers are removed from candidate action set and so in the worst case, the robot explores all available containers to find the target object if it exists. Using an LLM as the knowledge repository of where common objects of interest might be located in the given environment, we prompt the LLM provide an estimate of the marginal probability P_S and use it to compute the expected cost via Eq. (5.3). Such augmentation of planning with predictions from LLMs enables effective reasoning without explicitly relying on LLMs for multi-step reasoning thereby enabling improved performance.

5.5 Prompt Selection for LLM-informed Object Search

5.5.1 Overview of Prompt Selection

When using an LLM to inform planning for object search, prompts used to query LLMs for object likelihood predictions would ideally result in effective performance. However, effectiveness of a plan in the context of object search in partially-known environments can only be realized after the robot executes them in the environments. Thus, a poor prompting strategy may only be identified as such after the robot deploys and relies upon that LLM and prompt combination—a costly strategy of trial-and-error for robot navigation tasks. Instead, if we could identify poor prompts while limiting the need to deploy them during

deployment, we could rule them out quickly and prioritize selection of the best prompts to enable improved robot performance.

It is a key insight of this work that *offline replay* approach by Paudel and Stein (2023) can be used to select between prompts without the robot having to deploy the plans informed by LLMs using such family of prompts. While used by Paudel and Stein (2023) in the context of point-goal navigation in partially-mapped environments to replay the behavior of alternative policies without having to deploy them, we adapt offline replay to determine what the robot would have done if it had instead used a different prompt or LLM to guide its behavior. Costs from offline replay (Fig. 5.2) of alternative prompts and LLMs, $\bar{C}_k^{\text{rep}}$ (averaged over trials 1-through-k) can then be used in UCB bandit-like selection strategy (Fig. 5.1(b)) similar to that of Paudel and Stein (2023) to pick the policy π_θ with prompt-LLM pair θ for trial $k+1$ as:

$$\pi_\theta^{(\text{k}+1)} = \underset{\pi_\theta \in \Pi}{\text{argmin}} \left[\max \left(\bar{C}_k^{\text{rep}}(\pi_\theta), \bar{C}_k(\pi_\theta) - c\sqrt{\frac{\ln k}{n_k(\pi_\theta)}} \right) \right] \tag{5.4}$$

5.5.2 Object Search during a Trial

In each trial k when the robot is deployed in a partially-known environment to look for a target object g, it uses Eq. (5.4) to choose one of the policies $\pi_\theta^{(\text{k})} \in \Pi$ defined by its prompt-LLM pair $\theta = (\mathcal{P}, \mathcal{N})$, uses the prompt $\mathcal{P}$ to query the LLM $\mathcal{N}$ for object likelihood P_S and computes the next action a_t corresponding to searching one of the unexplored containers using Eq. (5.3). The robot searches the container corresponding to action a_t to look for the target object g, repeating planning and search each time the target object is not found. The total distance traveled by the robot to find the object is the cost $C_k(\pi_\theta)$ for trial k. The robot stores the information $\mathcal{Z}_k$ about the contents of all containers it explored and the known map of the environment to be used later for offline replay (Section 5.5.3).

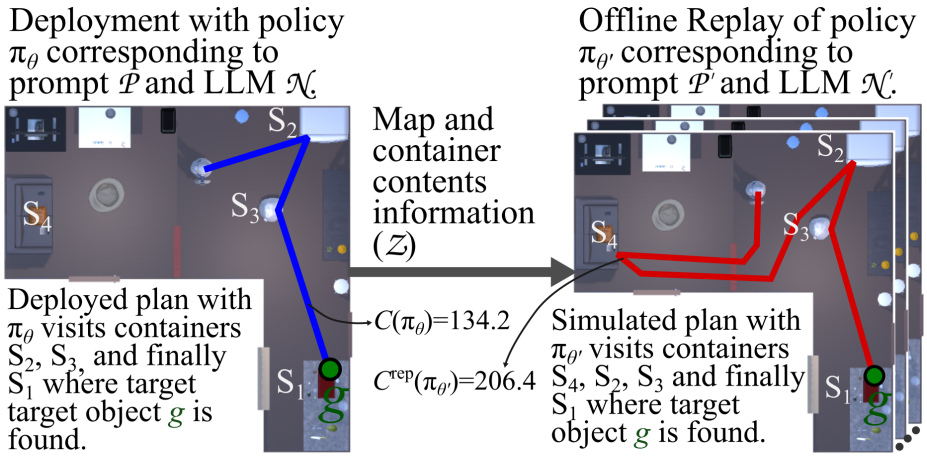


Figure 5.2: Overview of Offline Replay of Alternative Prompts and LLMs

5.5.3 Prompt Selection with Offline Replay of Alternative Prompts and LLMs

After a trial k is complete, our robot uses policy $\pi_{\theta'}$ with alternative prompt-LLM pair θ' and computes an alternative search plan. However, deploying such alternative plans is expensive in general. Instead, we use the hindsight information $\mathcal{Z}_k$ about the location of the target object found in trial k and the existing map to *replay* what the robot would have done if it had deployed a plan corresponding to an alternative prompt-LLM pair $\theta' = (\mathcal{P}', \mathcal{N}')$ (Fig. 5.2). Since we now know in advance which container contained the target object g based on the information $\mathcal{Z}_k$ obtained after completion of trial k, and pessimistically assume that all other containers would not have contained the target object, we can compute the cost of following a separate policy: the length of the trajectory the robot would have taken by following the alternate search policy to find that target object. The offline replay of a policy resembles actual deployment: replanning is done until the replayed policy suggests searching the container where the target object was found during deployment, accumulating traversal costs along the way. The average cost over trials of the offline-replayed plans, $\bar{C}_k^{\text{rep}}(\pi_{\theta'})$, for an alternative prompt-LLM pair θ' and average costs over trials of the chosen prompt-LLM pairs θ in trial k, $\bar{C}_k(\pi_\theta)$ are together used to pick the prompt and LLM in subsequent

Figure 5.3: Sample maps from ProcTHOR simulation environment

trials $k+1$ using Eq. (5.4). While the replay costs based on pessimistic assumptions may be somewhat biased, this has been empirically shown to more tightly constrain bandit-like selection compared to other relaxed assumptions in prior work of Paudel and Stein (2023) that inspired our offline replay approach.

5.6 Simulation Experiments and Results

5.6.1 Experiment Design

We perform simulation experiments for object search in 150 distinct household environments based on the ProcTHOR (Deitke et al., 2022) dataset, which consists of procedurally generated homes as shown in Fig. 5.3. Our robot has access to the underlying occupancy grid of the environment and what containers exist in what rooms, yet the contents of the containers are not known to the robot. Containers include any entities that could contain an object and so include more traditional containers, such as cabinets or boxes, but also surfaces, including tables, countertops, and bookshelves. The robot must travel to the container locations and search the containers to find the object of interest. For the purposes

of our experiments, we do not directly simulate the robot executing manipulation skills (for example, to open a fridge or a cabinet), and we treat objects as being instantaneously revealed upon reaching a container and so assign search cost $R_{\text{search}} = 0$.

Policies We perform object search experiments with our LLM+MODEL model-based planner discussed in Section 5.4 and also design two baseline policies, LLM-DIRECT and OPTIMISTIC+GREEDY discussed below:

LLM+MODEL This is our LLM-informed model-based planner that uses an LLM to obtain object likelihood probabilities and then uses Eq. (5.3) to select the best container search action as discussed in Section 5.4.

LLM-DIRECT This LLM-informed baseline policy directly prompts the LLM to respond with the container the robot should search next, instead of asking for probabilities as we do with our LLM+MODEL planner. As such, LLM-DIRECT policy does not use a planning framework to compute actions and instead directly executes actions picked by the LLM from a list of all available container search actions.

OPTIMISTIC+GREEDY This non-LLM baseline optimistically assumes that all containers could contain the target object and greedily searches the nearest container, replanning until the target object is found.

LLM Variants We experiment with two LLMs, GPT-5 Mini and Gemini 2.5 Flash, for object search tasks in PROCTHOR household environments.

Prompt Design We construct multiple prompts to query the LLMs for guiding the robot behavior. The prompt design for LLM+MODEL policy and LLM-DIRECT policy are slightly different since for LLM+MODEL we want the LLM to generate probability values for each container, while for the LLM-DIRECT policy, the LLM should directly output which container the agent should search next. While such prompts might be constructed with variations in language, context and the role that LLM should play in the interaction, each prompt includes a question asking the LLM to respond with either a probability value (for LLM+MODEL policy) or the name of the container to search (for LLM-DIRECT policy). We design three

P-CONTEXT-A
You are serving as part of a system in which a robot needs to find objects located around a household. Here is a schema that describes the connectivity of rooms in the house: The apartment contains the following rooms: bathroom, bedroom. The bedroom contains: bed, chair, sidetable. The bathroom contains: dresser, sidetable, sink, toilet. You will be asked to estimate the probability (a value between 1% and 100%) of where objects are located in that house, leveraging your considerable experience in how human occupied spaces are located. You must produce a numerical value and nothing else, as it is important to the overall functioning of the system. Here is an example exchange for an arbitrary house:
User: What is the likelihood that I find eggs in the refrigerator in the kitchen?
You: 90%
The logic here is that there is a high likelihood that a typical refrigerator in the kitchen contains eggs, but it is not guaranteed as not all refrigerators have eggs. Here is your prompt for today:
What is the likelihood that I find book in the sidetable in the bedroom?
Output: 95%

P-CONTEXT-B
You are assisting in a robotic system designed to locate items within a residence. The following is a description of the layout and connectivity between rooms in the home: The apartment contains the following rooms: bathroom, bedroom. The bedroom contains: bed, chair, sidetable. The bathroom contains: dresser, sidetable, sink, toilet. Your task is to estimate the likelihood (a percentage from 1% to 100%) that a specified object is in a given location. Base your reasoning on general patterns of human behavior and usage of household spaces. Your response must be a single numerical value, with no additional explanation, as precision is critical to system operation. Example exchange:
User: What is the probability of finding bread in the pantry in the kitchen?
You: 85%
The reasoning here is that bread is commonly stored in pantries, but exceptions exist, such as if it is refrigerated. Now, respond to this prompt:
What is the probability of finding pillow in the bed in the bedroom?
Output: 95%

P-DIRECT
You are assisting a robot in locating objects within a household based on a provided map of rooms and their contents. Your task is to determine the exact location where the specified object can be found, based on given description of the household. You will be asked pick a location to visit where the object could be found quickly. You should only pick one location from the given list. Here is an example:
User: The apartment contains: bathroom, bedroom, kitchen. The distance between rooms is as follows: bathroom and bedroom: 5.95 meters, bedroom and kitchen: 3.25 meters, bathroom and kitchen: 4.75 meters. The robot is currently located at bathroom and is looking for pillow. Available locations to search are: sink in bathroom, toilet in bathroom, bed in bedroom, sidetable in bedroom. Which of the given search locations should the robot visit to find pillow in the least time?
You: bed in bedroom
Now give your answer for another household with the following layout: The apartment contains the following rooms: bathroom, bedroom. The distance between rooms is as follows: bedroom and bathroom: 5.8 meters. The robot is currently located at bedroom and is looking for faucet. Available locations to search are: dresser in bathroom, sidetable in bathroom, sink in bathroom, toilet in bathroom, bed in bedroom, chair in bedroom, sidetable in bedroom. Which of the given search locations should the robot visit to quickly find faucet? Respond with a search location and nothing else.
Output: sink in bathroom

P-MINIMAL
What is the probability of finding plate in the dining table in the kitchen of a typical household? Your response should only include a numerical percentage value between 1% to 100% and nothing else.
Output: 80%

Figure 5.4: Samples of prompts used in our experiments

prompt templates for LLM+MODEL policy: P-CONTEXT-A, P-CONTEXT-B and P-MINIMAL, and one prompt template for LLM-DIRECT baseline policy: P-DIRECT. Examples of each are included in Fig. 5.4.

P-CONTEXT-A, P-CONTEXT-B: These prompts are designed around four main elements: (i) a description of the setting and the role that the LLM will serve, (ii) a description of the house including a list of the rooms present and the containers they contain, (iii) an example for reference, and (iv) the query asking for the probability of finding the object of interest in a container within a particular room. While the semantic meaning of these prompts are similar (see Fig. 5.4), each of these differ in terms of the language is used in the prompt text.

P-MINIMAL: This prompt doesn't include any of the aforementioned contexts about the LLM's role, environment description and reference example, and only includes the query asking for the probability of finding the target object in a container within a particular room.

P-DIRECT: This prompt for LLM-DIRECT policy is designed around five main elements: (i) a description of the setting and the role that the LLM will serve (ii) an example interaction for reference (iii) a description of the house including a list of the rooms present and the distances between them (iv) list of available containers that the robot can explore, and (v) the query asking which container the robot should explore to find the target object quickly. It should be noted that we include the distances in the prompt because we expect the LLM to behave like a planner and so provide all necessary information needed to plan effectively.

Prompt Selection We compare the prompt/LLM selection approach that leverages a high-level action abstraction to facilitate offline replay as discussed in Sec. 5.5, referred to as Replay Selection, against a baseline black-box UCB bandit selection approach. Since our selection approach is enabled by our high-level action abstraction, our other strategies that do not explicitly use planning or query LLMs—yet use the same underlying abstraction—are also amenable to offline replay, and hence can be included as a part of candidate strategies for bandit-like selection. As such, for our experiments, selection seeks to choose between all nine strategies for object search from Table 5.1. Each deployment lasts for 100 trials, each in a distinct ProcTHOR map, over which selection proceeds. While the UCB Selection uses only the deployment cost of a strategy to pick the policy-prompt-LLM combination for subsequent trials using Eq. (5.2), Replay Selection additionally uses the offline replay costs of all other policy-prompt-LLM combination to pick the strategy for next trial via Eq. (5.4).

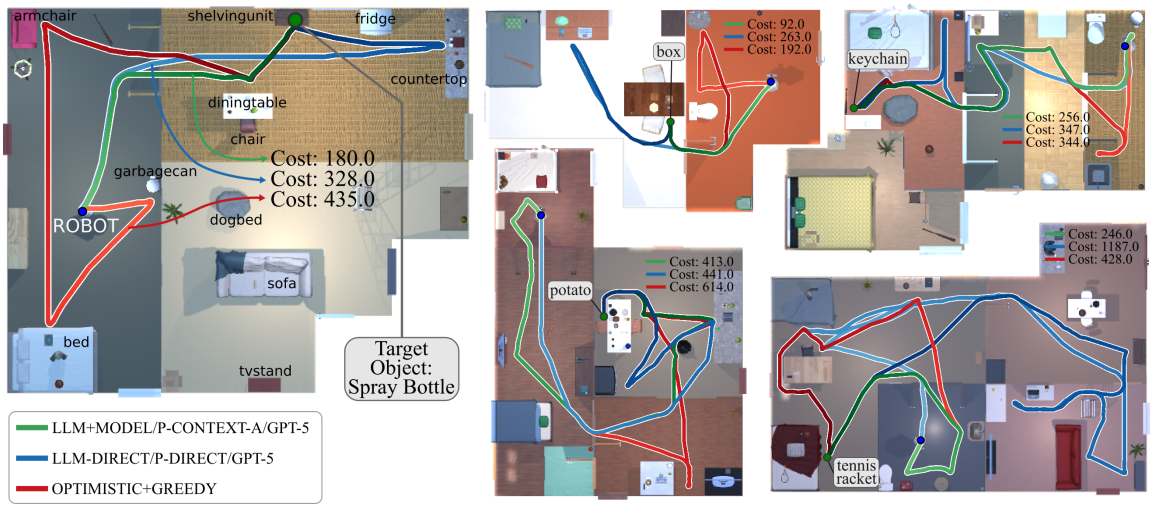


Figure 5.5: **Sample robot trajectories:** Using GPT-5 as LLM, our LLM+MODEL policy finds the target object with lower costs compared to LLM-DIRECT and OPTIMISTIC+GREEDY policies.

5.6.2 Policy Performance Results

We deploy different combinations of policies, prompts and LLMs (the state of the art GPT-5 and Gemini 2.5) each in 150 distinct maps from the PROCTHOR household environments, matching our discussion above. The average navigation costs for each such combinations are shown in Table 5.1. We also include example robot trajectories for three of the representative policies in Fig. 5.5.

Results with the GPT-5 LLM When using GPT-5 as the LLM to estimate object discovery likelihoods and so guide robot behavior, our model-based LLM+MODEL planners outperform the fully-LLM-based LLM-DIRECT policy, with our improvements between 4.0–11.8%, showing that our LLM-informed model-based planning approach enables considerable improvements in object search performance in partially-known environments. Additionally, both the LLM-DIRECT LLM-based baseline and our LLM+MODEL policies outperform the uninformed OPTIMISTIC+GREEDY baseline policy as they can both leverage the commonsense reasoning of the LLM to search more effectively. Our LLM+MODEL planners demonstrate improvements between 25.5–31.6% over the OPTIMISTIC+GREEDY policy.

Table 5.1: **Navigation Costs for Object Search Tasks**: Average costs incurred by the robot when deploying different combinations of policy, prompt and LLM in 150 distinct maps. Bold values indicate best performers with GPT-5 and Gemini respectively.

Policy / Prompt / LLM	**Avg. Navigation Cost**
LLM+MODEL (ours) / P-CONTEXT-A / GPT-5	**242.38**
LLM+MODEL (ours) / P-CONTEXT-B / GPT-5	252.14
LLM+MODEL (ours) / P-MINIMAL / GPT-5	263.94
LLM-DIRECT (baseline) / P-DIRECT / GPT-5	274.94
LLM+MODEL (ours) / P-CONTEXT-A / Gemini	224.70
LLM+MODEL (ours) / P-CONTEXT-B / Gemini	**215.59**
LLM+MODEL (ours) / P-MINIMAL / Gemini	266.20
LLM-DIRECT (baseline) / P-DIRECT / Gemini	233.41
OPTIMISTIC+GREEDY (baseline) / – / –	354.34

Results with the Gemini LLM Using Gemini as the LLM for guiding the robot behavior, we outperform the LLM-DIRECT policy with all our model-based LLM+MODEL planners except for the planner relying on the P-MINIMAL prompt, which includes no context about the surrounding environment; the remaining LLM+MODEL policies improve upon LLM-DIRECT baseline policy, achieving up to a 7.6% improvement. Additionally, our LLM+MODEL planners also outperform the uninformed OPTIMISTIC+GREEDY policy with improvements between 24.9–39.2%.

Discussion We observe that our LLM-informed model-based planning approach outperforms those that fully relies on LLM to pick robot's search actions. These results highlight the importance of using a model-based planning framework in tandem with LLMs, rather than using LLMs in place of planners, to benefit from the strengths of both—a key focus of this work. Moreover, we observe that even the same prompts used with different LLMs can result in significantly different performances: for GPT-5, P-CONTEXT-A prompt outperforms all other strategies, and for Gemini, P-CONTEXT-B prompt outperforms all other strategies. This result highlights that relying on a single prompt or LLM may not always yield the best performance and as such, one must select the best prompt and LLM combination *during deployment* to maximize good performance in the environment the robot is deployed in.

Metric	Selection Approach	Num of Trials (k)		
		$k = 20$	$k = 50$	$k = 100$
Avg. Cost	UCB Selection	258.72▲	254.63♦	247.25■
	Replay Selection (ours)	**248.12**▲	**240.40**♦	**231.29**■
Cumul. Regret	UCB Selection	884.7△	2102.8◇	3800.2□
	Replay Selection (ours)	**751.0**△	**1579.8**◇	**2516.5**□

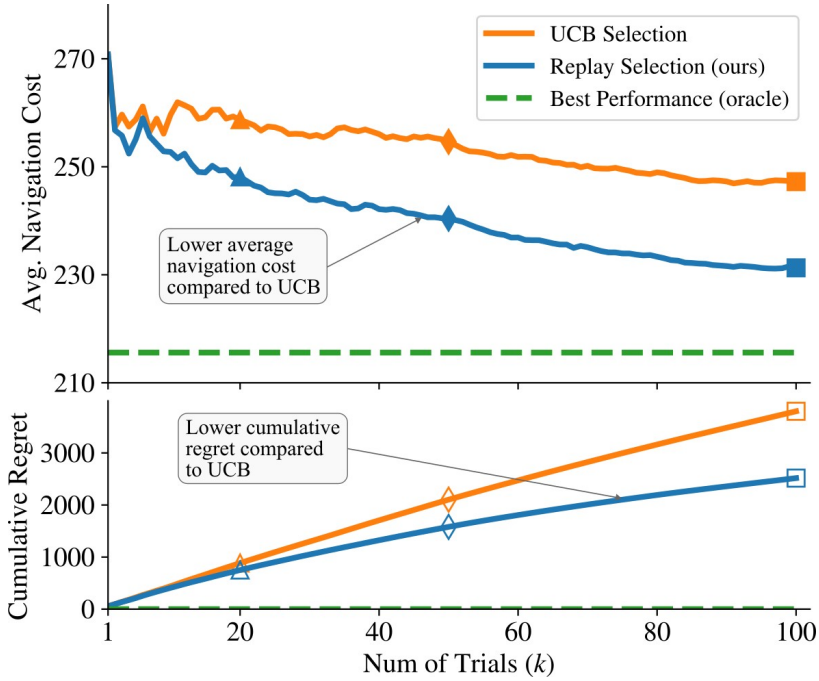


Figure 5.6: **Prompt-LLM Selection Results**: Leveraging offline replay for prompt/LLM selection allows faster selection of the best prompting strategy compared to the UCB, resulting in lower average cost (6.5% improvement) and lower cumulative regret (33.8% improvement).

5.6.3 Prompt Selection Results

As discussed in Sec. 5.5, it is another key insight of our work that the action abstraction leveraged in the previous section is compatible with the *offline replay selection*, which we can therefore leverage to more quickly select the best policy-prompt-LLM combination. To evaluate the statistical performance of selection, we generate 500 unique deployments by

Table 5.2: **Real-world LLM-informed Object Search Results:** Our LLM+MODEL policy outperforms baseline LLM-DIRECT policy. All costs are in meters.

Policy / Prompt / LLM	**Target Object**					**Average Cost**
	blanket	*cell phone*	*remote*	*pillow*	*wallet*	
LLM+MODEL (ours) / P-CONTEXT-B / GPT-5	23.5	**24.7**	**7.6**	**6.8**	**13.2**	15.2
LLM+MODEL (ours) / P-MINIMAL / GPT-5	**23.2**	**24.7**	**7.6**	**6.8**	**13.2**	**15.1**
LLM-DIRECT (baseline) / P-DIRECT / GPT-5	23.7	33.6	14.3	**6.8**	27.9	21.3

randomly permuting 100 trials from a set of 150 distinct maps, expecting the robot to perform selection over all nine policy-prompt-LLM combinations shown in Table 5.1 separately for each sequence.

In Fig. 5.6, we report the *average navigation cost*, which corresponds to the average of navigation costs incurred in trials 1-through-k, averaged over all 500 deployments. The *cumulative regret*, also shown in Fig. 5.6, tracks performance over time as the cumulative difference between the selection-based policy and a Best Performance oracle that knows in advance which strategy is best: LLM+MODEL/P-CONTEXT-B/Gemini. Our results demonstrate a reduction of 6.5% in average cost at the end of 100th trial compared to a standard UCB-bandit selection approach. In particular, we achieve 33.8% lower cumulative regret at the end of 100th trial compared to UCB-bandit selection, a number which would continue to grow with more trials.

Our results highlight the need and benefits of fast deployment-time selection of prompts and LLMs, since without such selection, the robot risks poor performance if it uses only one prompt or LLM preselected before deployment. Our selection approach enables the robot to quickly pick *during deployment* the prompts and LLMs that yield better behavior and hence maximizing long-term performance—a benefit afforded by our high-level action abstraction amenable to *offline replay* of Paudel and Stein (2023).

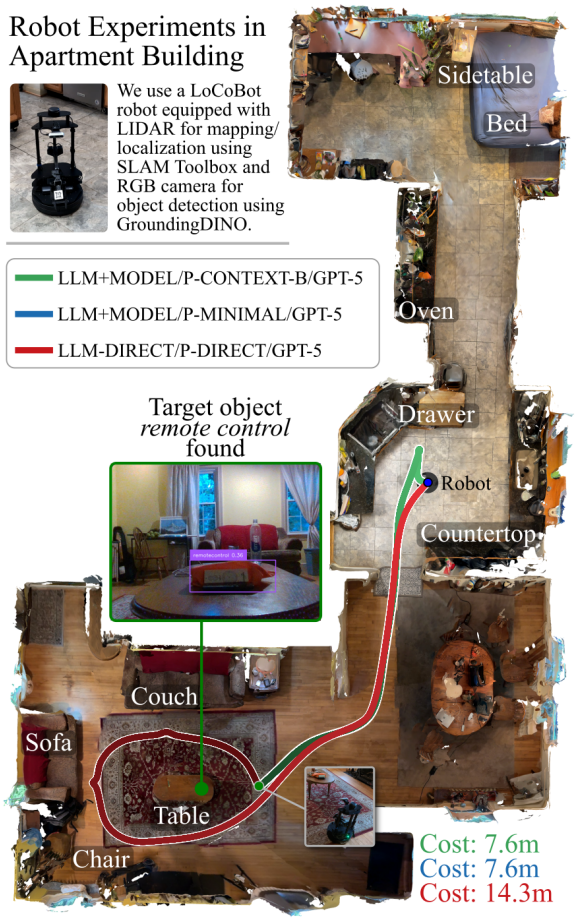


Figure 5.7: LLM-informed Object Search Experiment in an Apartment

5.7 Real-Robot Demonstration

5.7.1 Policy Performance Results

We demonstrate the effectiveness of our LLM-informed model-based planning and prompt selection approach with a LoCoBot robot in an apartment containing a kitchen, a dining room, a living room and a bedroom. These rooms contain a total of nine containers where the robot can search for target object as shown in Table 5.2. We conduct five trials in which the robot starts in the kitchen and is tasked to find a distinct object in each trial. For the purpose of demonstration, we use GPT-5 as our LLM and use P-CONTEXT-B and P-MINIMAL prompts for LLM+MODEL policy and compare with LLM-DIRECT policy using

Metric	Selection Approach	Num of Trials (k)		
		$k=2$	$k=3$	$k=5$
Avg. Cost	UCB Selection	28.6▲	21.57♦	17.0■
	Replay Selection (ours)	**24.1**▲	**18.6**♦	**15.2**■
Cumul. Regret	UCB Selection	21.9△	28.3◇	33.0□
	Replay Selection (ours)	**17.5**△	**21.0**◇	**21.6**□

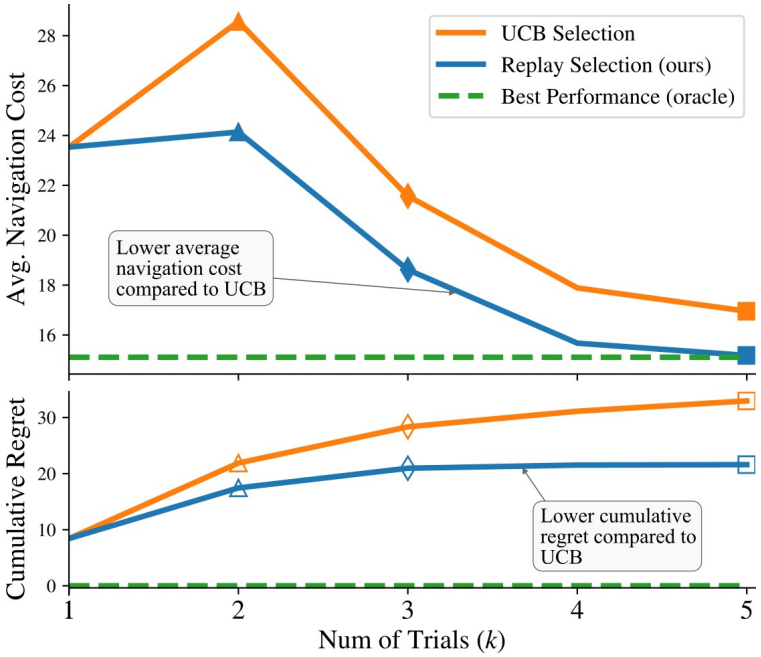


Figure 5.8: **Prompt-LLM Selection in an Apartment**: Our Replay Selection allows faster selection of the best prompting strategy compared to the UCB selection strategy, resulting in lower average cost and cumulative regret.

P-DIRECT prompt. We show one representative example in Fig. 5.7 and report all results in Table 5.2. Across five trials, our LLM+MODEL policy outperforms LLM-DIRECT policy by 29% in terms of average navigation cost.

5.7.2 Prompt Selection Results

We also deploy our prompt selection approach on the LoCoBot robot with three policy-prompt-LLM combination used in aforementioned object search experiments for five trials in which the robot is tasked to find a distinct object. The results are shown in Fig. 5.8.

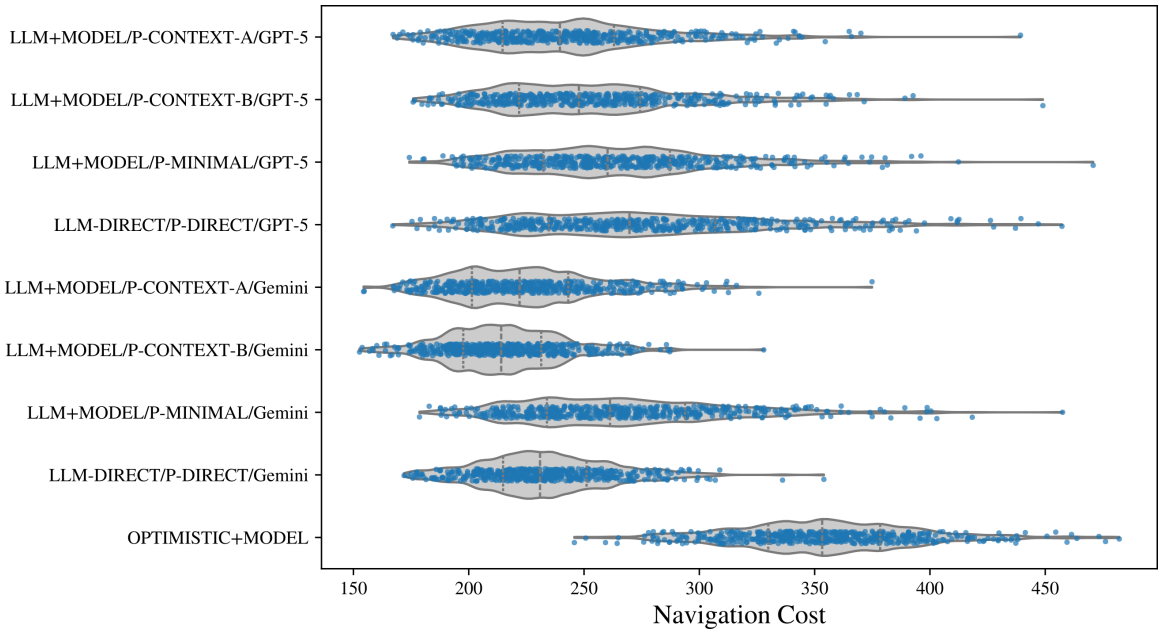


Figure 5.9: Distribution of navigation costs for different policy/prompt/LLM across 150 ProcTHOR maps. Dotted lines denote quartiles.

Over the course of five trials each with distinct target object, our Replay Selection approach incurs an average cost of 15.2m while UCB Selection incurs an average cost of 17.0m, an improvement of 10.5%. Similarly, our Replay Selection approach incurs a cumulative regret of 21.6m compared to UCB which incurs 33.0m, showing an improvement of 34.5%.

5.8 Additional Results, Details, and Discussion

5.8.1 Navigation Cost Distribution

In Fig. 5.9, we show violin plots with scatter plots of navigation costs for each policy/prompt/LLM across 150 ProcTHOR maps corresponding to the results in Table 5.1 for object search experiments.

5.8.2 Details about Belief Updates

As mentioned in Section 5.5.2, our belief state includes the occupancy map, locations of containers, whether each container has been explored or not, and objects found in each container. Upon exploring a container, the objects found in that container are added to the robot's partial map, and the container is marked as explored. If the target object was not found in that container, planning continues with the updated set of container search actions that only includes unexplored containers, and does not include any container that has been marked as explored. This process is identical for all planning strategies considered in our work.

5.8.3 Using Marginal Probabilities

For each container search action, we obtain likelihood P_S of finding the target object in that container by querying LLMs. These likelihoods are not normalized across available containers because they do not need to be a valid probability mass. For each container, P_S is a marginal probability that represents the likelihood of finding the target object in that container, which is treated as being independent of what other containers exist. The information about what other containers exist and their respective likelihoods are instead used by our planning approach to compute the best action thus alleviating the need to normalize these probabilities across containers. This is a more general formulation of the problem rather than normalizing probabilities across containers which implicitly asserts both that the object must exist and that there is only one to be found—an assumption that may not be valid in the general case.

5.8.4 Object Search Experiments with Open-Source LLMs

In addition to proprietary models, GPT-5 Mini and Gemini 2.5 Flash, we additionally perform object search experiments in 150 ProcTHOR maps with two open-source LLMs: GPT-OSS 120B and Llama3.2 3B. The results are shown in Table 5.3, which show that our

Table 5.3: Navigation costs for object search with open-source LLMs: GPT-OSS 120B and Llama3.2 3B. Our LLM+MODEL planners outperform LLM-DIRECT policy that fully relies on LLM for object search.

Policy / Prompt / LLM	**Avg. Navigation Cost**
LLM+MODEL (ours) / P-CONTEXT-A / GPT-OSS	**217.12**
LLM+MODEL (ours) / P-CONTEXT-B / GPT-OSS	221.47
LLM+MODEL (ours) / P-MINIMAL / GPT-OSS	218.78
LLM-DIRECT (baseline) / P-DIRECT / GPT-OSS	283.60
LLM+MODEL (ours) / P-CONTEXT-A / Llama3.2	302.83
LLM+MODEL (ours) / P-CONTEXT-B / Llama3.2	**281.05**
LLM+MODEL (ours) / P-MINIMAL / Llama3.2	313.24
LLM-DIRECT (baseline) / P-DIRECT / Llama3.2	806.38

Table 5.4: Token usage and incurred costs for GPT-5 and Gemini models

Model Name	**Tokens Used**			**Cost (USD)**
	Input	Output	Total	
GPT-5 Mini	903686	9264	912950	$0.14
Gemini 2.5 Flash	904209	8066	912275	$0.24

LLM+MODEL planners ourperform baseline LLM-DIRECT policy that fully relies on LLMs for object search, with improvements up to 65.2% for Llama3.2 and 23.4% for GPT-OSS.

5.8.5 Token Usage and Costs

In Table 5.4, we show approximate number of tokens used and incurred costs to use proprietary models, GPT-5 Mini and Gemini 2.5 Flash via their respective application programming interfaces (APIs), used in our experiments.

5.8.6 Scalability in terms of Number of Containers and Prompts/LLMs

As is true with many exhaustive search-based strategies, the speed of planning scales poorly with the number of containers or apartment size since the number of feasible plans scales factorially with the increase in number of containers. As discussed in Section 5.4, we limit

the action space to speed up planning by using heuristics to choose up to eight containers with high likelihoods P_S and low travel costs D and select among them the container with lowest expected cost, incorporating additional containers as the robot moves and searches. Since the set of actively considered containers is updated as those containers are explored, there is no loss of generality, and this strategy works well in practice and yields effective performance as demonstrated in our experiments.

In terms of the number of prompts and LLMs, our prompt selection approach scales linearly since the addition of a new prompt/LLM would only add the overhead of replaying this new prompt/LLM after a trial is complete. During each replay, the most computational overhead comes from querying the LLM, which we additionally mitigate by caching the LLM's responses from deployment. As such, the replay in itself is quite inexpensive (less than a couple of seconds each), making our prompt/LLM selection approach highly scalable to large number of prompts and LLMs.

5.9 Contributions

We present a novel framework that seeks to integrate LLMs with high-level model-based reasoning for performant object search. Our approach first introduces an LLM-informed model-based high-level planner for object search in partially-known environments that integrates predictions about uncertainty from LLMs and information partially-known from the environment. Second, by leveraging the high-level action abstraction upon which model-based planning relies, we demonstrate that the recent *offline replay* approach developed for model selection for learning-informed point-goal navigation (Paudel and Stein, 2023) can be made to support fast deployment-time prompt and LLM selection, a capability unique in this domain.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This dissertation introduced *introspection* as a core capability for autonomous robots, enabling them to reason about their past actions and counterfactual alternative behaviors to improve performance during deployment. Our work addressed the critical challenge of long-horizon planning under uncertainty, where robots must make reliable decisions despite having missing or incomplete knowledge of the environment.

Below is a summary of key advances enabled by the contributions presented in this dissertation:

- **Data-Efficient Policy Selection:** We first formalized the policy selection problem as an instance of the multi-armed bandit (MAB) and developed *offline alt-policy replay* as a tool for *introspection*. This technique reuses information collected during a trial to simulate the performance of alternative policies, allowing the computation of a lower bound on their potential performance. By using these bounds to constrain bandit-like selection, we achieved *fast and data-efficient selection* of the best-performing policy from a family of pretrained behaviors, while preserving guarantees on asymptotic sub-linear regret.

- **Multi-Strategy Deployment-Time Adaptation:** We extended the introspection framework to address the challenging scenario of selecting between non-stationary policies that are being continually learned or adapted during deployment, such as via online learning or visual domain adaptation. This approach allows a robot to deploy an ensemble of evolving strategies in parallel and reliably choose the best-performing

ones as soon as it improves while avoiding them early on during training or adaptation when their performance may be poorer.

- **LLM-informed Object Search and Prompt Selection:** Finally, we applied the concept of introspection to the domain of object search in partially known environments. We introduced a novel LLM-informed model-based planning framework that integrates commonsense world knowledge from LLMs with known traversal costs from an occupancy map. Leveraging our underlying abstraction, we demonstrated the ability to perform fast deployment-time selection of prompts and LLMs for guiding the robot's behavior.

Collectively, the techniques presented in this dissertation allow robots to *introspect* their behavior during long-horizon goal-directed planning tasks, enabling them to pick better behaviors and significantly improve performance and reliability in arbitrary unknown or partially known environments.

6.2 Future Work

The success of introspection in goal-directed navigation and object search tasks suggests several promising avenues for future research:

- **Generalizing to Task Planning under Uncertainty:** The ultimate goal is to apply our introspection techniques to general task planning problems under uncertainty, moving beyond specific navigation-centric tasks. In Arnob et al. (2026), I have made contributions towards developing abstractions for task planning under uncertainty. Future work should explore the integration of these abstractions with the introspection framework presented in this dissertation in order to benefit from effective and reliable deployment-time behavior in these domains.

- **Introspection in Multi-Robot Systems:** The principles of introspection presented in this dissertation have potential applications to complex scenarios like multi-robot

planning and coordination under uncertainty. This would involve an individual robot not only introspecting about its own actions and behaviors but also the anticipated behaviors of its teammates, leading to reliable and effective team behavior. While I have made contributions towards multi-robot task planning under uncertainty in Khanal et al. (2026), future work should explore how such multi-robot planning abstractions can benefit from the introspection framework presented in this dissertation.

- **Extending to Model-Free Policies:** In Chapter 3, our framework relies on policies that are amenable to counterfactual reasoning via offline replay. An important future direction is to explore how to extend the benefits of introspection and data-efficient selection to model-free policies trained via deep reinforcement learning, which are often brittle to distribution shifts and hard to replay.

- **Scaling Introspection for Long-Term Deployment:** In Chapter 4, for very lengthy deployments spanning thousands of trials, replaying all previous trials for every policy re-evaluation can become computationally intensive. Future work should investigate methods for selectively replaying specific trials by sampling a subset of past data to save on computation while maintaining the asymptotic performance guarantees of the bandit-like selection approach.

- **Introspective LLM-Guided Systems:** Further development of the LLM-informed object search system could focus on improving the robustness of the prompt and LLM selection process. This may involve integrating the LLM more deeply into the planning process to handle more complex, multi-step tasks or developing methods for the robot to introspect the quality of the LLM's own predictions over time. Additionally, our approach presented in Chapter 5 only considers a predefined set of prompts, and so deployment-time refinement and adaptation of prompts could be an important avenue for future research.

Bibliography

A. Ajay, A. Gupta, D. Ghosh, S. Levine, and P. Agrawal. Distributionally adaptive meta reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

P. Arjun, A. Melnik, and G. C. Nandi. Cognitive planning for object goal navigation using generative AI models. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki. Meta reinforcement learning for sim-to-real domain adaptation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

R. I. Arnob, M. Merlin, A. Paudel, B. Hedegaard, G. Konidaris, and G. J. Stein. Effective task planning with missing objects using learning-informed object search. *arXiv preprint arXiv:2602.11468*, 2026.

D. Arora, H. Singh, et al. Have LLMs advanced enough? A challenging problem solving benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

P. Bhandari and A. Paudel. Using contextual information for sentence-level morpheme segmentation. *arXiv preprint arXiv:2403.15436*, 2024.

K. Bousmalis, N. Silberman, D. Dohan, and D. Erhan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to

improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

J. Boye and B. Moell. Large language models and mathematical reasoning failures. *arXiv preprint arXiv:2502.11574*, 2025.

C. Bradley, A. Pacheck, G. J. Stein, S. Castro, H. Kress-Gazit, and N. Roy. Learning and planning for temporally extended tasks in unknown environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

D. Bradley and J. A. D. Bagnell. Domain adaptation for mobile robot navigation. In *NeurIPS 2009 Learning from Multiple Sources with Applications to Robotics Workshop*, 2009.

S. Candido and S. Hutchinson. Minimum uncertainty robot navigation using information-guided POMDP planning. In *2011 IEEE International Conference on Robotics and Automation*, 2011.

A. S. Chen, G. Chada, L. Smith, A. Sharma, Z. Fu, S. Levine, and C. Finn. Adapt on-the-go: Behavior modulation for single-life robot deployment. In *NeurIPS 2023 Robot Learning Workshop*, 2023.

S. Daftry, S. Zeng, J. A. Bagnell, and M. Hebert. Introspective perception: Learning to predict failures in vision systems. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi, and R. Mottaghi. ProcTHOR: Large-scale embodied AI using procedural generation. *Advances in Neural Information Processing Systems*, 2022.

V. S. Dorbala, J. F. Mullen Jr, and D. Manocha. Can an embodied agent find your "cat-shaped mug"? LLM-guided exploration for zero-shot object navigation. *arXiv preprint arXiv:2303.03480*, 2023.

M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection through learned hidden markov models. *Artificial Intelligence*, 2006.

Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 2016.

W. Ge, C. Tang, and H. Zhang. Commonsense scene graph-based target localization for object search. *arXiv preprint arXiv:2404.00343*, 2024.

A. Ghosh and S. R. Chowdhury. Model selection in reinforcement learning with general function approximations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2022.

J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1979.

L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 2023.

S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2021.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

R. Hazra, P. Z. Dos Martires, and L. De Raedt. SayCanPay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018.

J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. In *International Conference on Machine Learning*, 2018.

S. Hossain, A. Paudel, and G. J. Stein. Enhancing object search by augmenting planning with predictions from large language models. In *CoRL Workshop on Learning Effective Abstractions for Planning*, 2024.

S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

A. Irpan, A. Herzog, A. T. Toshev, A. Zeng, A. Brohan, B. A. Ichter, B. David, C. Parada, C. Finn, C. Tan, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

S. Kambhampati. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 2024.

S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt, and A. B. Murthy. LLMs can't plan, but can help planning in LLM-modulo frameworks. In *International Conference on Machine Learning*, 2024.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 2011.

A. Khanal, A. Paudel, H. Pham, and G. J. Stein. Multi-robot learning-informed task planning under uncertainty. In *International Conference on Robotics and Automation (ICRA)*, 2026.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

J. Kulhánek, E. Derner, T. De Bruin, and R. Babuška. Vision-based navigation using deep reinforcement learning. In *2019 European Conference on Mobile Robots (ECMR)*, 2019.

A. Kumar, Z. Fu, D. Pathak, and J. Malik. RMA: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems*, 2021.

H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 2011.

T. L. Lai, H. Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 1985.

T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

S. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

J. Lee, A. Pacchiano, V. Muthukumar, W. Kong, and E. Brunskill. Online model selection for reinforcement learning with function approximation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021.

A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 2022.

K. Liang, Z. Zhang, and J. F. Fisac. Introspective planning: Aligning robots' uncertainty with inherent task ambiguity. *Advances in Neural Information Processing Systems*, 2024.

C. Liao, Y. Zheng, and Z. Yang. Zero-label prompt selection. *arXiv preprint arXiv:2211.04668*, 2022.

S. Ling, Y. Wang, C. Fan, T. L. Lam, and J. Hu. ELHPlan: Efficient long-horizon task planning for multi-agent collaboration. *arXiv preprint arXiv:2509.24230*, 2025.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*. Elsevier, 1995.

B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

H. Liu, S. Dass, R. Martín-Martín, and Y. Zhu. Model-based runtime monitoring with interactive imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 2023b.

M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, 2015.

P. Mallozzi, E. Castellano, P. Pelliccione, G. Schneider, and K. Tei. A runtime monitoring framework to enforce invariants on reinforcement learning agents exploring complex

environments. In *2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)*, 2019.

S. Mannor and J. N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 2004.

J. B. Marshall, N. K. Makhija, and Z. D. Rothman. The introspective robot: Using self-prediction to improve robot learning. *Science*, 2008.

M. Merlin, N. Parikh, E. Rosen, and G. Konidaris. Locally observable markov decision processes. In *ICRA 2020 Workshop on Perception, Action, Learning*, 2020.

P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019.

S. Nayak, A. Morrison Orozco, M. Have, J. Zhang, V. Thirumalai, D. Chen, A. Kapoor, E. Robinson, K. Gopalakrishnan, J. Harrison, et al. Long-horizon planning for multi-agent robots in partially observable environments. *Advances in Neural Information Processing Systems*, 2024.

A. Pacchiano, C. Dann, C. Gentile, and P. Bartlett. Regret bound balancing and elimination for model selection in bandits and RL. *arXiv preprint arXiv:2012.13045*, 2020.

S. Palazzo, D. C. Guastella, L. Cantelli, P. Spadaro, F. Rundo, G. Muscato, D. Giordano, and C. Spampinato. Domain adaptation for outdoor robot traversability estimation from RGB data with safety-preserving loss. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

A. Paudel. Sophisticated students in boston mechanism and gale-shapley algorithm for school choice problem. *arXiv preprint arXiv:2108.05951*, 2021.

A. Paudel. Learning for robot decision making under distribution shift: A survey. *arXiv preprint arXiv:2203.07558*, 2022a.

A. Paudel. Motion primitives based path planning with rapidly-exploring random tree. *arXiv preprint arXiv:2210.15784*, 2022b.

A. Paudel. Introspection for long-horizon robot planning under uncertainty. In *IEEE ICRA 2025 Doctoral Consortium*, 2025.

A. Paudel and G. J. Stein. Data-efficient policy selection for navigation in partial maps via subgoal-based abstraction. In *International Conference on Intelligent Robots and Systems (IROS)*, 2023.

A. Paudel and G. J. Stein. Deployment-time selection of prompts for LLM-informed object search in partially-known environments. In *ICRA 2025 Workshop on Foundation Models and Neuro-Symbolic AI for Robotics*, 2025.

A. Paudel, R. Dhakal, and S. Bhattarai. Room classification on floor plan graphs using graph neural networks. *arXiv preprint arXiv:108.05947*, 2021.

A. Paudel, X. Xiao, and G. J. Stein. Multi-strategy deployment-time learning and adaptation for navigation under uncertainty. In *Conference on Robot Learning (CoRL)*, 2024.

A. Paudel, A. Khanal, R. I. Arnob, S. Hossain, and G. J. Stein. Object search in partially-known environments via LLM-informed model-based planning and prompt selection. *arXiv preprint arXiv:2603.23800*, 2026.

Q. M. Rahman, P. Corke, and F. Dayoub. Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, 2021.

R. Rahman and A. A. Mishra. A fragile number sense: Probing the elemental limits of numerical reasoning in LLMs. *arXiv preprint arXiv:2509.06332*, 2025.

A. Rajvanshi, K. Sikka, X. Lin, B. Lee, H.-P. Chiu, and A. Velasquez. SayNav: Grounding large language models for dynamic planning to navigation in new environments. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2024.

J. Reisinger, P. Stone, and R. Miikkulainen. Online kernel selection for bayesian reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

C. Richter and N. Roy. Safe visual navigation via deep learning and novelty detection. In *Robotics: Science and Systems*, 2017.

C. Richter, J. Ware, and N. Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

G. Schoettler, A. Nair, J. A. Ojea, S. Levine, and E. Solowjow. Meta-reinforcement learning for robotic industrial insertion tasks. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

J. Shin, A. Hakobyan, M. Park, Y. Kim, G. Kim, and I. Yang. Infusing model predictive control into meta-reinforcement learning for mobile robots in dynamic environments. *IEEE Robotics and Automation Letters*, 2022.

T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.

C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

T. Sorensen, J. Robinson, C. M. Rytting, A. G. Shaw, K. J. Rogers, A. P. Delorey, M. Khalil, N. Fulda, and D. Wingate. An information-theoretic approach to prompt engineering without ground truth labels. *arXiv preprint arXiv:2203.11364*, 2022.

G. J. Stein and N. Roy. GeneSIS-RT: Generating synthetic images for training secondary real-world tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

G. J. Stein, C. Bradley, and N. Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In *Conference on Robot Learning*. PMLR, 2018.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *International Conference on Learning Representations*, 2017.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati. Large language models still can't plan (A benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. On the planning abilities of large language models - A critical investigation. *Advances in Neural Information Processing Systems*, 2023.

G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. Rae, P. Mirowski, J. Z. Leibo, A. Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

Y. Wu, Z. Xiong, Y. Hu, S. S. Iyengar, N. Jiang, A. Bera, L. Tan, and S. Jagannathan. SELP: Generating safe and efficient task plans for robot agents with large language models. *arXiv preprint arXiv:2409.19471*, 2024.

M. Wulfmeier, A. Bewley, and I. Posner. Addressing appearance change in outdoor robotics with adversarial domain adaptation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Y. Xiang, W. Niu, J. Liu, T. Chen, and Z. Han. A PCA-based model to predict adversarial examples on Q-learning of path finding. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, 2018.

S. Yang, J. Kim, J. Jang, S. Ye, H. Lee, and M. Seo. Improving probability-based prompt selection through unified evaluation and analysis. *arXiv preprint arXiv:2305.14877*, 2023.

B. Yu, H. Kasaei, and M. Cao. L3MVN: Leveraging large language models for visual target navigation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

X. Zhang, H. Qin, F. Wang, Y. Dong, and J. Li. LaMMA-P: Generalizable multi-agent long-horizon task allocation and planning with LM-driven PDDL planner. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Z. Zhao, W. S. Lee, and D. Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 2024.

K. Zhou, K. Zheng, C. Pryor, Y. Shen, H. Jin, L. Getoor, and X. E. Wang. ESC: Exploration with soft commonsense constraints for zero-shot object navigation. In *International Conference on Machine Learning*, 2023.

W. Zhou, J. S. Berrio, S. Worrall, and E. Nebot. Automated evaluation of semantic segmentation robustness for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017a.

Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017b.

Y. Zhu, P. Thangeda, M. Ornik, and K. Hauser. Few-shot adaptation for manipulating granular materials under domain shift. In *Proceedings of Robotics: Science and Systems*, 2023.

Biography

Abhishek Paudel is a PhD candidate in Computer Science at George Mason University, advised by Prof. Gregory J. Stein in the Robotic Anticipatory Intelligence & Learning (RAIL) Group. His research lies at the intersection of robotics and machine learning, with a focus on long-horizon planning under uncertainty. His work focuses on integrating planning with learning to enable effective and reliable robot behavior in dynamic and uncertain environments. He received his Bachelor's degree from Tribhuvan University in Nepal and his Master's degree from George Mason University.