The Thesis Committee for Raghav Arora
certifies that this is the approved version of the following thesis:

Anticipatory Reinforcement Learning for Improving Continual Planning in Persistent Environments

SUPERVISING COMMITTEE:

Peter Stone, Supervisor

Roberto Martín-Martín, Co-supervisor

Anticipatory Reinforcement Learning for Improving Continual Planning in Persistent Environments

by

Raghav Arora

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2026

Acknowledgments

First and foremost I would like to thank my advisor, Dr. Peter Stone, for his guidance and encouragement not only for this thesis, but throughout my master's program. Peter has always made himself available to guide me through research, provide me with necessary resources, and give his feedback that proved invaluable in shaping this thesis. I feel incredibly grateful for getting the opportunity to work with Peter, which exposed me to fascinating robotics research and engineering. He trusted me to pursue my dream to work on real robots, and to go a step further to work on the latest robotics challenges on multiple robot embodiments.

Next, I would like to thank my co-supervisor, Dr. Roberto Martín-Martín. Roberto has been incredibly kind, patient, and generous with his time and guidance. His guidance as a mentor and as a teacher in the Advanced Robot Manipulation course helped me understand research trends and motivated me to pursue this project.

I would like to thank Dr. Yoonchang Sung, who helped formulate the idea for this work, and Yoonwoo Kim, who guided me towards open research gaps. Discussions with them started this research thread and pushed me to align my thoughts and thesis direction. I am grateful to Dr. Amy Zhang, who helped me understand the core concepts of Reinforcement Learning that were highly relevant to this work, and provided regular guidance to debug and revise my approach.

Lastly, I would like to thank all my friends and family. My parents, Davinder Kumar and Girija Arora, have backed my education at every step, allowing me to work so far from my home. I am deeply grateful to Nirlipta Pande, without whom I would not have come to UT Austin, and whose support helped set this research in motion. I would also like to thank my friends at UT: Rohit Khanna, Akankshya Satapathy, Preya Somani, Dakshata Koli; and from before: Jeet Yadav and Sanskar Jain for being constant pillars of support.

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by the National Science Foundation (FAIN-2019844, NRT-2125858), the Office of Naval Research (N00014-24-1-2550), Army Research Office (W911NF-17-2-0181, W911NF-23-2-0004, W911NF-25-1-0065), Lockheed Martin, Lyda Hill Philanthropies, and Good Systems, a research grand challenge at the University of Texas at Austin. The views and conclusions contained in this document are those of the author alone.

Abstract

Anticipatory Reinforcement Learning for Improving Continual Planning in Persistent Environments

Raghav Arora, M.S.E.
The University of Texas at Austin, 2026

SUPERVISORS: Peter Stone, Roberto Martín-Martín

Robots deployed in persistent environments, such as homes and restaurants, are increasingly asked to carry out long sequences of tasks where the world does not reset between instructions. A myopic agent that optimizes only for the immediate task can inadvertently leave the environment in a state that raises the expected cost of future tasks. Anticipatory planning mitigates this by preparing for predictable future routines. However, existing anticipatory planning methods rely on expensive, planner-generated offline datasets and are limited to a single-task lookahead objective, so they fail when preparatory actions must be amortized over a longer horizon. In this thesis, we propose a neurosymbolic approach that learns an anticipatory value function online via Reinforcement Learning. This creates a multi-task credit horizon without requiring a planner-generated anticipatory cost dataset. We deploy this value as a terminal heuristic to guide a cost-bounded symbolic search, and evaluate it in persistent two-room and three-room restaurants. Our value-guided anticipatory agent reduces total sequence cost by 31.8% (two-room) and 27.5% (three-room) relative to a myopic oracle, recovering 76% and 97% of the savings a clairvoyant planner attains with the future revealed in advance. The three-room restaurant isolates the benefit of reaching beyond a single task: a one-task supervised baseline recovers almost none of the saving (3.1%), whereas our multi-task value captures most of it (27.5%).

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Robots are increasingly deployed in homes, offices, and restaurants, where they will be asked to carry out long sequences of instructions in a world that persists between tasks. Longitudinal studies of household routines show that object use follows routines that can be predicted from a user's past activity (Patel and Chernova, 2022; Patel et al., 2023), so an agent has something specific to prepare for. A *myopic* agent receives a task and optimizes the sequence of actions needed to complete the task. In a persistent environment, a myopic agent can create side effects: the cheapest way to finish the current task may leave the environment in a state that raises the expected cost of future tasks, or in the extreme, makes some future task infeasible. *Anticipatory* behavior, instead, seeks states that complete the present task while leaving the environment well prepared for what comes next. Anticipation helps only under two conditions: the terminal state of the current task must change the expected cost of future tasks, which we call state-coupling, and the cheapest way to finish the current task must not already be the best-prepared one. Figure 1.1 illustrates the conflict where a myopic and an anticipatory agent receive the same sequence of tasks. The myopic agent completes the first task greedily, but leaves the environment in a state from which the second task is far more costly.

In a persistent environment, the agent completes a task and leaves the environment in some state; the next task begins from that state rather than from a reset. This persistence couples tasks: the terminal state of one task becomes the initial state of the next, so the manner in which a task is completed affects the cost of every task that follows. If future cost is independent of how the current task ends, then greedily solving each task is optimal. Given coupling and conflict, the form the solution takes depends on what the agent knows. If the future task sequence is known, a joint plan over the sequence is optimal. When it is unknown, the agent must instead minimize expected future cost under the task distribution, which is anticipation. The relevant

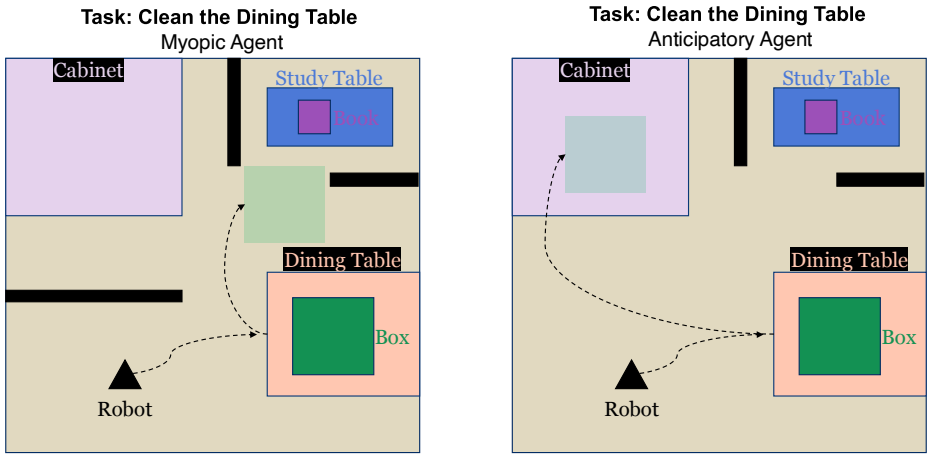


Figure 1.1: Motivation for Anticipatory Planning. The agent gets the task to clear the dining table. The myopic agent chooses the lowest cost plan, and places the box on the floor next to the table. This placement blocks the agent's path to the study table which would be needed for future tasks. The anticipatory agent instead places the box on the cabinet, which is more costly for the current task but leaves the path to the study table clear for future tasks.

comparison is over the total cost of the task sequence, not the cost of any single task: an anticipatory agent may pay more than the myopic agent on the current task, yet incur a lower total cost, because it does not pay to undo a trap it never created. A prepared terminal state may also be amortized across multiple likely future tasks, as when clearing a shared walkway once reduces the cost of every future task that must cross it.

Goal-based tasks are traditionally solved by symbolic planners, which compute a plan from a single initial state to a single goal (Fikes and Nilsson, 1971; Helmert, 2006). Because each task is solved in isolation, the resulting plan is myopic in a persistent environment, which, as discussed above, is harmful when tasks are coupled. Dhakal et al. (2023) introduced anticipatory planning as the joint optimization of current-task cost and expected future-task cost, with a learned offline estimator used to guide symbolic planning. They demonstrated the advantage of anticipatory planning over a myopic agent in an environment in which tasks have high interdependence, and the goal state for one task affects the plan for future tasks (Dhakal et al., 2023,

2024; Talukder et al., 2024; Arora et al., 2024; Singh et al., 2024). However, this offline, planner-based approach to anticipation suffers from four major limitations. **(L1)** It assumes the task distribution is fixed and known by the agent, allowing the anticipatory cost to be learned offline through supervised learning. **(L2)** Building the training dataset requires solving a planning problem for every task from every sampled start state, so the number of planner calls grows with the product of states and tasks, and does not scale to large task spaces. **(L3)** It minimizes the expected cost of an immediately available task and a *single* next task in the sequence, and so, anticipating only one task ahead, cannot capture preparedness that pays off over longer horizons. **(L4)** The anticipated future task does not depend on the current task, so anticipation cannot exploit correlations between what the agent is doing now and what is likely to come next.

This work instead formulates continual planning over a persistent world as a continuing task-conditioned Markov Decision Process (MDP), in which a single agent solves an ongoing stream of tasks, each beginning where the last ended. Rather than estimating future cost offline with a symbolic planner, the agent learns a task-conditioned *preparedness* value function that captures how well a terminal state serves the tasks expected to follow, and improves it online through reinforcement learning. Because value estimates propagate across task boundaries, the agent can anticipate more than a single task ahead, and because it learns from experience, it does not require a precomputed cost dataset. Like prior work, we assume the task distribution is stationary and known, and use it when estimating future value; we do not address that assumption (L1). Our approach addresses (L2) by learning from online experience instead of a planner-generated cost dataset, and (L3) by extending anticipation beyond a single task ahead. It does not address (L4): we draw each task independently, so the anticipated future task does not depend on the current one. Relaxing the distribution assumption (L1) and modeling task correlations (L4) are left to future work.

The learned preparedness value is the central object of this thesis, and we

deploy it in two ways. First, it can act directly as a greedy policy, choosing each action to maximize expected value. Second, it can serve as a heuristic for a cost-bounded symbolic search. Among the ways of finishing the current task, the search prefers the terminal state the value judges best prepared for what follows. Because it explores task completions rather than a single greedy trajectory, the search can reach terminal states whose benefit is amortized across many future tasks.

Thesis statement: Reinforcement Learning can estimate how well a state serves as a starting point for completing future tasks, and using this preparedness value to select how each task is completed, either directly as a policy or as a heuristic for symbolic search, may reduce the total cost of a task sequence relative to a myopic agent, without an offline cost estimator.

Contributions

This thesis makes the following contributions:

1. We recast anticipatory planning as a **continuing task-conditioned MDP**, in which preparedness for future tasks is captured by a value function conditioned on the current task.

2. We learn the preparedness value **online from experience**, so the agent needs neither the expensive symbolic-planner-based cost dataset that prior work precomputed offline, nor a supervised estimator that was trained over it.

3. We enable **multi-task anticipation** by propagating value across task boundaries, extending anticipation beyond the single task ahead that prior work considered. To test it, we build a **restaurant benchmark with a controllable anticipation signal** — the distance over which a single investment must amortize. When that distance spans several tasks, a one-task objective barely im-

proves on the myopic reference even when handed the prepared states (+3.1%), while the multi-task value cuts total cost by 27.5%.

4. We deploy the learned value as a **search heuristic for preparedness-aware planning**: a cost-bounded symbolic search ranks the ways of completing the current task by their preparedness value and selects the best-prepared one. Searching over task completions rather than following a single greedy trajectory lets the agent reach well-prepared terminal states, and puts our method in the same value-guided-planning form as prior anticipatory work, enabling a direct comparison. Deployed this way, the anticipatory value cuts total task-sequence cost by 31.8% over a myopic symbolic-planning oracle, recovering 76% of what a clairvoyant oracle shown the future in advance achieves.

Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 covers the background this thesis builds on: Markov decision processes and value-based reinforcement learning, goal-conditioned reinforcement learning, classical symbolic planning, and the anticipatory planning framework of prior work. Chapter 3 formalizes the continuing task-conditioned MDP and defines the preparedness value function and the myopic and anticipatory objectives. Chapter 4 presents the method: learning the preparedness value online with reinforcement learning, and deploying it as a heuristic for preparedness-aware symbolic search. Chapter 5 evaluates the anticipatory agent against myopic baselines and planning oracles in the restaurant environment. Chapter 6 summarizes the contributions and discusses limitations and future directions.

Chapter 2: Background and Related Work

This chapter reviews the background this thesis builds on: reinforcement learning for sequential tasks, value learning in persistent environments, and prior work on anticipatory planning. Section 2.1 introduces value-based reinforcement learning, from Markov decision processes to task-conditioned value functions, and reviews prior work on continuing, non-episodic environments. Section 2.2 describes the structured action-value functions our method builds on. Section 2.3 describes how symbolic planning finds a sequence of actions to reach a goal; Section 2.4 explains how a learned heuristic can guide such a planner; and Section 2.5 defines anticipatory planning and reviews prior approaches to it.

2.1 Value-Based Reinforcement Learning

Value-based reinforcement learning represents a policy through a value function and improves it from experience. This section introduces the pieces our method builds on: the Markov decision process and Q-learning, the distinction between episodic and continuing tasks, and value functions conditioned on the current task.

2.1.1 Markov Decision Processes and Q-Learning

A Markov Decision Process (MDP) is a mathematical model for sequential decision-making problems in stochastic environments (Puterman, 1994). It is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition kernel on $\mathcal{S}$, $\mathcal{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor (Sutton and Barto, 2018). At each state $s \in \mathcal{S}$, an agent takes an action $a \in \mathcal{A}$, transitions to a new state according to the transition kernel $\mathcal{P}$, and receives a reward $r = \mathcal{R}(s, a)$. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps each state to a distribution over actions. The objective of the agent is to maximize the expected

discounted return $\mathbb{E}_\pi[G_t]$, where the return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted sum of future rewards and γ controls how strongly future rewards are weighted against immediate ones. The state-action value function $Q^\pi(s,a)$ of a policy π is the expected return after taking action a in state s and following π thereafter:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a \right]$$

The state value function $V^\pi(s)$ is the corresponding expectation over actions drawn from the policy:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right].$$

These value functions satisfy the Bellman equations, which express the value of a state in terms of the values of its successors. The Bellman expectation equation decomposes Q^π into the immediate reward and the discounted value of the next state:

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim\mathcal{P}}\left[\mathcal{R}(s,a) + \gamma\, V^\pi(s')\right].$$

An optimal policy π^* maximizes the expected return from every state. Its value functions Q^* and V^* satisfy the Bellman optimality equation, which replaces the expectation over the policy with a maximization over actions:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{P}}\left[\mathcal{R}(s,a) + \gamma \max_{a'} Q^*(s',a')\right], \qquad V^*(s) = \max_a Q^*(s,a).$$

The optimal policy is recovered greedily as $\pi^*(s) = \arg\max_a Q^*(s,a)$.

When $\mathcal{P}$ and $\mathcal{R}$ are known, Q^* can be found by dynamic programming, iterating the Bellman optimality equation to its fixed point; when they are unknown and the state space is too large to enumerate, the agent instead learns Q^* from sampled experience. Temporal-difference Q-learning (Watkins and Dayan, 1992) does so by bootstrapping, updating $Q(s,a)$ from an observed transition (s,a,r,s') toward $r + \gamma \max_{a'} Q(s',a')$. Deep Q-networks (DQN) scale this idea to high-dimensional states by approximating Q with a neural network. An experience-replay buffer stabilizes training by decorrelating consecutive updates, and a periodically updated target

network holds the bootstrap target fixed (Mnih et al., 2015). In the DQN target, the same target network both selects and evaluates the bootstrap action through the $\max_{a'}$ operator, which tends to overestimate Q^*. Double-DQN reduces this bias by selecting the next action with the online network and evaluating it with the target network (Hasselt et al., 2016).

2.1.2 Continuing vs. Episodic Formulations

Reinforcement learning problems are commonly divided into *episodic* and *continuing* tasks (Sutton and Barto, 2018). An episodic task ends in a *terminal state*: the return terminates there, the state is assigned value zero, and bootstrapping stops. A continuing task has no terminal state; the return is an infinite discounted sum, value estimates bootstrap indefinitely, and the discount factor γ alone sets the effective horizon over which future rewards matter. The two cases differ only in where bootstrapping stops. Writing the one-step temporal-difference target with a terminal indicator d,

$$y = r + \gamma\,(1 - d)\max_{a'} Q(s', a'),$$

we see that an episodic task sets $d = 1$ at its terminal state, truncating the target to r, whereas a continuing task keeps $d = 0$ throughout and always bootstraps. Persistent environments are continuing: completing a task neither ends nor resets the world, and the state it leaves behind becomes the starting state of the next task, so the task stream continues without a natural terminal state — the setting studied in continual reinforcement learning (Khetarpal et al., 2022). Any episodic structure is therefore imposed rather than intrinsic: a per-task step budget or a periodic reset cuts the stream into segments for tractability, but neither is a true terminal state, and partial-episode bootstrapping (discussed below) handles such cutoffs accordingly. Whether the value recursion nonetheless stops at a task boundary or continues across it is a modeling choice, not a property of the environment; Chapter 3 makes this choice precise and uses it to distinguish the two agents we study.

2.1.3 Goal- and Task-Conditioned Value Functions

A value function is defined with respect to a fixed reward, so the Q^π and V^π above describe a single task; to act across a stream of different tasks, the agent needs a value function conditioned on the current task. Universal value function approximators (UVFAs) provide this value by conditioning the value on both the state and a goal, learning a single approximator that generalizes across goals rather than a separate one per goal (Schaul et al., 2015):

$$Q^\pi(s, g, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \, r_g(s_{t+k}, a_{t+k}) \;\middle|\; s_t = s,\, a_t = a \right],$$

where r_g is the reward for goal g, so the value generalizes over goals as well as states. Successor features achieve a similar generalization by decoupling environment dynamics from rewards. With a task's reward written as $r(s, a) = \phi(s, a)^\top w$, the value factorizes as

$$Q^\pi(s, a) = \psi^\pi(s, a)^\top w, \qquad \psi^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \, \phi(s_{t+k}, a_{t+k}) \right],$$

where the successor features ψ^π depend only on the dynamics and w only on the reward; this decoupling supports zero-shot transfer across tasks that share dynamics through generalized policy improvement (Barreto et al., 2017). More broadly, goal-conditioned reinforcement learning learns policies and values conditioned on a goal drawn from a goal space (Liu et al., 2022), for example by casting goal-reaching as contrastive representation learning (Eysenbach et al., 2022) or by giving the value function a quasimetric structure that reflects goal reachability (Liu et al., 2023). This thesis adopts the same goal-conditioning, with the current task τ playing the role of the goal; we use the term *task* rather than *goal* for semantic clarity. The value function is conditioned on τ through the augmented state $x = (s, \tau)$ (Chapter 3), so a single network represents value across the task distribution. Each task is still evaluated in isolation, however; none of these methods propagate value across task boundaries in a persistent stream, where the state left by one task changes the cost of the next — the preparedness this thesis targets.

2.1.4 Partial-Episode Bootstrapping

In our setting, a task has no deadline of its own: reaching its goal is the only completion criterion, so an agent could in principle take arbitrarily many steps to finish it. For tractable training, we cap each task at a fixed number of steps — a per-task step limit — and abandon the task if it is exceeded. This limit is artificial: it is imposed only for tractability and is not part of the task. The learning target at such a cutoff matters: treating it as a true terminal — setting the target to the reward alone, with no value beyond — would teach the agent that the world ends there, so the learned value would reflect the arbitrary cutoff length rather than the underlying continuing task.

Pardo et al. (2018) address this by distinguishing genuinely time-limited tasks, where the time remaining should be added to the state, from indefinite-horizon tasks that are only truncated for practical reasons. For the latter, they introduce *partial-episode bootstrapping* (PEB): at an artificial cutoff the transition is treated as non-terminal, keeping the terminal indicator $d = 0$ rather than setting it to 1,

$$y = \begin{cases} r, & \text{cutoff treated as terminal } (d = 1), \\ r + \gamma \max_{a'} Q(s', a'), & \text{partial-episode bootstrapping } (d = 0), \end{cases}$$

so it bootstraps the value of the state the cutoff reaches, exactly as for a non-terminal transition. In their single-task setting the episode ends and a new one begins on the *same* task; PEB changes only the learning target at the cutoff, not the task. This thesis applies the same bootstrap principle to two cutoffs: a periodic world reset, which plays the role of episode termination in Pardo's setting, and a per-task timeout, where — unlike their single-task setting — the persistent world continues and a fresh task is drawn. Chapter 4 gives their exact handling.

2.2 Structured Action-Value Functions

In many environments an action is not a single atomic choice but factors into several components: an action type together with one or more arguments, such

as which object to act on or where to place it (Chapter 5). Such a *factored action space* grows combinatorially in the number of components, so a single network output over the joint set of actions is impractical to learn. This section reviews the two architectural ideas the method combines to handle a factored action space: dueling networks, which separate a state's value from the advantages of individual actions, and branching networks, which factor the output over action components.

2.2.1 Dueling Networks

A dueling network decomposes the action value into a state-value term and an advantage term (Wang et al., 2016). Instead of estimating $Q(s, a)$ directly, it learns a single state value $V(s)$ together with a per-action advantage $A(s, a)$ and combines them as

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right).$$

The subtracted term is required for identifiability: without it, adding a constant to $V(s)$ and subtracting it from every $A(s, a)$ leaves Q unchanged, so the value and advantage streams could not be separated. Wang et al. (2016) subtract either the maximum or the mean advantage, with mean subtraction the more stable and common choice. Decoupling the two lets the network estimate a state's value without learning the effect of every action there, which helps in states where the choice of action barely changes the value. This thesis uses a mask-centered variant that subtracts the mean advantage over only the valid actions, which carries over directly to the factored action space described next.

2.2.2 Branching / Factored Action Spaces

A branching network handles a factored action space $\mathcal{A} = \prod_h \mathcal{A}_h$ by placing one output head per component rather than a single head over the joint space (Figure 2.1) (Tavakoli et al., 2018; Vinyals et al., 2019). Combined with the dueling decomposition, each component h contributes a mean-centered advantage over its

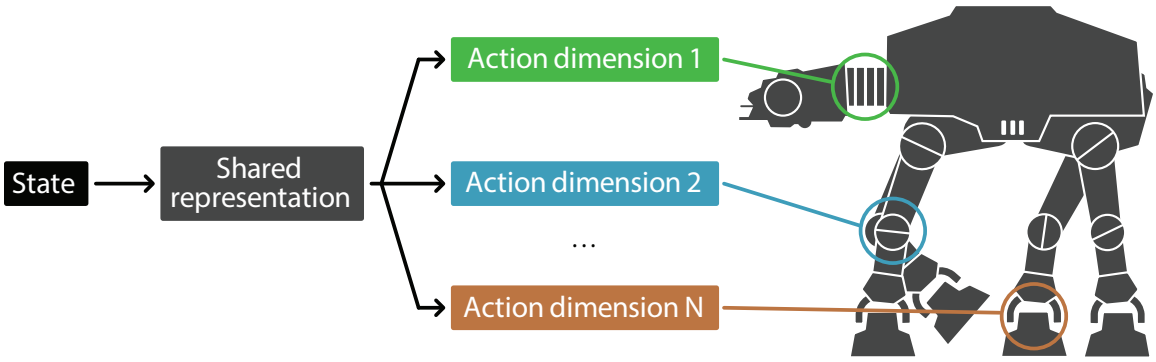


Figure 2.1: A conceptual illustration of the action branching network architecture: a shared state representation feeds a separate output head for each action dimension. Figure taken from Tavakoli et al. (2018).

own actions on top of a shared state value:

$$Q_h(s, a_h) = V(s) + \left(A_h(s, a_h) - \frac{1}{|\mathcal{A}_h|} \sum_{a'_h} A_h(s, a'_h) \right).$$

This factorization reduces the number of outputs from $O\big(\prod_h |\mathcal{A}_h|\big)$ for a joint head to $O\big(\sum_h |\mathcal{A}_h|\big)$, making a large factored action space tractable. A drawback of independent branching heads is that selecting each component by its own arg max can produce a suboptimal joint action when components interact, because the additive decomposition ignores cross-component dependencies (Landers et al., 2025). Conditioning the heads autoregressively — choosing components in a fixed order, each conditioned on those already selected — restores these dependencies. Chebotar et al. (2023) obtain an exact joint maximum this way by training a per-dimension Bellman decomposition in which each dimension maximizes over the next; our method instead recovers the joint arg max by enumerating valid action tuples directly (Chapter 4). Factored architectures of this kind perform well in practice (Flavin and Sen, 2026). Our method builds on this autoregressive branching dueling architecture, adding action masking that centers each advantage over only the valid actions; Chapter 4 gives the details.

2.3 Symbolic Planning

Classical symbolic planning represents the world with a discrete, factored state and a set of actions, each with preconditions and effects, written in a formal language such as STRIPS or PDDL (Fikes and Nilsson, 1971). A plan is a sequence of actions that transforms the initial state into one satisfying the goal. A planner searches the state space for such a sequence and can return either a cost-*optimal* plan or, more cheaply, a *satisficing* plan that reaches the goal without a cost guarantee. Fast Downward is a widely used heuristic-search planner supporting both modes through pluggable heuristics and search strategies (Helmert, 2006), and we use it for symbolic planning in this thesis.

The underlying search is typically best-first: nodes are expanded in order of a heuristic estimate of the cost to reach the goal, and A^* returns an optimal plan when that heuristic is admissible (Hart et al., 1968). A different strategy is width- or novelty-based search, which prioritizes states that make a previously unseen combination of facts true, exploring structurally new states rather than following a cost heuristic (Lipovetzky and Geffner, 2012). Bounded-suboptimal search sits between optimal and satisficing: it returns a solution whose cost is within a fixed factor of optimal, using a secondary criterion to choose among the solutions inside that bound (Pearl and Kim, 1982; Thayer and Ruml, 2011). This thesis uses symbolic planners as planning oracles and as the search that our learned value guides (Section 2.4).

2.4 Value-Guided Planning

A recurring idea is to pair a learned value function with search: the value scores the states a search reaches, and the search improves on the policy that the value alone would induce. This pairing underlies AlphaZero and MuZero, whose value networks evaluate the leaves of a Monte Carlo tree search rather than acting greedily (Silver et al., 2018; Schrittwieser et al., 2020), and expert-iteration methods that alternate between a search and the value it trains (Anthony et al., 2017).

Search is also the more reliable way to deploy an *imperfect* value. Acting greedily on an approximate value can incur a loss far larger than the approximation error, growing with the effective horizon (Singh and Yee, 1994), whereas a search needs the value only to rank the states it reaches. The same idea carries to symbolic planning, where a learned heuristic can guide or replace a hand-designed one (Ferber et al., 2020), improving search across task-and-motion planning (Chitnis et al., 2016), temporal planning (Micheli and Valentini, 2021), and bounded-suboptimal heuristic search (Spies et al., 2019).

Our method deploys the learned value in this way — as a heuristic for a symbolic search over ways of completing the current task, under a bounded-suboptimal cost constraint (Section 2.3) — as detailed in Chapter 4. Prior anticipatory-planning work also learns a value to guide symbolic planning, but offline and by supervised regression rather than online reinforcement learning; we review it next.

2.5 Anticipatory Planning

Dhakal et al. (2023) introduced *anticipatory planning*: rather than solving each task in isolation, the robot completes the current task while minimizing the expected cost of the next task drawn from the task distribution. The anticipated quantity is the anticipatory planning cost $C_{\mathrm{AP}}(s) = \mathbb{E}_{\tau' \sim P}[C^*_{\tau'}(s)]$, the expected optimal cost of one next task from state s (Chapter 3); a planner searches over ways of completing the current task and selects the terminal state that minimizes immediate plus anticipated cost.

This line of work shares an offline, planner-centric recipe. Because computing C_{AP} online is computationally prohibitive at deployment, Dhakal et al. (2023) train a graph neural network to estimate it from a dataset of states labeled by a symbolic planner, and the learned estimator then guides the planner at deployment. Dhakal et al. (2023) demonstrate this idea in a blockworld domain; Dhakal et al. (2024) extend it to continuous task-and-motion planning on a real robot; and Talukder et al. (2024)

scale it to large home and restaurant environments using a scene-graph network and focused sampling — the restaurant benchmark of this thesis is adapted from theirs. Most recently, Talukder et al. (2026) extend anticipation to multiple robots sharing a persistent environment, summing a per-robot estimate of expected future cost. Across these works the estimator is trained offline against a known, stationary task distribution and anticipates only a single next task — the limitations this thesis targets (Chapter 1).

A closely related idea appears in safe reinforcement learning: Krakovna et al. (2020) avoid side effects by rewarding states from which future tasks remain achievable, using an auxiliary term equal to the expected optimal value over possible future goals. This term has the same form as the preparedness value of this thesis, and is likewise learned with Bellman updates, either exactly or with a universal value function; it differs in that it is defined over a fixed, uniform set of hypothetical future goals rather than the actual task distribution, looks a single task ahead, and aims to prevent side effects rather than to minimize task-sequence cost.

A separate line predicts *which* future tasks will arrive rather than valuing the state for them. Arora et al. (2024) prompt a language model to anticipate the next tasks autoregressively, conditioned on a history of previous tasks, modeling the sequential dependence between tasks, and plan for them jointly; Singh et al. (2024) extend this to human–robot collaboration; Patel et al. (2023); Patel and Chernova (2022) instead learn to predict a person's routine object use for proactive assistance. These methods exploit correlations between successive tasks — the dependence this thesis does not model, since it draws tasks independently — but they require the future tasks to be predicted in advance, leaving a trap in place whenever a prediction is missed. This thesis takes the value-based route of the offline lineage but learns the preparedness value *online* by reinforcement learning, propagating it across task boundaries so that anticipation extends beyond a single next task without a planner-labeled dataset.

Chapter 3: Problem Formulation

We formalize the setting as an agent solving a stream of tasks in a persistent environment, where the state left by one task becomes the starting state of the next. Formally, the environment has a state space $\mathcal{S}$ and an action space $\mathcal{A}$, both of which we take to be discrete, as in prior anticipatory-planning work (Dhakal et al., 2023). The symbolic planner we later use to deploy the learned value (Chapter 4) operates over a discrete, atomic representation of the world, and our value network is factored over discrete action components. A task is a single goal, such as clearing a table or preparing coffee, completed by a sequence of lower-level actions. The agent is assigned a stream of tasks $\{\tau_k\}_{k\geq 1}$ drawn from a distribution P over a task space $\mathcal{T}$, presented one at a time: it receives τ_{k+1} only after completing (or failing) τ_k. Each task τ_k specifies a set of goal states $\mathcal{G}_{\tau_k} \subseteq \mathcal{S}$. Executing action a in state s incurs a cost $c(s, a)$. The same task can be assigned from any state, and hence the total cost of completing the task depends on the state it starts from. The environment is deterministic: within a task, executing action $a_t \in \mathcal{A}$ in state $s_{t-1} \in \mathcal{S}$ at step t leads to state $s_t = f(s_{t-1}, a_t)$, where $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function. The action incurs cost $c(s_{t-1}, a_t)$.

For a plan $\rho = (a_1, a_2, \ldots, a_n)$ that starts in s_0 and follows the deterministic transition $s_t = f(s_{t-1}, a_t)$, the total plan cost for task τ is

$$C_\tau(s_0) \;=\; C(\rho) \;=\; \sum_{t=1}^{n} c(s_{t-1}, a_t)$$

A plan solves task τ from state s if, starting from $s_0 = s$, it reaches a goal state $s_n \in \mathcal{G}_\tau$. The optimal cost of task τ from state s is the minimum cost over all such plans,

$$C^*_\tau(s) \;=\; \min_{\rho} \{C(\rho) : \rho \text{ solves } \tau \text{ from } s\},$$

and the optimal plan is the minimizer:

$$\rho^*_\tau(s) \;=\; \arg\min_{\rho} \{C(\rho) : \rho \text{ solves } \tau \text{ from } s\}\,.$$

At any step t, the agent receives a state s_t and the current task τ_t. We define an augmented state $x_t = (s_t, \tau_t) \in \mathcal{X}$, where $\mathcal{X} = \mathcal{S} \times \mathcal{T}$. For simplicity, we drop the term *augmented* from here and will refer to x as a state. We adopt the goal-augmented state formulation from Liu et al. (2022). Unlike the standard single-goal, episodic setting, our tasks arrive as a continuing stream. We define a continuing task-conditioned Markov Decision Process (MDP) as $\langle \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the transition kernel on $\mathcal{X}$, $\mathcal{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. When the agent takes an action a_t in state $x_{t-1} = (s_{t-1}, \tau_{t-1})$, it receives a reward r_t and transitions to $x_t = (s_t, \tau_t)$ according to the kernel $\mathcal{P}$. The reward combines a constant goal bonus $R_{\text{goal}} > 0$, paid when the action reaches a goal state, with the negated action cost:

$$r_t = R_{\text{goal}}\, \mathbf{1}\big[s_t \in \mathcal{G}_{\tau_{t-1}}\big] - c(s_{t-1}, a_t). \tag{3.1}$$

Thus $\mathcal{P}$ factorizes: the world evolves deterministically $s_t = f(s_{t-1}, a_t)$, while the task τ_t is piecewise-constant in t, and changes only when a goal state is reached:

$$\tau_t = \begin{cases} \tau_{t-1} & \text{if } s_t \notin \mathcal{G}_{\tau_{t-1}}, \\ \tau' \sim P & \text{if } s_t \in \mathcal{G}_{\tau_{t-1}}. \end{cases}$$

The stochasticity of $\mathcal{P}$ enters only at task boundaries, where the completed task is replaced by a fresh task $\tau' \sim P$ while the world state s_t persists.

In line with the standard RL definition, a policy $\pi : \mathcal{X} \to \Delta(\mathcal{A})$ maps each state $x \in \mathcal{X}$ to a probability distribution over the action space $\mathcal{A}$, with $\pi(a \mid x)$ the probability of taking action a in state x. The return G_t is the sum of discounted rewards accumulated by the agent from step t onward, beginning with the reward for the action a_t taken in state x_{t-1}.

$$G_t = \sum_{k=0}^{\infty} \gamma^k\, r_{t+k}, \quad \text{where } r_{t+k} = \mathcal{R}(x_{t+k-1}, a_{t+k}). \tag{3.2}$$

The action-value function Q^π of policy π is the expected return given the agent starts in state x, takes action a, then follows policy π:

$$Q^\pi(x,a) = \mathbb{E}_\pi\left[G_t \mid x_{t-1}=x, a_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k\, r_{t+k} \mid x_{t-1}=x, a_t=a\right] \tag{3.3}$$

The corresponding state value function is:

$$V^\pi(x) = \mathbb{E}_\pi\left[G_t \mid x_{t-1}=x\right]. \tag{3.4}$$

These value functions satisfy the Bellman equations. Writing $s' = f(s,a)$ for the next world state, we decompose the action-value function of policy π over whether the current task is completed:

$$\begin{aligned} Q^\pi\big((s,\tau),a\big) = -c(s,a) &+ \underbrace{\mathbf{1}[s' \notin \mathcal{G}_\tau]\,\gamma\, V^\pi(s',\tau)}_{\text{task continues}} \\ &+ \underbrace{\mathbf{1}[s' \in \mathcal{G}_\tau]\big(R_{\text{goal}} + \gamma\, \mathbb{E}_{\tau'\sim P}\big[V^\pi(s',\tau')\big]\big)}_{\text{task completed}}, \end{aligned} \tag{3.5}$$

with the state value recovered by averaging over the policy,

$$V^\pi(s,\tau) = \sum_a \pi(a \mid s,\tau)\, Q^\pi\big((s,\tau),a\big). \tag{3.6}$$

The optimal action-value function Q^* satisfies the corresponding optimality equation:

$$\begin{aligned} Q^*\big((s,\tau),a\big) = -c(s,a) &+ \mathbf{1}[s' \notin \mathcal{G}_\tau]\,\gamma \max_{a'} Q^*\big((s',\tau),a'\big) \\ &+ \mathbf{1}[s' \in \mathcal{G}_\tau]\big(R_{\text{goal}} + \gamma\, \mathbb{E}_{\tau'\sim P}\big[\max_{a'} Q^*\big((s',\tau'),a'\big)\big]\big). \end{aligned} \tag{3.7}$$

When the agent completes a task in state s, the next task $\tau' \sim P$ begins from s. We define the *anticipatory value* $V_{\text{AP}}(s)$ as the expected value of s under this task distribution:

$$V^\pi_{\text{AP}}(s) = \mathbb{E}_{\tau'\sim P}\left[V^\pi(s,\tau')\right], \tag{3.8}$$

$$V^*_{\text{AP}}(s) = \mathbb{E}_{\tau'\sim P}\left[V^*(s,\tau')\right]. \tag{3.9}$$

We also call V_{AP} the *preparedness value*. The two names denote the same quantity: the first names its role in the objective, the second what it measures about the

state. Substituting the optimal form into the completion branch of Eq. 3.7, a task-completing action ($s' \in \mathcal{G}_\tau$) has value

$$Q^*(x, a) = -c(s, a) + R_{\text{goal}} + \gamma\, V^*_{\text{AP}}(s'). \tag{3.10}$$

The anticipatory value V_{AP} is the reinforcement-learning counterpart of the *anticipatory planning cost* introduced by Dhakal et al. (2023). In our notation, their objective is the expected optimal cost of the next task drawn from the task distribution:[1]

$$C_{\text{AP}}(s) = \mathbb{E}_{\tau' \sim P}\left[C^*_{\tau'}(s)\right]. \tag{3.11}$$

Both quantities have the same form: an expectation over the next task $\tau' \sim P$ of a per-task optimum from s. They differ only in that optimum. In Eq. 3.11 the optimum is the cost $C^*_{\tau'}(s)$ of solving a *single* next task, so anticipation reaches exactly one task ahead. In Eq. 3.9 it is the value $V^*(s, \tau')$, which by the Bellman recursion of Eq. 3.7 unrolls into a discounted sum over the entire future task stream. Prior work is the one-task-ahead limit of this value: truncating the recursion after the next task and setting $\gamma = 1$ makes the value-optimal plan cost-optimal, giving $V_{\text{AP}}(s) = R_{\text{goal}} - C_{\text{AP}}(s)$. Ours differs in two ways — it discounts costs by when they fall, and, more importantly, it keeps the continuation, so *how* the next task is completed is chosen with the rest of the stream in view, which C_{AP} cannot express.

As introduced in Chapter 1, we define two types of agents: a *myopic* and an *anticipatory* agent. Both agents act in the same persistent environment, starting each task from the state in which the previous one ended. However, a myopic agent optimizes for the task it receives and tries to find the highest-reward action sequence to complete it, whereas an anticipatory agent finishes the task while accounting for the preparedness value V_{AP} of the resulting state. For example, consider that the task is to serve water. The shortest path – to fill a cup at the water fountain and serve it

[1] Dhakal et al. (2023) denote this quantity V, as it is a cost to be minimized. We reserve V for the value function (a return to be maximized) and write C for cost.

at the serving station – is what the myopic agent will take. The anticipatory agent, on the other hand, might take extra steps to fill a reusable jar, then fill the cup with the jar. This costs more now, but the filled jar spares repeated trips to the fountain on future water tasks. In the Bellman equation of Eq. 3.5, the reward plus the discounted next-state estimate forms the temporal-difference (TD) target the agent regresses toward. The two agents differ only in the target used at a task boundary. The anticipatory agent bootstraps *through* the boundary. On completion, its target retains the term $\gamma\, V_{\mathrm{AP}}(s')$, so credit propagates across the whole task stream, and it uses the optimality equation of Eq. 3.7 unchanged.

$$\begin{aligned} Q^*_{\mathrm{ant}}\big((s,\tau),a\big) = -c(s,a) + \mathbf{1}[s' \notin \mathcal{G}_\tau]\,\gamma \max_{a'} Q^*_{\mathrm{ant}}\big((s',\tau),a'\big) \\ + \mathbf{1}[s' \in \mathcal{G}_\tau]\big(R_{\mathrm{goal}} + \gamma\, \mathbb{E}_{\tau' \sim P}\big[\max_{a'} Q^*_{\mathrm{ant}}\big((s',\tau'),a'\big)\big]\big). \end{aligned} \tag{3.12}$$

The myopic agent instead treats task completion as terminal for bootstrapping: the $\gamma\, V_{\mathrm{AP}}(s')$ term is dropped, and its completion target is the goal reward alone:

$$\begin{aligned} Q^*_{\mathrm{myo}}\big((s,\tau),a\big) = -c(s,a) + \mathbf{1}[s' \notin \mathcal{G}_\tau]\,\gamma \max_{a'} Q^*_{\mathrm{myo}}\big((s',\tau),a'\big) \\ + \mathbf{1}[s' \in \mathcal{G}_\tau]\, R_{\mathrm{goal}}. \end{aligned} \tag{3.13}$$

Comparing Eq. 3.13 with Eq. 3.12, the anticipatory target is exactly the myopic target plus the discounted preparedness value $\gamma\, V^*_{\mathrm{AP}}(s')$ of the state left behind on completion. The two agents therefore correspond to two different episode structures over the *same* continuing environment dynamics. The myopic agent treats each task as a separate episode: task completion is an episode terminal with no continuation value. So the state s' carries no value to the myopic agent and it has no incentive to shape the world for what follows. The anticipatory agent runs a single continuing episode across the whole task stream, so completion bootstraps into the next task and the post-task state s' is valued through $V^*_{\mathrm{AP}}(s')$. This difference affects only *learning* and not the environment. The physical world is never reset at task boundaries for either agent, and both share the same environment, actions, rewards, and reset schedule.

The two agents differ only at *successful* completion ($s' \in \mathcal{G}_\tau$). If a task is not completed within a fixed step budget, it is abandoned and a new task is drawn;

the world state persists. This step budget is a practical rollout limit, applied identically during training and evaluation. Because the world does not end when a task is abandoned, the transition at the cutoff is not treated as terminal for learning. Following partial-episode bootstrapping (Pardo et al., 2018), the target bootstraps the persisting state under the *same, unfinished* task τ — not the newly drawn one — as $r + \gamma \max_{a'} Q\big((s', \tau), a'\big)$, for both the myopic and anticipatory agents. Bootstrapping τ rather than a completion means the target never includes the next-task term $\gamma V_{\mathrm{AP}}(s')$; the τ-trajectory then ends and a new one begins under the freshly drawn task, with no learning target bridging the two. Evaluation applies the same abandonment and resampling; an abandoned task counts as a failure in the reported metrics. It involves no learning target, so the bootstrapping choice does not arise there. For the anticipatory agent, the discount γ controls the preparedness horizon. Since a completing action bootstraps $\gamma\, V^*_{\mathrm{AP}}(s')$ and V^*_{AP} itself bootstraps the task after that, credit reaches across the whole task stream. Discounting is applied per step and not per task, so if the i-th future task takes T_i steps to complete, the weight on the k-th future task is $\gamma^{\sum_{i \le k} T_i}$. Future tasks are therefore discounted at an effective per-task factor $\gamma^{\bar{T}}$, where $\bar{T}$ is the mean number of steps per task, and this factor is smaller than γ whenever a task takes more than one action. As $\gamma \to 0$, the future-task term vanishes and the anticipatory target reduces to the myopic one; as $\gamma \to 1$, preparedness for the full stream is valued. We make this dependence precise in Chapter 4.

Chapter 4: Method

In the continuing task-conditioned MDP defined in Chapter 3, the agent seeks to maximize the discounted return G_t (Eq. 3.2), which rewards completing tasks and penalizes action cost. The return G_t cannot be optimized directly: it is an infinite sum, and computing it in closed form would require the transition function f and cost function c, which the agent does not have. Instead, each agent learns $\hat{Q}_\theta$ by Bellman bootstrapping, so credit propagates backward across steps and, for the anticipatory agent, across task boundaries. Figure 4.1 illustrates the agents' interaction with the environment and learning loop. We now describe how the value function is learned

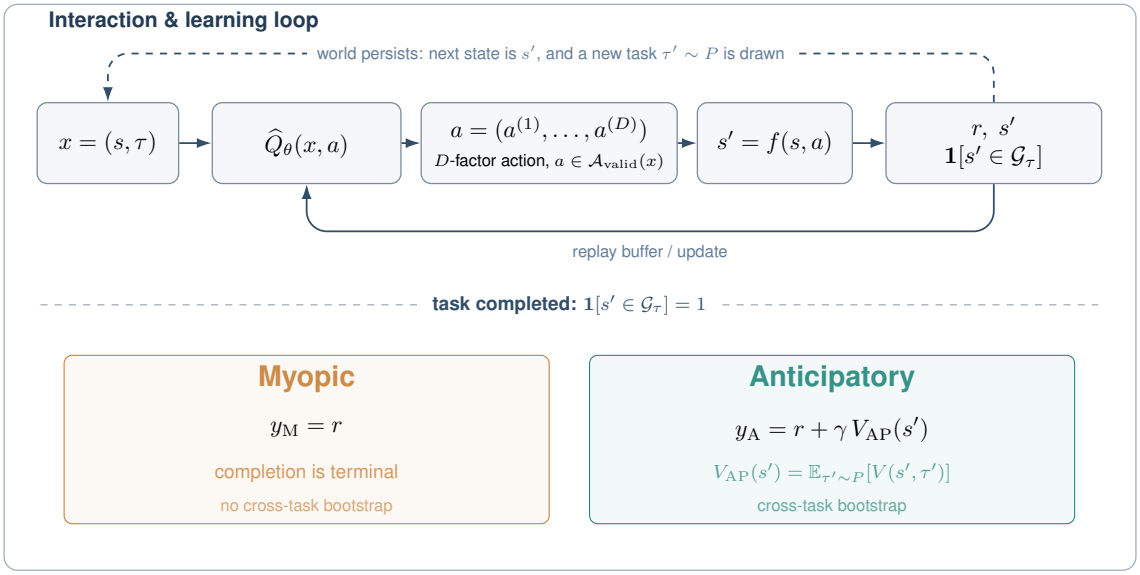


Figure 4.1: Interaction and learning loop. The two agents share the same interaction with the environment and differ only in the Bellman target at a completed task boundary: the myopic target treats completion as terminal, while the anticipatory target retains the preparedness value $V_{\text{AP}}(s')$ of the state left behind.

in Section 4.1, and how it guides a symbolic planner at deployment in Section 4.2.

4.1 Value-Function Approximation

We approximate Q^* by $\hat{Q}_\theta$ with a factored network trained by temporal-difference updates, with cutoffs handled so that the myopic and anticipatory agents differ only in credit horizon.

4.1.1 Network Architecture

As shown in Figure 4.1, a shared encoder maps the state $x = (s, \tau)$ to a latent representation. Because the task τ is part of the input, the same network is conditioned on the current task, which lets a single value function represent preparedness across tasks. Each action is parameterized by an action type together with object and location arguments—in our evaluation environment, for example, `place`(`cup`, `servingtable`), with components $\langle$`place`, `cup`, `servingtable`$\rangle$ (Chapter 5 gives the full action set). The joint action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ thus grows combinatorially, so a single output head over it would need $O\big(\prod_{i=1}^{k} |\mathcal{A}_i|\big)$ outputs. We therefore build on the branching dueling Q-network (BDQ) of Wang et al. (2016); Tavakoli et al. (2018), which places one output head per component—reducing the output dimensionality to $O\big(\sum_{i=1}^{k} |\mathcal{A}_i|\big)$—with two modifications suited to our domain. First, the heads are *autoregressive*: the components are ordered, and the head for component d takes as input the state embedding together with the embeddings of components $1, \ldots, d-1$, so the advantage it produces is a function of the partial tuple and not of the state alone. This is necessary because the valid values for a component, and its semantic role, can depend on earlier choices; selecting each component independently, as in the original BDQ, would produce invalid joint actions. For example, once the action type `pick` is chosen, the object head is restricted to objects that are actually pickable from the current state, and a location argument may no longer apply. This conditioning does not make selection sequential. Every valid tuple is scored, with that tuple's own earlier components supplied as input to its later heads, so a component's advantage is recomputed for each partial tuple it appears

in. Selection is then exact: we enumerate the action types and, within each type, maximize jointly over the grid of that type's valid arguments, so the greedy action is the arg max over scored tuples rather than a chain of per-component arg maxes. Selection is tractable because each action type takes at most two arguments, so the enumeration costs $\sum_{\text{types}} \prod_i |\mathcal{A}_i^{\text{type}}|$ rather than the full product over all components. Figure 4.2 (A) outlines this branching network. Second, we compose the component advantages into a *single* scalar action value: each component head produces an advantage $\bar{A}_d$, centered to zero mean over that component's mask-valid values, and these are summed onto a shared value output to give $Q(s, \tau, a)$. This composition differs from BDQ, which keeps a separate action value per branch trained by a temporal-difference error averaged across branches; here a single value trains against one temporal-difference target. Preparedness is a property of the state rather than of any action component. For a single task, the state value is $V(s, \tau) = \max_a Q(s, \tau, a)$; a state's preparedness is the anticipatory value $V_{\text{AP}}(s) = \mathbb{E}_{\tau' \sim P}[V(s, \tau')]$, the average of this state value over the tasks that may follow. Actions whose preconditions are unmet are masked during selection, so the agent only chooses valid actions. For instance, a `pick` action requires the agent and the object to be at the same location, so the agent cannot choose `pick(apple)` unless both are co-located.

4.1.2 Training Algorithm

Algorithm 1 trains $\hat{Q}_\theta$ with experience replay and Double-DQN targets (Mnih et al., 2015; Hasselt et al., 2016): the agent acts ϵ-greedily, stores each transition, and regresses Q_θ toward a bootstrapped target on sampled minibatches (lines 5–8, 20 of Algorithm 1). The replay buffer is seeded at initialization with a small set of demonstrations from a symbolic planner, and a fixed fraction of every minibatch is reserved for them to speed early learning (Appendix A.1). The myopic and anticipatory agents run the same loop and differ only in that target at a completed task boundary (lines 15 and 17; Eq. 3.13 vs Eq. 3.12): the myopic agent truncates to the reward, while the anticipatory agent adds the discounted anticipatory value

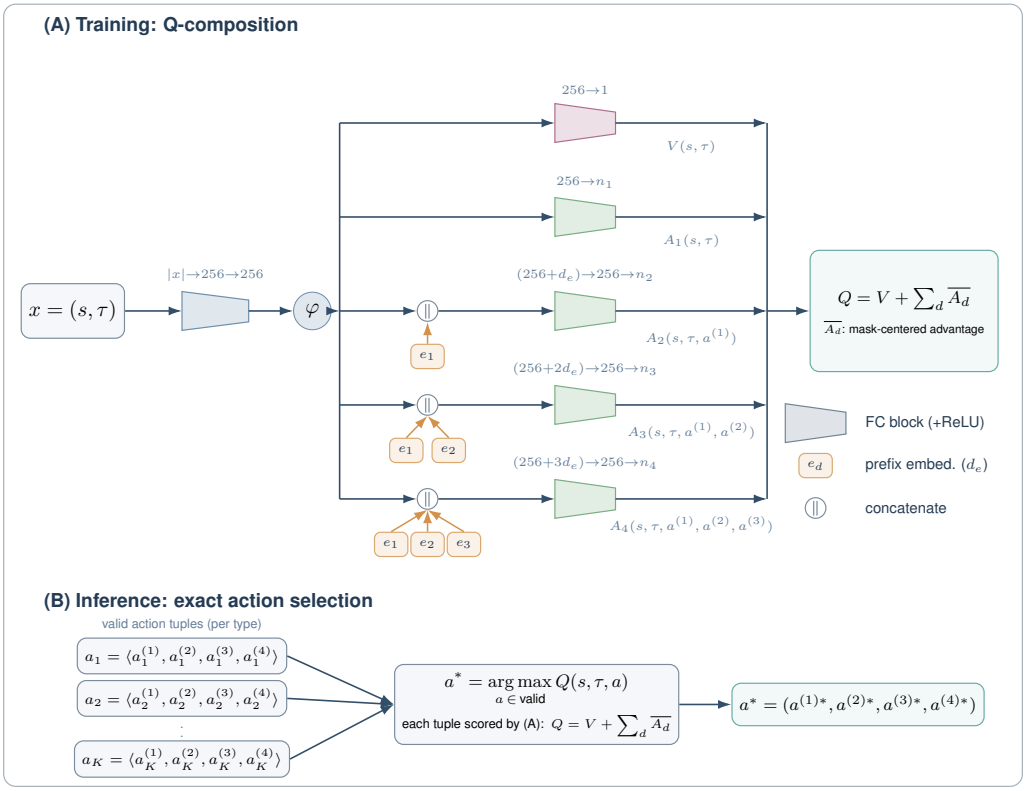


Figure 4.2: This figure shows the Branching network used for Deep Q-learning. **(A)** We generate a latent representation of the state x and use that representation to learn the state-value, one mask-centered advantage per action component, each conditioned on the state and the previously selected components. **(B)** We choose the action in a state by exact maximization over valid action tuples, with each component head conditioned on the components before it.

$\gamma\, V_{\mathrm{AP}}(s') = \gamma\, \mathbb{E}_{\tau' \sim P}[V(s', \tau')]$, computed exactly as a weighted sum.

Algorithm 1 Anticipatory / Myopic Factored Double-DQN

Initialize replay memory $\mathcal{D}$, online network Q_θ, and target network Q_{θ^-} with $\theta^- \leftarrow \theta$
Reset environment to state s; sample initial task $\tau \sim P$
for $t = 1, \ldots, T$ **do**
    $x \leftarrow (s, \tau)$
    Select valid action a from x using ϵ-greedy policy derived from Q_θ
    Execute a; observe reward r, next state s', and next task τ' ▷ $\tau' \sim P$ if τ completed or timed out, else $\tau' = \tau$
    Store transition (x, a, r, s') in $\mathcal{D}$
    Sample minibatch of transitions $B = \{(x_j, a_j, r_j, s'_j)\}$ from $\mathcal{D}$
    for each $(x_j, a_j, r_j, s'_j) \in B$ **do**
        Extract $(s_j, \tau_j) \leftarrow x_j$
        $a^*_{\tau'} \leftarrow \arg\max_{a'} Q_\theta(s'_j, \tau', a')$ for the relevant task τ' ▷ online-network greedy action
        if task τ_j is not completed **then**
            $y_j \leftarrow r_j + \gamma\, Q_{\theta^-}\big(s'_j, \tau_j, a^*_{\tau_j}\big)$ ▷ within-task or timeout
        else if myopic agent **then**
            $y_j \leftarrow r_j$
        else ▷ anticipatory agent
            $y_j \leftarrow r_j + \gamma \sum_{\tau'} P(\tau')\, Q_{\theta^-}\big(s'_j, \tau', a^*_{\tau'}\big)$
        end if
    end for
    Perform gradient descent step on θ minimizing $\frac{1}{|B|} \sum_j \mathcal{L}_{\mathrm{Huber}}\big(y_j - Q_\theta(x_j, a_j)\big)$
    $s \leftarrow s'$; $\tau \leftarrow \tau'$ ▷ Advance state and task
    if $t \pmod C = 0$ **then**
        $\theta^- \leftarrow \theta$
    end if
    if N tasks have been completed **then**
        Reset environment to a fresh state s and sample $\tau \sim P$
    end if
end for

4.1.3 Partial Episode Bootstrapping

A task carries no deadline of its own: reaching its goal is the only completion criterion, so in principle an agent may interact with the environment for arbitrarily

many steps before finishing. For tractable training, however, we cap each task at a fixed number of steps. Similarly, the persistent world is reset periodically for state diversity. Neither cutoff is part of the task, so following Pardo et al. (2018) we do not treat either as a terminal for learning; the transition at a cutoff bootstraps the value of the state it reaches, rather than being truncated to the reward alone.

We include two types of such cutoffs. **(a) Per-task timeout:** When a task exceeds the step limit, its final transition is stored with the same unfinished task and its target bootstraps that task, $y = r + \gamma \max_{a'} Q(s', \tau, a')$, so it estimates the value of the same task's counterfactual, untruncated continuation — the return had the artificial limit not cut it off. A fresh task τ' is then drawn and a new trajectory begins from (s', τ'). The stored transitions in the buffer are retained, but no target bootstraps across the cutoff. **(b) World reset:** When the world resets after a fixed number of tasks, the transition into the reset bootstraps the pre-reset state value, and a new trajectory begins in the fresh world. Both cutoffs are handled identically for the myopic and anticipatory agents: a timeout is not a task completion, so even the anticipatory agent bootstraps the same unfinished task rather than crossing into a new one. The agents therefore differ only in the credit horizon at task success.

4.2 Value-Guided Planning

The learned value $\hat{Q}_\theta$ can be deployed in two ways: directly, as a greedy policy that takes the highest-valued action at each step, $\arg\max_a Q(x, a)$; or as a heuristic for a symbolic search over ways of completing the current task. With an exact Q-function, greedy deployment would be optimal for the discounted-return objective; our cost-bounded terminal-selection rule is a different, more robust deployment heuristic and need not select the same completion. The learned value is approximate, however, and acting greedily on an approximate value can cost far more than the approximation error itself, by a factor that grows with the effective horizon (Singh and Yee, 1994). We therefore deploy it through the search, which is the more reliable vehicle for such

a value, for two reasons.

First, it *reliably returns a valid completion*: the Bellman–Novelty search itself is budget- and depth-limited and need not reach a completing state on its own, but it is backed by a classical symbolic planner that solves the current task and is seeded as a candidate before the search begins. Whenever that planner returns a plan within its time budget — as it did on every task in our experiments — a task-completing terminal is available, whereas a greedy policy can stall before finishing. Second, it *asks less of the value*. Completing a task in a well-prepared way is a multi-step investment: several actions that raise the immediate cost and pay off only over later tasks. A greedy policy, choosing one action at a time, must rank the investment above the cheaper alternative at *every* step, so a single misranking by the imperfect value abandons it. The search instead ranks the *completed* terminals only once, by their preparedness V_{AP}, reaching a well-prepared terminal—even one a step-by-step reading of the value would prune—through its novelty lane and planner fallback. The value need only judge which terminal is best prepared, not every action along a long investment; the longer the investment, the larger this difference. We quantify its magnitude in Chapter 5, taking the same value deployed greedily as the baseline against which the search is measured.

This use of a learned value—to score the terminal states a search reaches—is a standard way of combining a learned value with search, in which the value evaluates the leaf nodes of a tree search and the search improves on the policy that value alone would induce (Silver et al., 2018; Schrittwieser et al., 2020; Anthony et al., 2017). We apply the same principle with a best-first symbolic search over task completions rather than Monte Carlo tree search, scoring the terminal state each completion leaves behind. Prior anticipatory-planning work also learns a value to guide symbolic planning (Dhakal et al., 2023; Talukder et al., 2024), but trains it offline by supervised regression on planner-generated cost labels; our value is learned online by reinforcement learning (Section 4.1), needing no such dataset. The remainder of this section describes the search and how candidate terminals are scored and selected.

4.2.1 Bellman–Novelty Search

An anticipatory agent must jointly optimize over two objectives: completing the current task and reaching a state with high anticipatory value for what follows. These pull against each other, because the states that leave the world best prepared for future tasks are rarely the cheapest way to finish the current one. A search that only follows the value toward cheap completion therefore never reaches the better-prepared states. We resolve this by expanding states from two queues in a fixed round-robin schedule — three Bellman turns to one novelty turn (line 4 of Algorithm 2).

Bellman Lane. The first queue H_B is a best-first heuristic that follows the learned value toward completing the current task. A node at depth d with discounted prefix return G is prioritized by $G + \gamma^d \max_{a'} Q(s, \tau, a')$, so popping from it (line 4) expands the states the value estimates are closest to completing the task.

Novelty Lane. The second queue H_N is value-independent and rewards structurally new states, following the novelty measure of iterated width search (Lipovetzky and Geffner, 2012). A state's *novelty* is the size of the smallest set of facts that no state previously expanded *in this lane* made true — 1 for a brand-new fact, 2 for a new pair, and so on; the facts are recorded only when a state is expanded on a novelty turn (line 8). The lane expands lowest-novelty states first: introducing a brand-new fact is a more fundamental advance than merely realizing a new pair of familiar ones. This lane deliberately expands states that the Bellman lane would prune as expensive, such as a state in which a reusable container has been filled, and is what allows the search to reach well-prepared terminals.

Fallback. Before searching, we run a classical symbolic planner (Helmert, 2006) on the current task and, when it returns a plan, seed the candidate set with that terminal (line 1); its cost is the myopic reference C_{myo} for selection (Section 4.2.2). A complete

planner returns a valid plan whenever one exists given sufficient planning time, but under a finite time budget it may fail, in which case the candidate set starts empty and is filled by the search alone. In our experiments the planner always returned a plan within its budget, so a task-completing terminal was always available. Algorithm 2 shows the search procedure: a popped node that has already been expanded or exceeds the depth cap forfeits its turn (line 6); otherwise, its successors that complete the task are collected as candidates (line 11) and the rest are pushed back onto the two frontiers (line 13). When the expansion budget is spent, a cost-bounded selection returns the most-prepared eligible terminal (line 19). For readability the main-text version omits the duplicate-state bookkeeping, the depth cap that bounds open nodes, and the novelty-width recomputation at pop time; Appendix A.3 gives the full procedure.

4.2.2 Terminal Scoring and Selection

The search collects a set of candidate terminal states, each a different way of completing the current task. The SelectTerminal routine of Algorithm 2 then scores each by its preparedness for future tasks and selects one, as we now describe.

Scoring. A candidate terminal state s' is scored by its anticipatory value,

$$V_{\mathrm{AP}}(s') = \mathbb{E}_{\tau' \sim P}[V(s', \tau')],$$

which is the same weighted expectation over the task distribution used during training (Section 4.1). Before scoring, we apply the consumption that completing the task entails: a served coffee or water is consumed and no longer available for a future task. This matters for how an investment is valued. A consumed preparation (for example, brewing a coffee that the current task then serves) does not carry over, whereas a reusable one (for example, filling a jar that later refills many cups) remains in the state and raises V_{AP}; scoring the post-consumption terminal is what distinguishes the two.

Selection. Selection balances two competing objectives: completing the current task cheaply, and leaving the world well prepared. A Bellman-consistent score combines them into one quantity, $G+\gamma^{d-1}R_{\text{goal}}+\gamma^{d}\,V_{\text{AP}}(s')$, where $G \leq 0$ is the discounted prefix return (the accumulated step costs), d the depth of the completion, and R_{goal} the completion reward earned at the final step. We instead constrain the first objective and maximize the second, following the principle of bounded-suboptimal search, in which a secondary criterion chooses among solutions whose cost is within a fixed factor of optimal (Pearl and Kim, 1982; Thayer and Ruml, 2011). Let C_{myo} be the cost of the myopic reference terminal from the symbolic planner (if the planner returns no plan, we take the cheapest search terminal as the reference instead). Among candidates whose current-task cost is within $\beta\,C_{\text{myo}}$ for a fixed factor $\beta \geq 1$, we select the one with the highest V_{AP}:

$$s'^{*} = \arg \max_{s' \,:\, C(s') \leq \beta\, C_{\text{myo}}} V_{\text{AP}}(s'). \tag{4.1}$$

The factor β is the agent's budget for current-task overhead. At $\beta = 1$ it takes only preparation that costs no more than the myopic plan; larger β permits progressively more expensive investment. Because $\beta \geq 1$, the myopic reference is always eligible and a selection always exists. A combined score weighs cost against preparedness but imposes no such bound.

Constraining also makes selection insensitive to the scale of the learned value. The rule depends on V_{AP} only through the order it induces among eligible candidates, and is unchanged when the value is rescaled by a positive constant. A combined score adds V_{AP} to a known cost, so an error in its magnitude changes which terminal wins. Bootstrapped value estimates are prone to such inflation (Hasselt et al., 2016). The value ranks terminals; it does not price them against cost.

Algorithm 2 Bellman–Novelty Search. Two frontiers over one node set: H_B is value-guided, ordered by $G(n) + \gamma^{d(n)} q_{\max}(s_n)$, where $q_{\max}(s) = \max_a Q(s, \tau, a)$ scores the *current* task; H_N is value-blind, ordered by novelty width then depth. Completions are scored by $V_{\mathrm{AP}}(s) = \mathbb{E}_{\tau'}[\max_a Q(s, \tau', a)]$, an expectation over the *next* task. Appendix A.3 gives the bookkeeping omitted here.

Require: initial state s_0, current task τ, trained Q, budget B, depth cap $d_{\max}$, slack $\beta \geq 1$

$\mathcal{T} \leftarrow \{\textsc{FastDownward}(s_0, \tau)\}$ if a plan is found, else $\emptyset$ ▷ myopic reference
push s_0 onto H_B and H_N; $e \leftarrow 0$
while $e < B$ **and** either frontier is non-empty **do**
    schedule lane H_B on three of every four turns, else H_N; **if** the scheduled lane is empty **then continue** ▷ turn forfeited; no fall-through to the other lane
    $n \leftarrow$ its top node (PopMax for H_B, PopMin for H_N)
    if $\textsc{Stale}(n)$ **then continue** ▷ n re-expands a state at no higher G, or $d(n) \geq d_{\max}$ (Appendix A.3)
    $e \leftarrow e + 1$
    if the turn was H_N's **then** register s_n's facts ▷ before its successors, so their novelty is measured against facts already seen in this lane
    for all successors n' of n **do**
        if n' completes τ **then**
            $\mathcal{T} \leftarrow \mathcal{T} \cup \{n'\}$ ▷ a completion of τ
        else
            push n' onto H_B; onto H_N too if n' is novel
        end if
    end for
end while
$C_{\mathrm{myo}} \leftarrow$ cost of the myopic plan, or the cheapest terminal if none was found
$\mathcal{E} \leftarrow \{t \in \mathcal{T} : c_t \leq \beta\, C_{\mathrm{myo}}\}$ ▷ spend at most $\beta\times$ the myopic cost
return $\arg\max_{t \in \mathcal{E}} V_{\mathrm{AP}}(t)$, ties broken by smaller cost; NO-SELECTION if $\mathcal{E} = \emptyset$

Chapter 5: Experiments and Results

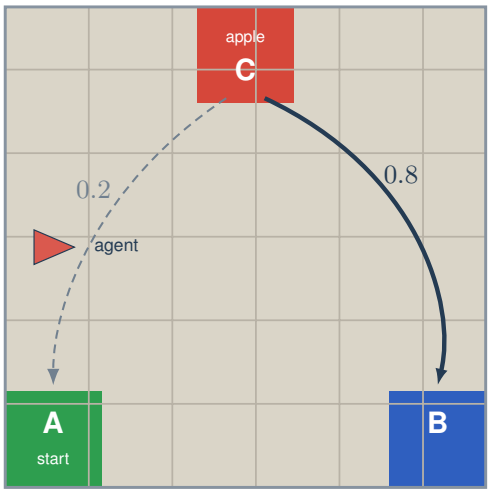
In this chapter, we compare different agents interacting with a persistent environment, and show the advantages of anticipation quantitatively and qualitatively. We show that over a sequence of tasks, anticipation reduces the total cost of completing the sequence, and that this reduction appears only when the anticipatory value is paired with value-guided deployment—neither the value nor the search alone is enough. We then ask how far ahead anticipation must reach, moving the reusable investment from a nearby shelf to a distant pantry to separate preparation that pays off within one task from preparation that pays off only across several. Finally, we show that learning the value online through reinforcement learning captures the anticipation signal as well as the offline, supervised estimator used by prior work, without its planner-generated dataset. The metrics that we use for comparison are: (1) mean planning cost per task; (2) number of actions taken per task; (3) success rate; (4) improvement over the myopic reference.

5.1 A Minimal Example

First, we illustrate the myopic–anticipatory distinction in a toy setting to give an intuition of how anticipation is captured in our method. Consider a 5×5 grid with three receptacles as shown in Figure 5.1a: A (bottom-left), B (bottom-right), and C (top-center), where an apple always spawns, and the agent starts on a random cell. The agent is given two tasks in sequence. **Task 1** is to clear receptacle C, and the agent should not be holding anything to complete the task, i.e., the agent needs to put the apple on either receptacle A or B. **Task 2** is to deliver the apple to a target drawn from a fixed distribution — B with probability 0.8, A with probability 0.2. If the apple already sits on the drawn target, task 2 is satisfied at no cost. The task-1 placement is therefore a preparedness decision: leaving the apple on B pre-satisfies

the 80%-likely task 2, at the risk of the 20% case. Each task completion earns +10, each step costs 1, and the discount factor $\gamma = 0.99$.

We train two agents, myopic and anticipatory, and they are identical except in the Bellman target at the task-1 boundary. Panel (b) of Figure 5.1 reports the



(a)

Agent	Q(place A)	Q(place B)	Δ
Myopic	9.0	9.0	−0.01
Anticipatory	12.5	17.5	+5.0

(b)

Figure 5.1: **(a)** An apple spawns at C; the agent places it on A or B (task 1), then delivers it to a target drawn to be B with probability 0.8 and A with probability 0.2 (task 2, arrows). **(b)** Learned Q of the two task-1 placements: the myopic values are equal to within approximation noise, while the anticipatory values prefer B.

learned Q-value of object placement on receptacle A or B for both agents. The myopic values are equal, and this is exactly what the severed target predicts. With completion treated as terminal, both placements are worth R_{task1} minus the step cost, i.e. $10-1 = 9$ for the myopic agent (independent of where the apple is left — matching the observed 9.0). A myopic value cannot distinguish the two futures, and the residual $\Delta = -0.01$ is approximation noise. The anticipatory values differ by $\Delta \approx +5.0$, and this gap is the discounted preparedness value. Both placements earn the same reward ($= 9$) on the placing action, so the two placement values are $9 + \gamma V_A$ and $9 + \gamma V_B$, giving $V_A \approx 3.5$ and $V_B \approx 8.6$. The gap is $\gamma(V_B - V_A) \approx 0.99 \times 5.1 \approx 5.0$. State B is worth more because it auto-satisfies the 80%-likely task 2, whereas A auto-satisfies

only the 20% case. The single task-1 boundary thus carries the entire anticipatory signal, and the gap is a direct, inspectable instance of the preparedness value V_{AP}. This value gap is reflected in behavior: under stochastic (softmax) evaluation, the anticipatory agent places the apple on B in 99% of episodes, while the myopic agent splits nearly evenly (47% on B). A and B are deliberately equidistant, so the two placements cost the same and the myopic agent splits evenly. Were A even slightly closer, it would place the apple on A every time — the cheaper placement — despite B being the 80%-likely target. Because the myopic agent chooses by immediate cost alone, an asymmetric example can make it almost always prepare for the wrong task.

5.2 Experimental Setup

This section describes the restaurant environment the agents act in, the agents and baselines we compare, the evaluation protocol, and how the completion reward is calibrated. We use two layouts of the restaurant environment called *two-room restaurant*, and *three-room restaurant*. The layouts are illustrated in Figure 5.2.

5.2.1 The restaurant environment

We instantiate the continuing task-conditioned MDP defined in Chapter 3 as a restaurant where an agent is deployed to perform tasks sequentially. It follows our definition of a persistent environment, i.e., the current task starts from the state in which the previous task ended. This environment is derived from the restaurant environment provided by Talukder et al. (2024), and uses the same action set: the agent can *move*, *pick*, *place*, *wash*, *fill*, *refill*, *pour*, *drain*, *make coffee*, and *make a fruit bowl*. There are six types of tasks:

1. washing an object
2. making coffee
3. serving water
4. clearing containers

5. making a fruit bowl
6. pick and place.

A task type may be instantiated with different arguments; a washing task, for instance, may target a bowl or a knife. The restaurant has two rooms, the kitchen and the serving area, with six locations in total. The kitchen holds the countertop, coffee machine, dishwasher, and shelf; the serving area holds the serving table and the water fountain. We call this domain *2-room restaurant*. See Figure 5.2 (a) for an illustration of the restaurant layout. A jar, the most useful object for amortization in our experiments, starts on the shelf. For the coffee or serving water tasks, an agent can choose to fill a cup with water every time from the fountain, or fill the jar once, and fill the cup from the jar at a lower cost. Filling the jar saves cost because filling a cup from the jar is cheaper, and the agent can move the jar to avoid trips to the fountain. When optimizing for a single task, the optimal plan is to directly fill the cup at the fountain, since there is no advantage for filling the jar, but if an agent can anticipate future tasks, it can choose to pick the jar. We introduce another variant of this domain, called the *3-room restaurant*, where we add a third room called "pantry" that is much further from the kitchen and serving rooms. In this domain, the jar starts in the pantry (see Figure 5.2 (b)).

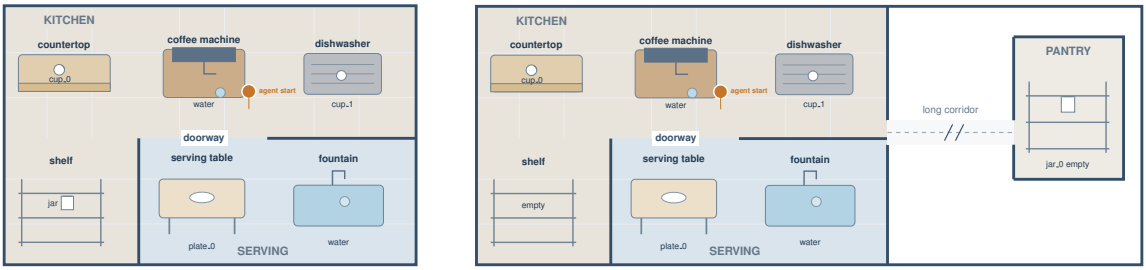
We compare the performance of the agents in these two domains to see the advantage of the horizon of anticipation. All experiments use these two layouts: the rooms, locations, and objects are fixed, and only the object states and the task sequence vary between runs. Tasks are sampled i.i.d. from the fixed distribution $P(\tau)$ in Table 5.1.

Task type	Wash	Coffee	Water	Clear	Fruit bowl	Pick-place
$P(\tau)$	0.30	0.30	0.25	0.05	0.05	0.05

Table 5.1: The task distribution $P(\tau)$ used for training and evaluation.

Movement follows a fixed inter-location cost matrix, while all other actions

have a fixed scalar cost. Moves within a room are cheap (base distance 1–3), while moving between the kitchen and the serving area is far more expensive (base distance 7–9). In the three-room restaurant, the pantry is farther still: reaching it from any other location costs 35.[1] As defined in Chapter 3, the reward for each action is $-c(s_{t-1}, a_t)$, and task completion gives a fixed reward, R_{goal}. The magnitude of R_{goal} is calibrated, as explained in Section 5.2.4. Delivered items are consumed on completion — a served coffee or water is emptied — so each task requires genuine re-preparation rather than reusing a standing artifact.



(a) Two-room Restaurant (b) Three-room Restaurant

Figure 5.2: Restaurant layouts used in the experiments.

5.2.2 Agents and baselines

We compare the performance of the following agents. The first two are non-learning, planning references that bound achievable performance; the third reproduces the prior work of Talukder et al. (2024), and the fourth is an extension of the third.

1. **Myopic Oracle**: Solves each task independently with a symbolic planner: A* search (Hart et al., 1968) with the blind heuristic, as implemented in Fast Downward (Helmert, 2006). Given enough search time, A* returns a plan whenever one exists. The blind heuristic is admissible ($h \equiv 0$), so every returned plan is

[1] These are base inter-location distances; the movement cost that enters the reward and the reported metric scales them by the travel factor for each cost scale (RL and PDDL, respectively), as with every other action cost.

cost-optimal *for the given task*. It has no notion of future tasks and therefore serves as the *no-anticipation* reference.

2. **Clairvoyant Oracle**[2]: Plans with perfect knowledge of the next K tasks, using a symbolic planner to minimize their joint cost, but commits only the current task before advancing to the next. Acting on future knowledge no real agent has, it estimates how much anticipation could save in principle. Its joint search is satisficing rather than optimal (Helmert, 2006), so the reported cost is an approximate ceiling — an upper bound on the true clairvoyant optimum.

3. **One-task GNN**: Our reproduction of the offline baseline of Talukder et al. (2024); Dhakal et al. (2023, 2024). A graph neural network is trained by supervised regression to estimate the anticipatory planning cost $C_{\mathrm{AP}} = \sum_{\tau'} P(\tau') C^*_{\tau'}$ (Eq. 3.11) (Talukder et al., 2024) — the expected optimal cost of a single next task, computed by solving every task in the known distribution. At plan time it scores candidate completions of the task by immediate cost plus estimated anticipatory cost, and executes the cheapest. It represents *one-task* anticipation, so investments whose payoff spans several future tasks cannot be amortized. Following Talukder et al. (2024), we generate the training set by sampling tasks from $P(\tau)$, solving each with the myopic planner, and labeling the states visited along these plans with their anticipatory planning cost. As a result, the training data does not contain any states that the myopic planner does not consider, for instance, a state in which the jar has been filled in another room. We make another variant of this baseline by augmenting the training dataset with additional, counterfactual states.

4. **One-task GNN (augmented)**: Identical to the One-task GNN, but the train-

[2]A distributional Anticipatory Oracle that plans against the distribution $P(\tau)$ rather than known future tasks would be the theoretically ideal reference, but computing it requires expectations over exponentially many task sequences and is intractable; we therefore use the clairvoyant variant as a computable approximate ceiling.

ing states are augmented with counterfactual *prepared* states that the myopic plans never visit — for example, states in which the jar has been filled and carried. Each added state is labeled with its expected next-task cost in the same way. This variant tests whether the baseline's limitation is one of state *coverage* rather than of the one-task objective itself.

The remaining four agents come from two learned value functions, each deployed in two ways. Their classification depends on two factors: (1) credit horizon (myopic/anticipatory); (2) deployment (greedy/guided). The value is trained with either a one-task (myopic) or a multi-task (anticipatory) credit horizon, and is then used either greedily or as the terminal heuristic of the cost-bounded search (Section 4.2). All four share the same factored network architecture.

5. **Myopic Value Function**: Trained with a one-task credit horizon (Equation 3.13), which treats task completion as terminal, so no credit propagates across a task boundary.

   5.1. **Myopic RL (greedy)**: The value defines the policy implicitly: at each step the agent acts greedily with respect to it, $\pi(s, \tau) = \arg\max_a Q(s, \tau, a)$.

   5.2. **Myopic RL (guided)**: The value is used to score the candidate terminal states of the cost-bounded search.

6. **Anticipatory Value Function**: Trained with a multi-task credit horizon (Equation 3.12), whose target bootstraps across task boundaries, so the value includes the expected return of the tasks that follow.

   6.1. **Anticipatory RL (greedy)**: The value defines the policy in the same way, acting greedily at each step.

   6.2. **Anticipatory RL (guided)**: (*Our method.*) The value is used to score the candidate terminal states of the same search, combining a multi-task credit horizon with value-guided planning.

A symbolic planner returns a valid plan whenever one exists, provided enough planning time. We give each planner-based agent a generous time budget, and in our experiments every such agent solved every evaluation task, so their success rates are 100%; a planner that exceeded its budget would count as a failed task, exactly as for the greedy agents. In practice, only the two greedy agents (5.1, 6.1), which act step-by-step from the value network, fell below 100%.

5.2.3 Evaluation Protocol

Every agent is evaluated on both the two-room and three-room restaurants. Within each restaurant, all agents see the same ten 50-task sequences, drawn i.i.d. from $P(\tau)$, so differences in cost reflect the method rather than the task stream. Within a sequence the world persists and is re-sampled only between sequences, so preparedness accumulates over 50 tasks before being reset. Each task is allotted 64 primitive steps; a task not completed within that budget is abandoned and counts as a failure (Chapter 3). We train each RL value function at discount $\gamma = 0.97$, average each seed over the ten sequences, and report the mean and standard deviation across seeds; the per-domain seed counts are given in Appendix A.1. The planning references are deterministic, so they carry no seed spread. We report: (1) cost per task, in PDDL cost units; (2) actions per task; (3) success rate; (4) improvement over the myopic oracle, $(C_{\text{oracle}} - C)/C_{\text{oracle}}$. The symbolic planner represents each valid primitive-action cost as $c_{\text{PDDL}}(s, a) = 100\, c_{\text{RL}}(s, a)$, where c_{RL} is the action cost c used in Equation 3.1. Consequently, for any valid plan ρ, $C_{\text{PDDL}}(\rho) = 100\, C_{\text{RL}}(\rho)$. We report C_{PDDL}; rewards and completion bonuses remain in RL units. The myopic oracle and the One-task GNN plan optimally, with A* under the blind heuristic. The clairvoyant oracles at $K = 3$ and $K = 4$ use a satisficing planner with 600 seconds per planning window, so their reported costs are upper bounds on the true optimum; the exact $K = 2$ oracle uses A* with an admissible heuristic and reaches optimality within its budget. The value-guided search calls Fast Downward with a 60-second timeout, and expands at most 20,000 states up to a depth of 20. The cost bound β

of Section 4.2.2 is shared by both value-guided agents: the myopic and anticipatory values are deployed through the same cost-bounded search under the same β, so any difference between them reflects the value function alone. It is fixed at 1.25 on the two-room restaurant and 3.0 on the three-room, where the costlier pantry investment needs a larger current-task budget to be admissible; the three-room value was selected on three development sequences (Appendix A.1).

5.2.4 Reward Calibration

Equation 3.1 defines $r_t = R_{\text{goal}}\,\mathbf{1}\big[s_t \in \mathcal{G}_{\tau_{t-1}}\big] - c(s_{t-1}, a_t)$. Here $c = c_{\text{RL}}$ is the deterministic action cost used for training, with $c_{\text{RL}} = c_{\text{PDDL}}/100$, and R_{goal} is a fixed scalar in the same RL units; we set it to 81.07 for the two-room and 74.01 for the three-room restaurant. Here, we describe the calibration process and show it using the values for the two-room restaurant. The process is equivalent for the other domain. Since every action incurs a negative reward, the completion reward must be large enough that finishing a task is preferable to abandoning it. If R_{goal} is too small, the agent learns to deliberately fail expensive tasks, because letting the step budget expire costs less than completing them. We therefore pick R_{goal} so that executing any plan in an oracle sample yields non-negative discounted return. For a plan ρ of length T for task τ, whose completing action is taken at step T:

$$R_{\text{goal}} \geq \sum_{t=1}^{T} \gamma^{\,t-T} c_t =: D(\rho, \tau).$$

Note that $D(\rho, \tau)$ depends on γ, and we run calibration at $\gamma = 0.95$. We run a persistent myopic oracle (albeit satisficing) over 2,000 tasks, execute its plans in the environment with $R_{\text{goal}} = 0$, and compute D for each. The most expensive plan gives $D_{\max} = 67.56$. We then set:

$$R_{\text{goal}} = (1 + m) \times D_{\max} = 1.2 \times 67.56 = 81.07$$

with margin $m = 0.2$ hedging against the fact that a 2,000-task sample may miss rarer expensive tasks. A much larger R_{goal} would also satisfy this bound, but because

the completion reward is discounted by γ^{T-1}, a large value makes the agent prefer plans that finish in fewer steps over plans that cost less. We therefore pick a value close to the lower bound.

This calibration is deliberately conservative. Since $\gamma^{t-T} > 1$ for $t < T$, the requirement D grows as γ falls, so calibrating at $\gamma = 0.95$ imposes a strictly larger bound than the $\gamma = 0.97$ used in our experiments. The satisficing planner used during calibration returns plans at least as expensive as optimal ones. The value 81.07 therefore exceeds what the reported runs require.

5.3 Deploying the Learned Value

Section 5.2.2 describes the two ways of deploying the learned value functions: greedily by taking the highest-valued action at each state (Agents 5.1, 6.1), and by using the value to guide a cost-bounded search (Agents 5.2, 6.2). As mentioned in Section 4.2, guiding a symbolic planner with the learned value is a more reliable way to act on an imperfect value than acting greedily. Here, we measure the difference between the two.

The greedy points in Figure 5.3 and Table 5.2 show the results of evaluating the greedy policy on our evaluation sequences. Both greedy and guided anticipatory agents converge to a high success rate ($\geq 93\%$) on both two- and three-room restaurants, showing that the greedy policy learns to solve the given task. On the two-room restaurant, the two greedy agents perform closely: the myopic agent takes slightly fewer steps per task, and the two cost about the same.

Under value-guided deployment, the same two value functions instead score the terminal states of the search. On the two-room restaurant, the anticipatory value falls from 1,432 to 914 cost per task under guided deployment, a 36% reduction obtained by changing only how the value is deployed, not the value itself. Under the identical search and budget, the myopic value improves as well but far less, from 1,427 to 1,196.

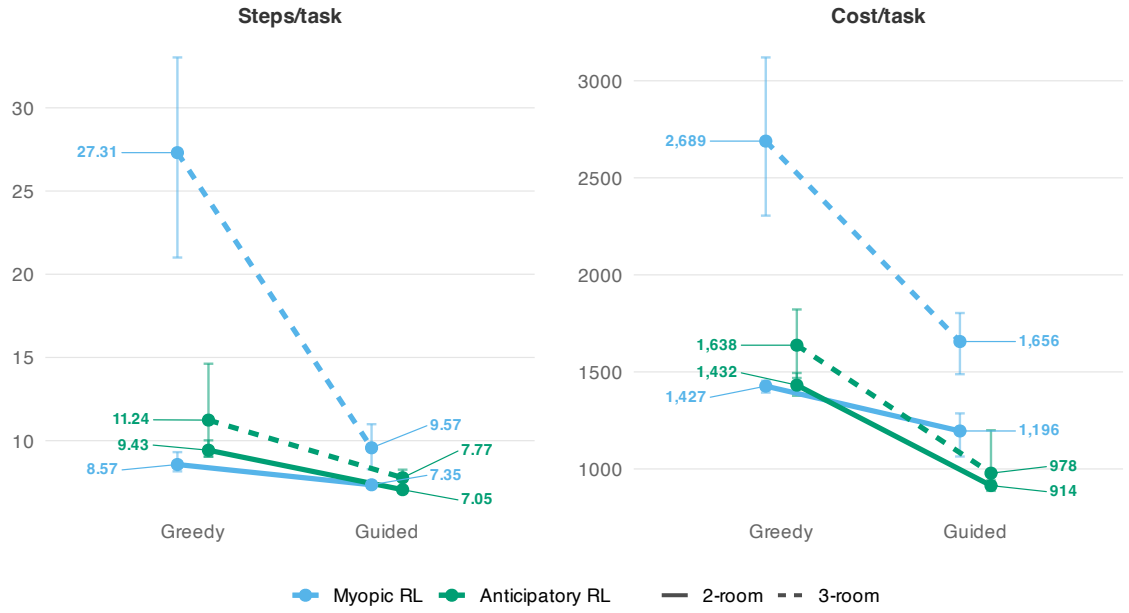


Figure 5.3: Greedy and value-guided evaluation of the two learned value functions on the two-room and three-room restaurants. Points are means across checkpoint seeds (each seed averaged over ten canonical 50-task sequences, 500 tasks per seed) and error bars span the minimum and maximum across seeds. Greedy agents do not always complete the task; their reported cost includes penalties for abandoned tasks, so part of the greedy-to-guided reduction is success recovery rather than a pure deployment effect. Detailed values are given in Appendix A.

We qualitatively observe that the greedy rollout often takes redundant actions — for example, picking up an object and setting it back down, or moving between locations without making progress — which the search eliminates. This accounts for the shorter guided plans, and is unrelated to preparedness.

When we compare the myopic and anticipatory agents under value-guided deployment, the steps per task are close (about 7 each), yet the average cost is much lower for the anticipatory agent (23.5% lower). Steps and cost decouple: the cheaper plan reflects the preparedness the myopic value lacks, not merely a shorter one. The anticipatory agent completes each task in the same number of actions, but takes cheaper ones. The difference in cost between value-guided and greedy deployment shows that the anticipatory value function contains the preparedness signal, which the greedy agent is unable to exploit. Because of this difference in plan cost, we use the value-guided deployment as our primary method and compare results with other baselines in Section 5.4.

5.4 Results

We now compare our results against the planning references and the prior-work baseline of Section 5.2.2. Table 5.2 reports the mean cost per task for every agent; the full cost distributions are in Appendix A.4 (Figure A.4).

5.4.1 Planning References

The myopic oracle solves each task cost-optimally without anticipation, so its 1,340 per task on the two-room restaurant and 1,349 on the three-room restaurant are the reference against which we measure the other agents.

The clairvoyant oracle plans with exact knowledge of the next K tasks and commits only the current one, giving a computable ceiling on how much anticipation can save. It is not a deployable agent: it uses future-task knowledge no real agent has, and computing it is expensive enough that, for tractability, it plans with a satisficing

Method	Mean cost/task	% vs Myopic Or.	Success
2-room			
Myopic Oracle	1,340	— (0%)	100%
Myopic RL (greedy)	$1{,}427 \pm 26$	−6.5%	$97.6 \pm 1.0\%$
Myopic RL (guided)	$1{,}196 \pm 108$	+10.8%	100%
One-task GNN	$1{,}341 \pm 12$	−0.1%	100%
One-task GNN (augmented)	945 ± 4	+29.5%	100%
Anticipatory RL (greedy)	$1{,}432 \pm 49$	−6.9%	$96.0 \pm 0.9\%$
Anticipatory RL (guided)	$\mathbf{914 \pm 19}$	**+31.8%**	100%
Clairvoyant Oracle (K=2)	809	+39.6%	100%
Clairvoyant Oracle (K=3)	782	+41.6%	100%
3-room			
Myopic Oracle	1,349	— (0%)	100%
Myopic RL (greedy)	$2{,}689 \pm 316$	−99.3%	$63.8 \pm 8.8\%$
Myopic RL (guided)	$1{,}656 \pm 131$	−22.8%	100%
One-task GNN	$1{,}342 \pm 6$	+0.5%	100%
One-task GNN (augmented)	$1{,}308 \pm 1$	+3.1%	100%
Anticipatory RL (greedy)	$1{,}638 \pm 136$	−21.4%	$93.3 \pm 3.6\%$
Anticipatory RL (guided)	$\mathbf{978 \pm 127}$	**+27.5%**	100%
Clairvoyant Oracle (K=2)	1,229	+8.9%	100%
Clairvoyant Oracle (K=3)	967	+28.3%	100%
Clairvoyant Oracle (K=4)	943	+30.1%	100%

Table 5.2: Main results on the canonical evaluation for both restaurants: ten persistent 50-task sequences, mean ± std across seeds. Improvement is measured against each restaurant's own myopic oracle. On the three-room restaurant the anticipatory bound $\beta = 3$ was selected on three of these ten sequences, so the three-room anticipatory guided figures include development sequences; the held-out result on the remaining seven is reported in Section 5.4.3. The clairvoyant oracle is solved optimally at K=2 and with satisficing search at $K \geq 3$ (600 s per planning window); the $K \geq 3$ costs are therefore upper bounds on the true clairvoyant optimum, not exact. Of these windows, 32.8% reach the time cap in the two-room restaurant (K=3), and 33.6% and 53.2% in the three-room restaurant (K=3 and K=4).

rather than an optimal planner, so its reported costs are upper bounds on the true clairvoyant optimum. A deployable agent has neither its foresight nor its compute budget, so we learn an anticipatory value from experience instead.

We use the reduction in clairvoyant cost as K grows to measure how much anticipation each environment rewards. On the two-room restaurant, one task of lookahead ($K = 2$) already achieves a 39.6% reduction, and a second ($K = 3$) adds almost nothing, reaching 41.6%. The ceiling is thus reached with about one task of lookahead, leaving little for a longer horizon to exploit — which is what motivated the three-room variant. On the three-room restaurant the picture is different: $K = 2$ reduces cost by only 8.9%, but $K = 3$ reaches 28.3%, a large gain from the extra task of lookahead, and $K = 4$ reaches 30.1%.

These figures approximate the ceiling rather than fix it exactly. The $K = 2$ oracle is solved optimally, whereas the $K = 3$ and $K = 4$ oracles use the satisficing planner, so the larger-K reductions understate what an optimal planner would achieve. The satisficing planner also degrades as the horizon grows: with a 600-second budget per planning window, 33.6% of windows time out at $K = 3$ and 53.2% at $K = 4$. The near-equal $K = 3$ and $K = 4$ reductions are therefore consistent with the anticipation horizon saturating near two tasks, but we cannot rule out that a faster planner would extract more at $K = 4$.

5.4.2 Anticipatory RL

Our primary agent is the anticipatory value function deployed through the cost-bounded search (Section 5.3); we report where it lands between the myopic oracle (the no-anticipation floor) and the clairvoyant oracle (the ceiling). On the two-room restaurant it reaches 914 per task, a 31.8% reduction over the myopic oracle, capturing 76% of the clairvoyant ceiling. On the three-room restaurant — the domain that requires anticipation beyond a single task — it reaches 978 per task, a 27.5% reduction, capturing about 97% of the $K = 3$ clairvoyant saving. In both

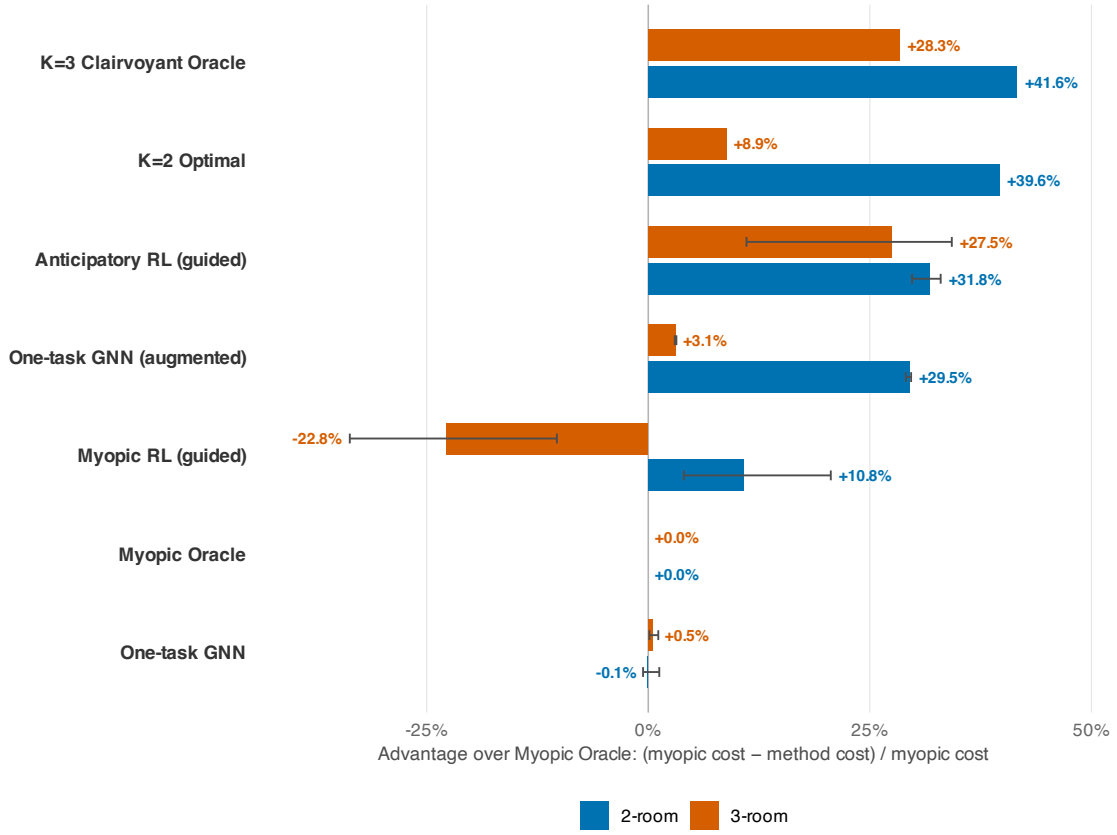


Figure 5.4: Cost reduction over the myopic oracle for each agent, on the two-room and three-room restaurants.

restaurants, then, the learned value recovers most of what an oracle with exact future knowledge achieves. The myopic value does not: under the same guided deployment its three-room cost (1,656) sits 22.8% *above* the myopic oracle, because the myopic value is miscalibrated on this domain (Appendix A.2) and guides the search toward terminals that are cheap for the current task but poorly prepared for the next.

5.4.3 Comparison with prior work

For both the two-room and three-room restaurants, the one-task GNN performs nearly identically to the myopic oracle (-0.1% and $+0.5\%$ respectively). This behavior is expected, because as discussed in Section 5.2.2, the training data for GNN

lacks the states that help task anticipation. However, our counterfactual re-creation, One-task GNN (augmented), performs differently. On the two-room restaurant, the One-task GNN (augmented) achieves a 29.5% reduction in cost over the myopic oracle, which is only slightly lower than our guided anticipatory RL agent. This matches our previous analysis that the two-room restaurant plateaus at one-task lookahead, and hence, a supervised learning baseline is able to capture the anticipation signal. Figure 5.4 shows the relative improvement over the myopic oracle for both the restaurant environments.

On the three-room restaurant, One-task GNN (augmented) reduces cost by 3.1% against the myopic oracle, whereas Anticipatory RL (guided) reduces it by 27.5%. Averaged over training seeds, Anticipatory RL (guided) is cheaper than the augmented GNN on all ten paired evaluation sequences (exact two-sided Wilcoxon signed-rank test, $p = 0.002$, the smallest value attainable at $n = 10$). Aggregating by seed instead — each seed averaged over the ten sequences — Anticipatory RL's mean cost is 25.2% lower than the augmented GNN's (Welch's $t(4) = 5.78$, $p = .004$; 95% CI for the reduction 13.1%–37.3%).

The three-room anticipatory bound $\beta = 3$ was selected on three of these ten sequences (Appendix A.1), so the all-ten figures above include those development sequences. On the seven sequences held out from the β choice (350 tasks), the anticipatory guided agent costs 1,038 per task against the myopic oracle's 1,403 — a +26.0% reduction, and 22.8% cheaper than the augmented GNN — essentially matching the all-ten values (+27.5% and 25.2%). It is cheaper than the augmented GNN on all seven held-out sequences (exact two-sided Wilcoxon signed-rank test, $p = 0.0156$). The advantage is therefore not an artifact of the sequences used to choose β.

Figure A.4 shows the full cost distributions: on the two-room restaurant our agent and the augmented GNN overlap, whereas on the three-room restaurant our agent's distribution sits clearly below the augmented GNN's. This result addresses limitation L3 from the introduction: even when the one-task GNN is given the pre-

pared states, its one-task objective cannot justify the distant-jar investment, whereas our agent's multi-task credit horizon can. The saving concentrates exactly where a reusable investment exists — the water tasks that draw on the jar — and is negligible on tasks without such structure, as the preparedness mechanism predicts (Figure A.6).

Beyond the cost reduction, our method also differs from the GNN baseline in how much planning it requires. The one-task GNN is trained by supervised regression on offline labels $C_{\mathrm{AP}}(s) = \sum_{\tau'} P(\tau')C^*_{\tau'}(s)$, and every label requires solving each task in the distribution from state s. With $|\mathcal{T}| = 48$ task instantiations in this domain, a dataset of N states costs $N|\mathcal{T}|$ planner invocations. The faithful dataset ($N = 2{,}000$) therefore consumed 9.6×10^4 Fast Downward calls, and the augmented dataset ($N = 8{,}054$) approximately 3.9×10^5. Our agent needs no such dataset: it learns the value online from 5×10^5 environment transitions, supervised only by the reward of Equation 3.1. This difference in cost addresses the limitation L2 discussed earlier.

5.4.4 Value–Search Ablation

The guided Anticipatory RL agent differs from the One-task GNN baseline in two factors: the credit horizon of the learned estimator, and the cost-bounded search used to deploy it. To separate the two, we run an ablation that uses our Bellman–Novelty search (BN search) and changes only the value, substituting the augmented one-task GNN value (Section 5.4.3) for the anticipatory value on the three-room restaurant. Table 5.3 decomposes the improvement. Deploying the one-task value through our search reaches 1,187 (+12.0%), better than its native one-step rule (1,308) but well short of the anticipatory value under the identical search (978). Holding the search fixed, the anticipatory value accounts for a 209-unit reduction; the search itself, holding the value fixed, accounts for 121. The multi-task credit horizon is thus the larger factor, and it cannot be recovered by giving the one-task value a better search.

Configuration	Mean cost/task	% vs Myopic Or.
GNN value, native selection	$1{,}308 \pm 1$	+3.1%
GNN value, BN search	$1{,}187 \pm 41$	+12.0%
Anticipatory value, BN search	$\mathbf{978 \pm 127}$	**+27.5%**

Table 5.3: Isolating the credit horizon on the three-room restaurant. *GNN value* is the augmented one-task GNN of Section 5.4.3; *native selection* uses the Talukder et al. (2024) selection rule (immediate cost plus estimated anticipatory cost); *BN search* is our cost-bounded Bellman–Novelty search. Improvement is reported against the three-room myopic oracle (1,349). *Anticipatory value, BN search* is the Anticipatory RL (guided) agent used in previous sections. Values are mean ± standard deviation across seeds (counts in Appendix A.1).

5.5 Summary

This chapter measured anticipation in a persistent restaurant environment. The minimal example isolated the effect: an anticipatory value assigns a completed state the preparedness value of the tasks that may follow, a signal a myopic value cannot represent. In the restaurant, both the greedy and guided anticipatory agents reach high success, but the cost advantage of anticipation appears only under value-guided deployment — a greedy policy over the same value does not realize it. On the two-room restaurant, where the useful preparation pays off within a single next task, the anticipatory agent matches the augmented one-task GNN. On the three-room restaurant, where the preparation must be amortized across several tasks, the anticipatory agent's multi-task value clearly surpasses the one-task baseline and recovers most of the clairvoyant ceiling. Throughout, the value is learned online from interaction, without the planner-generated cost dataset the offline baseline requires.

Chapter 6: Conclusion and Analysis

This thesis recast anticipatory planning as a continuing task-conditioned MDP, in which one agent solves a stream of tasks in a world that is never reset between them. Under this formulation, preparedness is the value of the state a completed task leaves behind, averaged over the tasks that may follow, and it is contained in a task-conditioned action-value function. The agent learns the preparedness value online by reinforcement learning; the myopic and anticipatory agents run the same loop and the same environment, and differ only in whether the Bellman target at a completed task retains the discounted value of the resulting state, averaged over the next task. We then deployed the learned value in two ways: directly, as a greedy policy, and as the terminal-state heuristic of a cost-bounded symbolic search over ways of finishing the current task.

6.1 Findings

The cost reduction reported in Chapter 5 needs two things together: a value trained across task boundaries, and a deployment that ranks completed states rather than individual actions. Neither alone suffices. A greedy policy over the anticipatory value costs 1,432 per task, no cheaper than one over the myopic value (1,427); the search over the myopic value costs 1,196. Only the search over the anticipatory value reaches 914. The value estimates what a completed state is worth for the tasks that follow, and the search compares completed states; a greedy policy does not compare completed states.

We evaluate the agents on two restaurants: the two-room restaurant, derived from the environment of prior anticipatory-planning work, and a three-room variant that adds a distant pantry. On the two-room restaurant the useful preparation pays off within the next task, so the augmented one-task GNN — a supervised, offline

baseline — matches our agent (945 versus 914). On the three-room restaurant the jar sits in a distant pantry, and its cost is recovered over several later tasks; there the one-task objective cannot justify the trip to the distant pantry (3.1% reduction), while the multi-task value does (27.5%). The multi-task credit horizon is what recovers this saving; a one-task value cannot, even deployed through the same search (Section 5.4.4). Against the clairvoyant oracle, which plans over the next K tasks it is shown in advance, our agent recovers 76% of the $K = 3$ saving on the two-room restaurant and about 97% on the three-room, with no knowledge of the future sequence and no planner-generated cost dataset. The oracle is handed the actual sequence, so it marks what a future-blind method could aspire to rather than a bound it must reach.

6.2 Addressing the Four Limitations

Chapter 1 listed four limitations of the offline, planner-based approach to anticipation. This work addresses two of them: L2 and L3.

(L2): Training labels are no longer produced by a planner, so the $N\,|\mathcal{T}|$ planner calls disappear. Section 5.4.3 quantifies what that dependence costs the offline method: the one-task GNN reaches the reported level only once its training states are augmented with prepared ones, and each added state is another $|\mathcal{T}|$ planner calls, so the repair that closes its accuracy gap widens the scaling gap L2 names. Offline methods are bounded by the coverage of the dataset one collects (Levine et al., 2020); online exploration instead supplies that coverage through interaction, without being told where to look. The $|\mathcal{T}|$ factor does survive on our side: the boundary target and the terminal score are exact sums over the task distribution, evaluated at every replayed boundary and every candidate terminal. But each term is a network forward pass, not a planner call. The planners our method still invokes are a one-time demonstration seed at initialization (Appendix A.1) and, at deployment, one per task for the fallback completion and the cost reference of Eq. 4.1. What L2 names as prohibitive

is per-state planner labeling during training, and that is removed.

(L3): Because value propagates across task boundaries, the credit horizon is set by the discount, not fixed at one task ahead. The three-room restaurant shows this matters: when the useful preparation must amortize across more than one task, the one-task objective cannot recover the saving even when given the prepared states (3.1%), while the multi-task value does (27.5%; Section 5.4.3). What this domain does not measure is the cost of a still-longer horizon for our own agent; the discount sweep that would trace it is confounded with within-task discounting, and we return to it among the limitations below.

(L1) and (L4) remain open. We still assume the task distribution is stationary and known, using it in the boundary target and in the terminal score at deployment (L1); and tasks are drawn i.i.d., so the anticipated task is independent of the current one, as in prior work (L4). Both are left as future directions.

6.3 Limitations and Future Work

Scale. The evidence comes from a single restaurant family — two layouts that differ only in the jar's placement, with a fixed object set and task space — so generality is untested. A larger benchmark, with more objects, more task types, and varied layouts, would show whether the preparedness value learned here holds up as the domain grows. The completion reward was calibrated from a myopic-oracle sample of this cost model, so a distribution with rarer, more expensive plans would need recalibrating first.

Realism. A natural extension is to real human activity. The task distribution here is specified by hand; driving it instead from recorded human routines, where the tasks and their timing come from data, would test the method in the setting that motivates it.

Tighter bounds and the horizon-cost tradeoff. The clairvoyant ceiling

and the guided costs both come from a satisficing, budget-limited search, so the ceiling is an upper bound — the 76% we report is optimistic — and the guided costs are the best a budget-limited search found, not the lowest a preparedness-scored search could reach. Stronger, ideally exact, bounds would sharpen these numbers, and with them one could measure directly how the credit horizon trades off against success rate, which the present γ sweep cannot separate from within-task discounting.

Relaxing the distribution assumption. The method uses the exact distribution $P(\tau)$ at both the boundary target and terminal scoring, so it must be known (L1). Replacing the exact expectation with a K-task Monte Carlo estimate — sampling tasks rather than weighting by known probabilities — would soften this to needing only samples from the stream. Because the distribution enters only through this expectation, $P(\tau)$ can instead be formed from the agent's own experience: an empirical average over the observed task stream removes the need to know P at all, and a running average that weights recent tasks more heavily would track a drifting P, pointing anticipatory reinforcement learning toward the lifelong, non-stationary regime of continual reinforcement learning (Khetarpal et al., 2022).

Sequential task dependence. Tasks are drawn i.i.d., so the anticipated task does not depend on the current one (L4). Replacing $P(\tau')$ with a conditional $P(\tau' \mid \tau)$ in the boundary target and terminal score — the only two places the distribution enters — would let the agent exploit correlations such as cooking followed by cleaning, with little change to the method.

The deployment gap. Neither deployment guarantees both completion and cheap completion: the guided agents' 100% success comes from the planner, not the value, and the greedy agents — the deployment that needs no planner — leave 2.4% and 4.0% of tasks unfinished on the two-room restaurant, and 36.2% and 6.7% on the three-room restaurant. The value selects well among completions; it does not reliably produce one.

Closing

A value learned online, with no planner-generated cost dataset, carries enough of a preparedness signal that a cost-bounded search over task completions recovers much of what an oracle shown the actual future achieves. Framed as an online learning problem, the value improves with use and extends naturally — to sequentially correlated tasks, to unknown distributions, and to non-stationary ones that drift. Each is a step toward robots that plan for the long horizon of tasks ahead.

Appendix A: Supplementary Material

A.1 Reproducibility

All experiments implement Algorithm 1 (Chapter 4) and are trained on a single GPU, with runs distributed across two machines: an Intel Xeon Gold 6342 (96 threads, 503 GB) with NVIDIA A40 GPUs, and an Intel Core Ultra 9 275HX (24 threads, 30 GB) with a single RTX 5080; both run Ubuntu 24.04. Reported costs are PDDL plan costs produced by a deterministic planner, so they are machine independent.

Code. Code to reproduce the experiments is available at `https://raraghavarora.github.io/anticipatory-rl/`.

Environments. The two restaurants — their rooms, locations, and movement costs — are described in Section 5.2. Both use the same nine objects (two cups, two water sources, apple, bowl, knife, jar, plate) and the task distribution of Table 5.1.

Observation and action representation. The observation is a flat vector of one-hot and binary fields, of dimension 318 in the two-room restaurant and 329 in the three-room restaurant (the difference is the extra location). It concatenates, in order: the agent location (L dims); a hand-free bit (1); the held object (O+1, including a *none* slot); then, for each of the O objects, its location (L+1), a dirty bit (1), its liquid contents (4: empty, water, coffee, or none), what it is contained in (O+1), and its kind (7); a bread-spread field (O); and finally the task encoding. Here L is the number of locations (6 or 7) and $O = 9$ is the number of objects (two cups, two water sources, apple, bowl, knife, jar, plate). The task is encoded as four one-hot fields appended to the observation: task type (6), target location (L+1), target kind

(7+1), and target object (O+1), each with a *none* slot where the field does not apply to that task type.

The action is factored into four components, each a separate network head: an *action type* and three arguments — `object1`, `location`, and `object2`. There are ten active domain action types (move, pick, place, wash, fill, make-coffee, make-fruit-bowl, pour, refill-water, drain) plus an internal `auto_complete` event used only to credit an already-satisfied task, never emitted as a policy action. Each action type uses only the arguments it needs (Table A.1): *move* and *place* take a location; *pick*, *wash*, *fill*, *make-coffee*, *pour*, and *drain* take one object; *make-fruit-bowl* and *refill-water* take two. `object1` and `object2` range over the O objects plus a *none* index, and `location` over the L locations plus a *none* index. Any argument a given action type does not use is fixed to its *none* index, both in the environment and in the masks that constrain valid choices during action selection.

Table A.1: Arguments used by each action type. Unlisted arguments are fixed to their *none* index.

Action type	Arguments
move, place	`location`
pick, wash, fill, make-coffee, pour, drain	`object1`
make-fruit-bowl, refill-water	`object1`, `object2`

Hyperparameters. Table A.2 lists the training hyperparameters. Myopic and anticipatory runs share every value except the episode length: the myopic agent treats each task as its own episode (one task per episode), while the anticipatory agent trains over 50-task episodes. The two restaurants differ only in the completion reward, which is calibrated per cost model (Section 5.2.4), and in the protected-demo fraction. Each RL value function is trained with four seeds $(0, 4, 8, 16)$ on the two-room restaurant and five $(0, 4, 8, 16, 42)$ on the three-room; the GNN configurations

use four seeds $(0, 4, 8, 16)$, and the planning references are deterministic and carry no seed spread.

Table A.2: Training hyperparameters. All values are shared between the myopic and anticipatory agents except the tasks per episode.

Parameter	Two-room	Three-room
Total training steps	500,000	
Replay buffer size	50,000	
Minibatch size	128	
Hidden dimension	256	
Learning rate	3×10^{-4}	
Discount γ	0.97	
ϵ schedule	$1.0 \rightarrow 0.05$, exponential decay over 100,000 steps	
Target network update	every 1,000 steps	
Max gradient norm	1.0	
Tasks per episode	1 (myopic) / 50 (anticipatory)	
World reset interval	50 tasks	
Max steps per task	64	
Boundary bootstrap	weighted (exact $\mathbb{E}_P$)	
Completion reward R_{goal}	81.07	74.01
Protected-demo fraction	0.2	0.25

Demo transitions. The replay buffer is seeded at initialization with pre-generated oracle demonstrations. Each demonstration is one task outcome solved by the symbolic planner in a persistent rollout (auto-satisfied outcomes are settled without a planner call and store no action-conditioned transition), so the number of planner calls is at most the number of demonstrated outcomes. In the two-room restaurant the seed set is 500 task outcomes, giving 3,746 transitions for the myopic agent and 3,742 for the anticipatory agent. In the three-room restaurant it is 480 outcomes (3,276 transitions) for the myopic agent and 48 outcomes (320 transitions) for the anticipatory agent. We increased the number of oracle transitions for the myopic because the training was unstable and results were poor without it. All demonstrations are generated once at seed 0 and shared across training seeds. The protected-demo

fraction reserves that share of every minibatch for the demo buffer, preventing demo washout during early training.

Cost bound β. The value-guided search bound β (Section 4.2.2) is shared by the myopic and anticipatory guided agents and is fixed at 1.25 on the two-room restaurant. On the three-room restaurant it was selected from $\{1.25, 3, 6, 8\}$ on three development sequences, giving $\beta = 3.0$; the larger bound is needed because the pantry investment is too expensive to fall within a tighter current-task budget. Because these three sequences are among the ten reported, we recomputed the advantage on the remaining seven, held out from the selection: the anticipatory guided agent is 22.8% cheaper than the augmented one-task GNN and 17.4% cheaper than the $K = 2$ optimal oracle, versus 25.2% and 20.4% across all ten. The result is therefore not an artifact of the β choice; β affects only deployment, not value learning.

GNN baseline. The One-task GNN (item 3) reproduces the graph-network cost estimator of Talukder et al. (2024), matching their hyperparameters where the paper reports them. A state is encoded as a graph whose nodes are the agent, the rooms, the locations, and the objects. Each node carries a 399-dimensional feature: a 384-dimensional SBERT embedding (`all-MiniLM-L6-v2`) of the node's name, a 4-way node-type one-hot (agent / room / location / object), the node's (x, y) position, and nine binary attributes (dirty, filled-with-water, filled-with-coffee, empty, held, liquid-source, container, hand-free, bread-spread). Edges connect the agent to its current location, each object to its current location and to any container holding it, and each location to its room (bidirectional). The estimator is four TransformerConv layers (Shi et al., 2021) of width 64, each followed by batch normalization and a LeakyReLU; a graph embedding is formed by adding global mean- and sum-pooling, and a single linear head maps it to the scalar anticipatory planning cost C_{AP} (Eq. 3.11), giving 152,897 parameters. It is trained by L_1 regression against C_{AP} with Adagrad (learning rate 0.01, batch size 8, an 80/20 train/validation split, and

a step scheduler halving the rate every 1,000 steps), over four seeds $(0, 4, 8, 16)$; the faithful variant trains for 40 epochs and the augmented variant (item 4) for 12. The augmented variant differs only in its training set, which adds counterfactual prepared states, as described in Section 5.2.

A.2 Learning Dynamics

Figure A.1 tracks four quantities over 500,000 training steps, comparing the anticipatory and myopic agents on both restaurants: what the agent achieves (success rate), how much value it captures (task return), and whether its value estimates are stable (selected Q and TD target). Task return, selected Q, and the TD target are in RL cost units — the reward scale of Eq. 3.1 — not the PDDL cost units of the results tables. The top row is the two-room restaurant with 4 seeds per agent; the bottom row is the three-room restaurant with 5 seeds per agent. The two rows tell different stories, and the difference is the point: on the two-room restaurant the agents are hard to tell apart during training, while on the three-room restaurant they separate.

Success rate. These are rolling training-time success rates over a window of 100 tasks, measured while the agent is still exploring. On the two-room restaurant both agents finish close together, at 0.95 (anticipatory) and 0.97 (myopic). The anticipatory agent rises faster for the first third of training—0.44 against 0.35 at 50,000 steps, 0.80 against 0.75 at 100,000—after which the myopic agent overtakes it and finishes two points ahead. On the three-room restaurant the gap is large and in the other direction: 0.93 against about 0.5. The myopic agent never settles, oscillating between roughly 0.35 and 0.8 for the whole run, while the anticipatory agent converges smoothly. The boundary bootstrap therefore does not slow learning down on either restaurant; on the harder one it is what makes training converge at all.

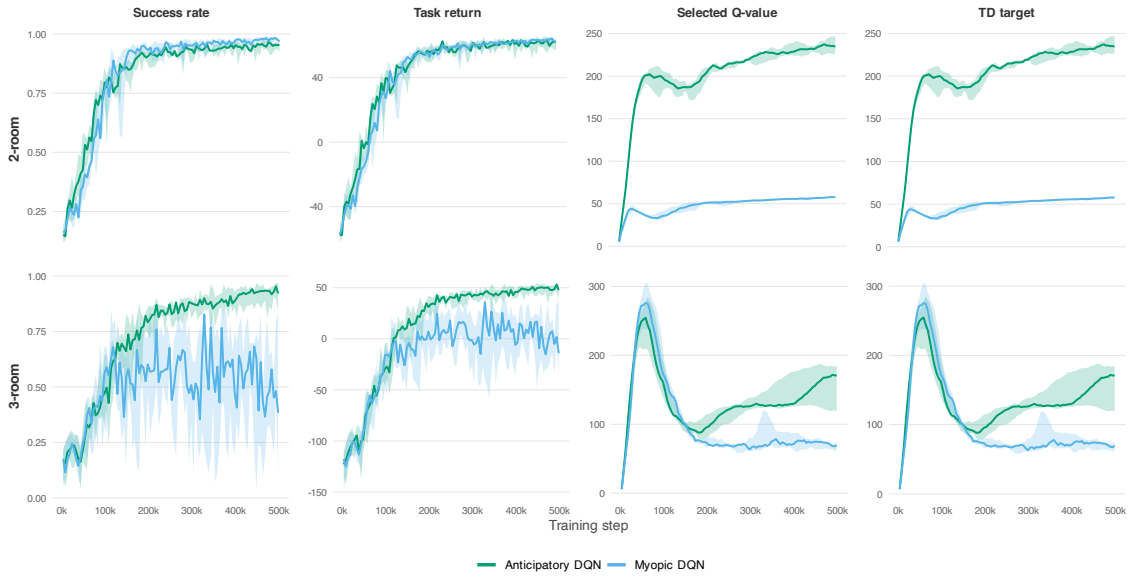


Figure A.1: Training diagnostics for the anticipatory and myopic agents on the two-room (top) and three-room (bottom) restaurants. Lines are the median across seeds and bands span the minimum and maximum; 4 seeds per agent on the two-room restaurant and 5 on the three-room restaurant.

Task return. Task return measures the cumulative discounted reward per task (goal bonus minus action cost). On the two-room restaurant both agents converge to nearly the same value, 60.9 (anticipatory) and 62.7 (myopic), with the anticipatory agent ahead for the first 150,000 steps and about two units behind at the end. That small final gap is the cost of anticipation measured on the current task alone: this metric scores one task at a time and cannot credit a preparedness premium back. Whether it is recovered over the sequence is the central empirical question addressed in Chapter 5. On the three-room restaurant the anticipatory agent ends at 49.6 while the myopic agent hovers around 0 and ends slightly negative, so it is ahead even by the single-task measure that ought to penalize it.

Selected Q and TD target. On the two-room restaurant both Q-values and TD targets rise steadily and converge. The anticipatory agent's values settle around 233, roughly four times the myopic agent's 57. This separation is what the two credit horizons predict. The myopic target truncates at task completion, so its Q approximates the return of a single task: the completion reward $R_{\text{goal}} = 81.07$ less the discounted cost of reaching it. The anticipatory target retains $\gamma V_{\text{AP}}(s')$, so its Q accumulates the discounted return of the tasks that follow as well. The size of the gap is consistent with that reading. At $\gamma = 0.97$ and the 9.4 steps per task this agent averages (Table A.4), the effective per-task discount is $\gamma^{9.4} \approx 0.75$, so an anticipatory value accumulates about $1/(1-0.75) \approx 4$ tasks' worth of return—close to the observed ratio of 4.1. The magnitude of Q therefore separates the two agents by construction and is not itself evidence that either value is better calibrated. The close tracking between selected Q and the TD target in both agents, with no divergence, indicates that Double-DQN with the weighted exact boundary expectation trains stably across all 4 seeds at $\gamma = 0.97$. This stability does not extend to longer horizons; Section A.5 reports what happens above $\gamma = 0.97$.

On the three-room restaurant the shape is different. Both agents overshoot early, peaking at 255 (anticipatory) and 275 (myopic) around 58,000 steps, fall back

to roughly 90 and 80 respectively by 180,000, and then diverge: the anticipatory value recovers to 170 while the myopic value settles at 70. The myopic figure is the diagnostic one. A myopic target truncates at task completion, so its Q cannot exceed the completion reward, which is 74.01 on this restaurant. The early peak of 275 is nearly four times a bound the myopic objective cannot cross, and indicates systematic overestimation rather than a longer effective horizon; the settled value of 70 sits just below the bound, a value that has partially recovered but never recalibrated. This is consistent with the myopic agent's unstable success rate on the same restaurant, and it means the three-room myopic rows should be read as a weaker baseline than the two-room ones rather than as a like-for-like comparison.

Jar use over training. Figure A.2 tracks the share of *all* actions that are `refill_water`, the action that draws from the reusable jar rather than walking to the fountain. This is a smaller number than the jar's share of water *deliveries* in Table A.5, since most actions are moves and pickups; the two measure the same behaviour against different denominators. The share scales with the credit horizon. At $\gamma = 0.98$ it climbs to 1.65% and is still rising at the end of training; at $\gamma = 0.97$ and 0.95 it peaks earlier, at 1.34% and 1.11%, and then decays to roughly half its peak; at $\gamma = 0.90$ it never exceeds 0.17%. The myopic agent starts at 0.41%, which is the behaviour present in the oracle demonstrations seeded into its replay buffer, and decays to almost zero: it is not that the myopic agent never sees the jar, but that it unlearns it. The jar is therefore a learned preparation whose adoption depends on how far credit reaches, not a behaviour built into the environment or the demonstrations.

Convergence and checkpoint selection. Figure A.3 shows rolling success for every run, with a dashed rule at the step from which `restaurant_dqn_best.pt` was taken. The rule marks only the step. The checkpoint is selected on persistent greedy evaluation over 100 tasks, which is a different measurement from the rolling training success plotted here, and on the noisiest runs the two differ substantially—the myopic

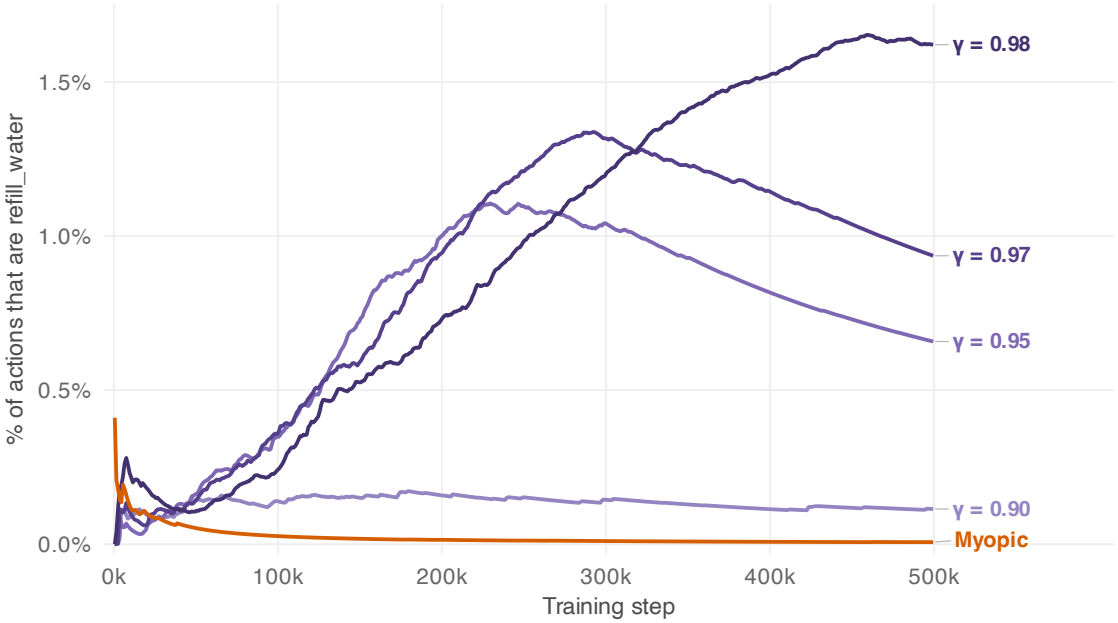


Figure A.2: Share of actions that are `refill_water` over training, for anticipatory agents at four discount factors and the myopic agent (seed 0; three-room restaurant). Jar use rises with the credit horizon and decays to zero for the myopic agent.

checkpoint scores 0.84 under evaluation at a step where rolling training success is 0.34. Two things are visible. First, runs at $\gamma \leq 0.97$ reach a plateau by 300–400,000 steps and hold it, so the 500,000-step budget is not truncating training. Second, every checkpoint at those discounts is taken between 430,000 and 500,000 steps, inside the plateau rather than at an early peak. At $\gamma = 0.98$ and 0.99 the picture differs: individual seeds collapse partway through training and do not recover, which is the behavioural counterpart of the value divergence in Section A.5. The checkpoints for those runs are correspondingly earlier, at 240,000 and 290,000 steps, taken before the collapse.

A.3 Bellman–Novelty Search: Full Algorithm

The main text (Algorithm 2) omits the bookkeeping that makes the search deterministic and terminating. The full procedure is given below: Algorithm 3 is the

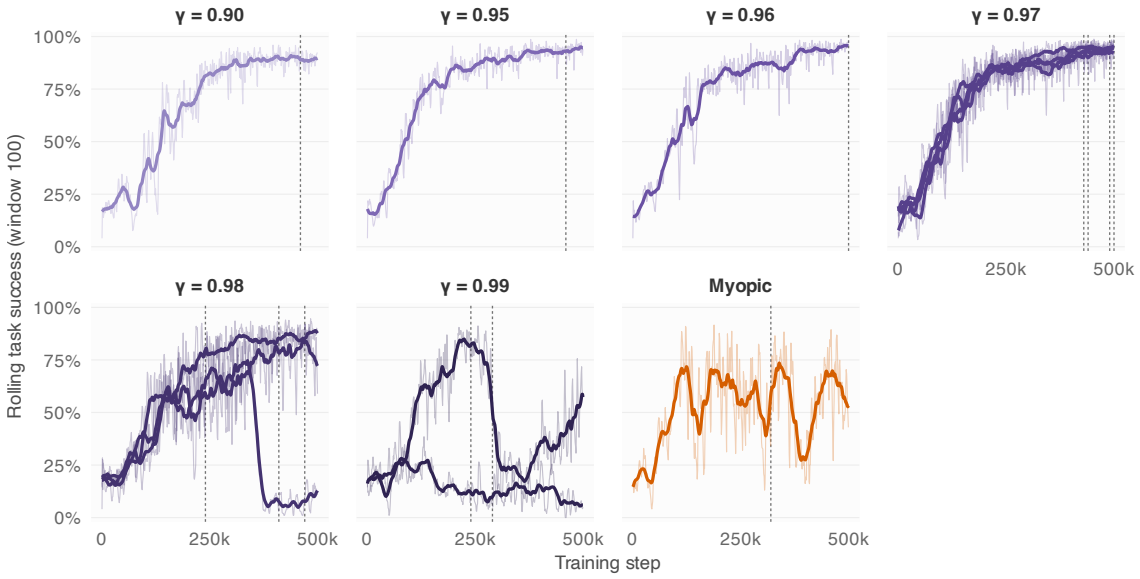


Figure A.3: Rolling task success (window 100) over training, one line per run, faceted by discount factor and agent. Bold lines are a 25,000-step centred moving average; the raw trace is drawn faintly behind each. Dashed rules mark the step at which each saved best checkpoint was taken; the checkpoint is selected on greedy evaluation, so the rule marks the step and not a value on this curve. For $\gamma \in 0.97, 0.98, 0.99$ multiple seeded runs are shown.

search loop and Algorithm 4 the node admission, terminal scoring, and cost-bounded selection, with notation in Table A.3. Here $\mathrm{sig}(s)$ is a canonical key that identifies a symbolic state (two states with the same key are equal for deduplication), $\mathrm{atoms}(s)$ is the set of ground facts true in s, and $\mathrm{pairs}(s)$ the set of unordered pairs of such facts; $\textsc{Width}(s; \mathcal{A}, \mathcal{P})$ returns 1 if s makes true some fact not in $\mathcal{A}$, else 2 if it makes true some pair not in $\mathcal{P}$, else $\perp$ (not novel). $\textsc{Step}(n, a)$ applies action a to node n, forming the successor whose fields update as noted in Algorithm 3.

Symbol	Meaning
$s_0,\ \tau$	initial state; *current* task being solved
$\mathcal{D}$	distribution over the next task τ' (state-independent)
$n = (s, \pi, d, G, c)$	node: state, action prefix, depth, discounted prefix return $G \leq 0$, undiscounted cost $c \geq 0$
γ	RL discount, used for G and for the search priority
$B,\ d_{\max}$	expansion budget; depth cap on *open* nodes (20)
β	cost-ratio slack (3.0 three-room, 1.25 two-room)
$q_{\max}(s)$	$\max_{a \in \mathcal{A}(s)} Q(s, \tau, a)$ over mask-valid actions, *current* task
$V_{\mathrm{AP}}(s)$	$\mathbb{E}_{\tau' \sim \mathcal{D}}[\max_a Q(s, \tau', a)]$, *next* task
$w(s) \in \{1, 2, \perp\}$	novelty width w.r.t. atom table $\mathcal{A}$ and pair table $\mathcal{P}$
H_B	Bellman heap, max-key $G(n) + \gamma^{d(n)} q_{\max}(s_n)$
H_N	novelty heap, min-key $(w, d, \text{counter})$, FIFO tie-break
$k(n)$	dedup key $(\mathrm{sig}(s_n), d(n))$
seen, expanded	maps $k(n) \mapsto$ best G: pre-push dedup / expansion ledger
$\mathcal{T}$	terminal set (candidate completions of τ)

Table A.3: Notation for Bellman–Novelty Search.

Deduplication is heuristic. For each key $(\mathrm{sig}(s), d)$, the search keeps only the path reaching it with the highest discounted return G. This prioritizes the higher-value path to a state, but the cost bound in SelectTerminal is applied to the *undiscounted* cost c, and at equal depth a higher G does not imply a lower c (discounting weights earlier costs more). Deduplication can therefore, in rare cases, retain a completion of a given state that is costlier by undiscounted cost than a discarded one, and so change whether that state is eligible under $\beta\, C_{\mathrm{myo}}$. Because all paths to

Algorithm 3 Bellman–Novelty Search, headline configuration (`cost_bounded`).

Require: $s_0, \tau, \mathcal{D}$; trained Q; budget B; depth cap $d_{\max}$; slack β

$\mathcal{T} \leftarrow \emptyset$; seen,expanded$\leftarrow$ empty; $\mathcal{A}, \mathcal{P} \leftarrow \emptyset$; $H_B, H_N \leftarrow \emptyset$; $e \leftarrow 0$; $\mathit{phase} \leftarrow 0$
$n_{\text{FD}} \leftarrow \textsc{FastDownward}(s_0, \tau)$ ▷ myopic reference plan; may fail or time out
if $n_{\text{FD}} \neq \textsc{fail}$ **then** $\textsc{TryAddTerminal}(n_{\text{FD}})$ ▷ seed $\mathcal{T}$; otherwise it starts *empty*
$\textsc{Enqueue}((s_0, \langle\rangle, 0, 0, 0))$
while $e < B$ **and** $(H_B \neq \emptyset$ **or** $H_N \neq \emptyset)$ **do**
  if $\mathit{phase} < 3$ **then** ▷ Bellman turn: three of every four iterations
    $\mathit{phase} \leftarrow (\mathit{phase} + 1) \bmod 4$
    if $H_B = \emptyset$ **then continue** ▷ turn forfeited, no budget charged
    $n \leftarrow \textsc{PopMax}(H_B)$
    if $\textsc{Stale}(n)$ **then continue**
    $\textsc{Expand}(n)$
  else ▷ novelty turn: one of four, retries until it finds work
    $\mathit{phase} \leftarrow 0$; $n \leftarrow \textsc{NoveltyPop}$
    if $n = \perp$ **then continue**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \text{atoms}(s_n)$; $\mathcal{P} \leftarrow \mathcal{P} \cup \text{pairs}(s_n)$ ▷ tables grow *only* here
    $\textsc{Expand}(n)$
  end if
end while
return $\textsc{SelectTerminal}(\mathcal{T})$
procedure $\textsc{Stale}(n)$
  return expanded$[k(n)] \geq G(n)$ **or** $d(n) \geq d_{\max}$
end procedure
procedure $\textsc{Expand}(n)$
  expanded$[k(n)] \leftarrow G(n)$; $e \leftarrow e + 1$ ▷ budget counts *successful* expansions
  for all mask-valid factored actions a at s_n **do**
    $n' \leftarrow \textsc{Step}(n, a)$ ▷ $s'{=}f(s_n, a)$; $d'{=}d{+}1$, $G'{=}G - \gamma^d c(s_n, a)$, $c'{=}c + c(s_n, a)$
    if n' satisfies τ **then** $\textsc{TryAddTerminal}(n')$ **else** $\textsc{Enqueue}(n')$
  end for
end procedure
procedure $\textsc{NoveltyPop}$
  while $H_N \neq \emptyset$ **do**
    $(w_{\text{old}}, \cdot, \cdot, n) \leftarrow \textsc{PopMin}(H_N)$
    if $\textsc{Stale}(n)$ **then continue**
    $w \leftarrow \textsc{Width}(s_n; \mathcal{A}, \mathcal{P})$
    if $w = \perp$ **then continue** ▷ no longer novel: discard
    if $w > w_{\text{old}}$ **then** $\textsc{Push}(H_N, (w, d(n), \textsc{next}, n))$; **continue**
    return n
  end while
  return $\perp$
end procedure

Algorithm 4 Node admission, terminal scoring, and cost-bounded selection.

procedure ENQUEUE(n)
  if $d(n) \geq d_{\max}$ **then return** ▷ cap gates *open* nodes only
  if $k(n) \in \mathsf{seen}$ **and** $G(n) \leq \mathsf{seen}[k(n)]$ **then return**
  $\mathsf{seen}[k(n)] \leftarrow G(n)$
  PUSH($H_B,\ G(n) + \gamma^{d(n)} q_{\max}(s_n)$) ▷ value-guided lane, *current* task
  $w \leftarrow \textsc{Width}(s_n; \mathcal{A}, \mathcal{P})$; **if** $w \neq \bot$ **then** PUSH($H_N, (w, d(n), \textsc{next}, n)$)
end procedure

procedure TRYADDTERMINAL(n) ▷ n completes τ; no depth cap here
  if $k(n) \in \mathsf{seen}$ **and** $\mathsf{seen}[k(n)] \geq G(n)$ **then return**
  $\mathsf{seen}[k(n)] \leftarrow G(n)$
  $s' \leftarrow \textsc{ConsumeDelivery}(s_n, \tau)$ ▷ delivered items leave the world
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(\pi_n,\ c_n,\ d_n,\ V_{\mathrm{AP}}(s'))\}$
end procedure

procedure SELECTTERMINAL($\mathcal{T}$)
  if $\mathcal{T} = \emptyset$ **then return** NO-SELECTION
  $C_{\mathrm{myo}} \leftarrow c$ of the Fast Downward terminal if present, **else** $\min_{t \in \mathcal{T}} c_t$
  $c_{\mathrm{bud}} \leftarrow \beta \cdot C_{\mathrm{myo}}$ ▷ spend at most $\beta\times$ the myopic cost
  $\mathcal{E} \leftarrow \{t \in \mathcal{T} : c_t \leq c_{\mathrm{bud}}\}$
  if $\mathcal{E} = \emptyset$ **then return** NO-SELECTION
  return $\arg\max_{t \in \mathcal{E}} V_{\mathrm{AP}}(t)$, ties broken by smaller c_t, then smaller d_t, then discovery order
end procedure

the same state share the same V_{AP}, the effect is confined to which representative of an identical state is scored; we treat this as a heuristic approximation rather than a guarantee.

A.4 Detailed Evaluation Results

Table A.4 gives the numeric values behind Figure 5.3.

Restaurant	Deployment	Horizon	Success (%)	Steps/task	Cost/task
2-room	Greedy	Myopic	97.6 ± 1.0	8.57 ± 0.54	$1{,}427 \pm 26$
		Anticipatory	96.0 ± 0.9	9.43 ± 0.46	$1{,}432 \pm 49$
	Value-guided	Myopic	100.0 ± 0.0	7.35 ± 0.17	$1{,}196 \pm 108$
		Anticipatory	100.0 ± 0.0	7.05 ± 0.06	$\mathbf{914 \pm 19}$
3-room	Greedy	Myopic	63.8 ± 8.8	27.31 ± 4.98	$2{,}689 \pm 316$
		Anticipatory	93.3 ± 3.6	11.24 ± 2.07	$1{,}638 \pm 136$
	Value-guided	Myopic	100.0 ± 0.0	9.57 ± 1.14	$1{,}656 \pm 131$
		Anticipatory	100.0 ± 0.0	7.77 ± 0.38	$\mathbf{978 \pm 127}$

Table A.4: Greedy and value-guided evaluation of the two learned value functions on the two-room and three-room restaurants, mean ± std across checkpoint seeds (each seed averaged over ten canonical 50-task sequences, 500 tasks per seed). Costs are in PDDL cost units. The value-guided agents reached 100% success on these evaluations, as the planner solved every task within its time budget.

Figures A.4 and A.5 give the full per-task cost distributions and the cumulative cost over a sequence.

Per-task-type cost breakdown. On the two-room restaurant, the cost reduction under value-guided anticipatory deployment is concentrated in the two task types that require water; the remaining task types are essentially unchanged (Figure A.6).

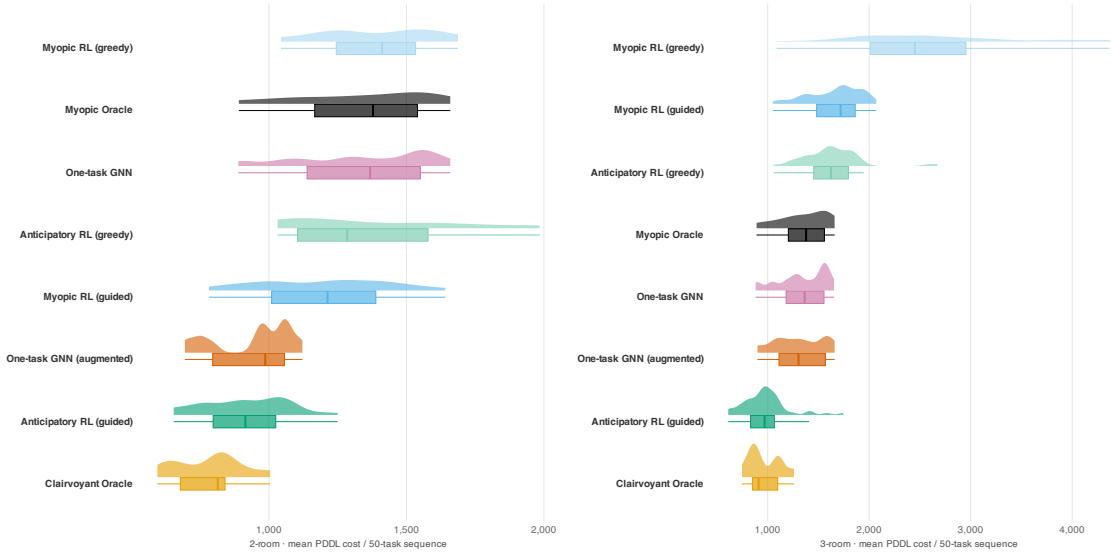


Figure A.4: Distribution of mean PDDL cost per task across 50-task sequences, sorted by median cost. Greedy agents are shown for their spread; their $< 100\%$ success rate means their cost includes abandoned-task penalties, so they are not directly comparable on cost. The clairvoyant oracle corresponds to $K = 3$.

A.5 Horizon vs. Calibration Tradeoff

Table A.5 summarizes how the discount factor γ affects jar sharing, task completion, and value calibration. As γ rises the reusable jar is used more often, from 4.6% of water deliveries at $\gamma = 0.90$ to 73.0% at $\gamma = 0.99$, but task completion degrades and the mean Q-value inflates. The rise is not strictly monotone: $\gamma = 0.96$ draws from the jar less often (14.5%) than $\gamma = 0.95$ (21.3%). Success is highest in the middle of the sweep, at $\gamma = 0.95$, and falls at both ends (Figure A.8). The ratio $R^*/\text{mean}\, Q$ compares the mean Q-value against the fixed completion reward R^*. Mean Q scales with the effective horizon $1/(1-\gamma)$, so this ratio falls as γ rises whether or not the value is overestimated, and it is not a horizon-invariant measure of accuracy. The Succ column is the training success rate averaged over the whole run, not the greedy evaluation protocol of Table A.4.

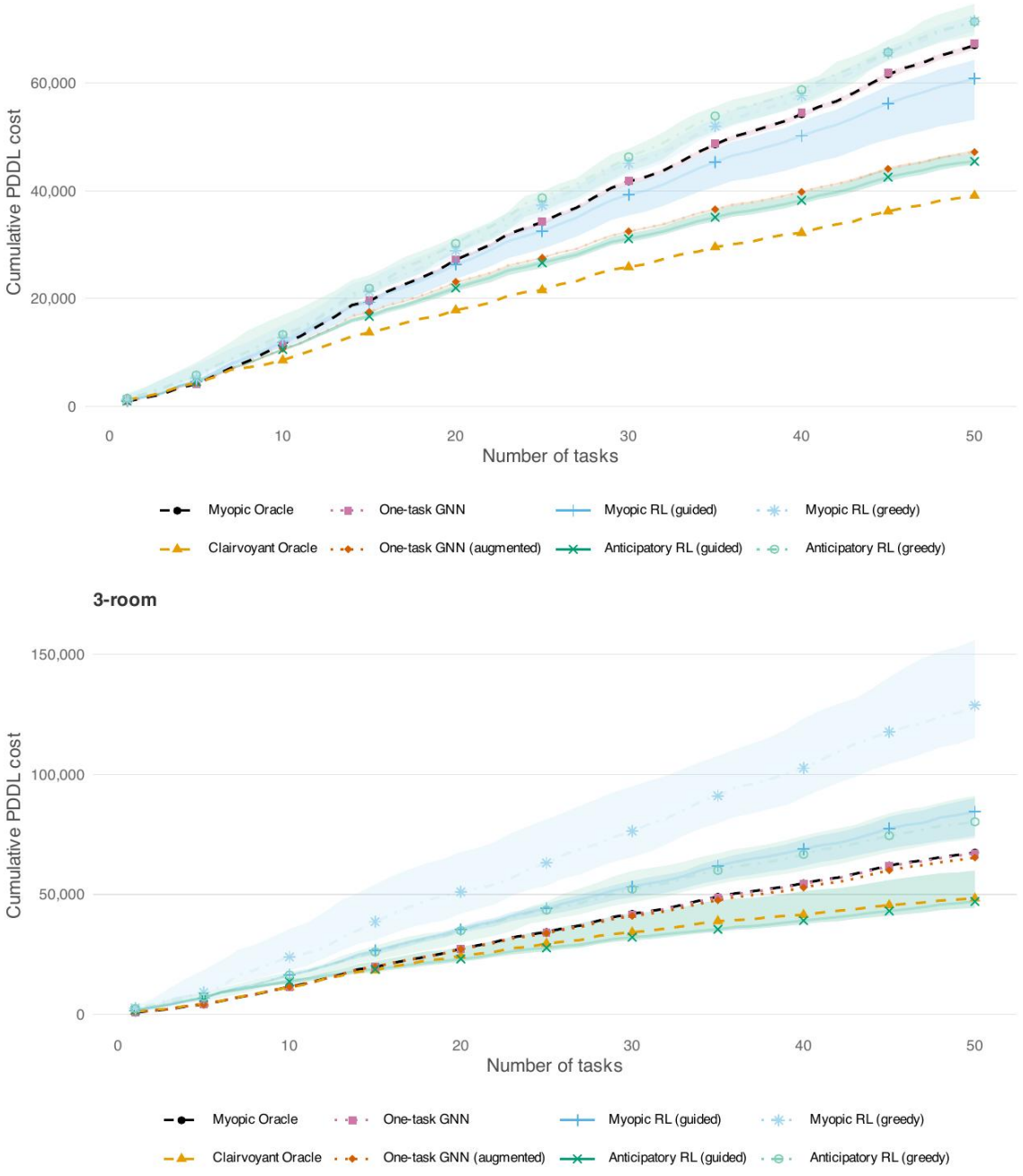


Figure A.5: Cumulative PDDL cost for all eight methods on the two-room (top) and three-room (bottom) restaurants. Shaded curves show checkpoint-seed medians and min–max ranges. The value-guided anticipatory agent separates early from the myopic methods, with a widening gap.

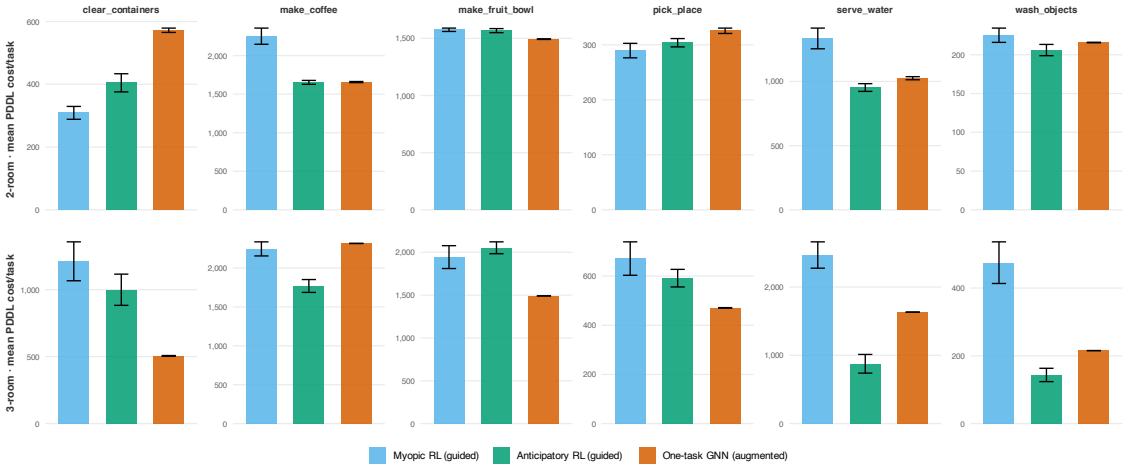


Figure A.6: Comparison of cost between our guided RL agents and the augmented one-task GNN, split by task types. Error bars show the standard error of the mean across seeds.

Divergence at long horizons. The table reports seed 0, which understates the instability at the top of the sweep. The sweep was run with fewer seeds than the main comparison, for compute reasons: three seeds at each of $\gamma = 0.98$ and 0.99. At $\gamma = 0.98$ the three seeds reach mean Q of 195, 194, and 244,649; at $\gamma = 0.99$ they reach 401, 443, and 1.8×10^8. Figure A.7 shows this directly: two runs leave the band the others occupy and grow by roughly ten and six orders of magnitude, while every run at $\gamma \leq 0.97$ stays flat. Past $\gamma \approx 0.97$, then, the value function does not merely lose calibration, it can diverge outright, and the calibration ratio understates the failure because it is finite even when the value is not usable. This is the cost of a longer credit horizon in this domain: the horizon that buys the most reusable preparation (Figure A.2) is also the one at which training stops being reliable.

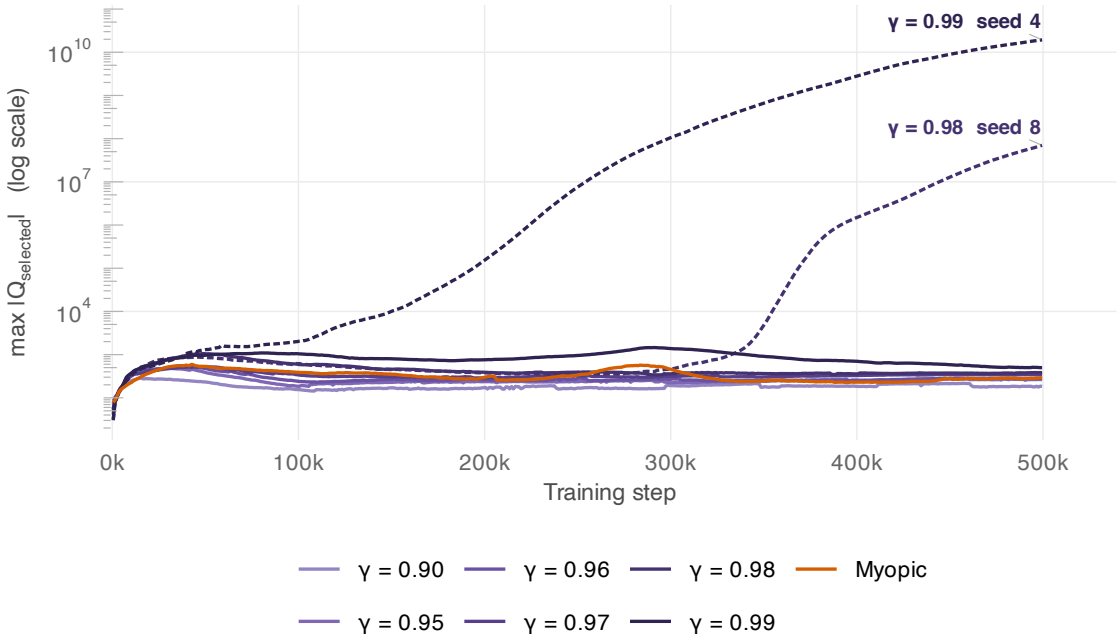


Figure A.7: Maximum absolute selected Q over training, log scale. Solid lines are the best seed at each discount factor, ranked by the greedy evaluation score of its saved checkpoint; they stay flat throughout. Dashed lines are the two runs that diverge, by roughly ten and six orders of magnitude. Both are the *worst* seed at their discount factor, so their sibling runs at the same γ remain stable.

Table A.5: Discount factor γ vs. jar sharing, success rate, mean Q-value, and calibration ratio $R^*/\text{mean}\,Q$, on the three-room restaurant for the anticipatory agent. All rows are seed 0, matching Figure A.8. Mean Q is the seed-0 batch average; it is a different aggregation from the seed-averaged final selected-Q values in Section A.2 and should not be read as the same quantity. Mean Q and $R^\star$ are in RL cost units.

γ	Jar share	Succ	Mean Q	$R^*/\text{mean}\,Q$
0.90	4.6%	0.783	36.9	200%
0.95	21.3%	0.842	88.7	83%
0.96	14.5%	0.826	105.5	70%
0.97	32.9%	0.815	132.2	56%
0.98	65.2%	0.750	195.5	38%
0.99	73.0%	0.661	401.0	18%

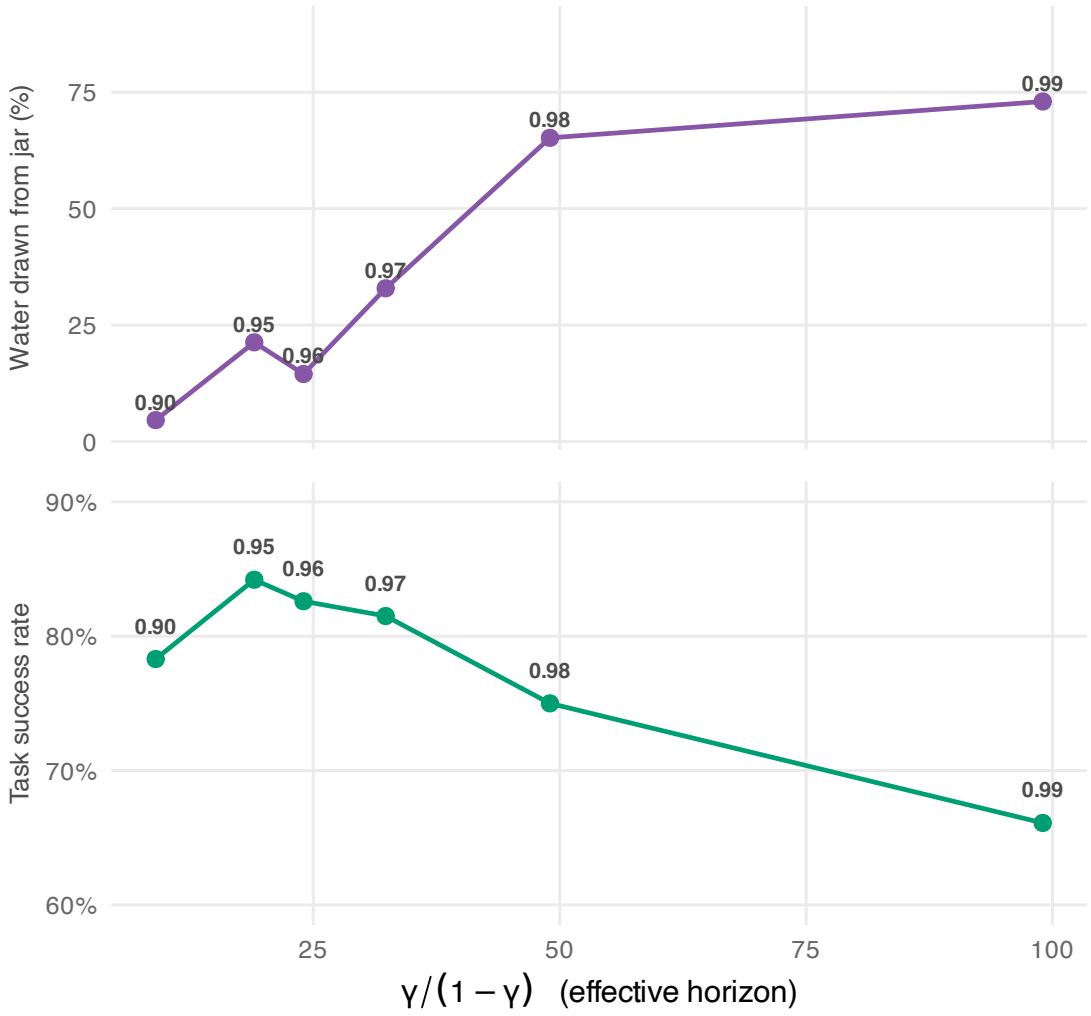


Figure A.8: Effect of the credit horizon on the anticipatory agent (three-room restaurant, seed 0), plotted against the effective horizon $\gamma/(1-\gamma)$. Top: share of water drawn from the reusable jar, which rises with the horizon. Bottom: training task success rate, which peaks near $\gamma = 0.95$ and falls at longer horizons. The reported experiments fix $\gamma = 0.97$.

References

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5366–5376, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Raghav Arora, Shivam Singh, Karthik Swaminathan, Ahana Datta, Snehasis Banerjee, Brojeshwar Bhowmick, Krishna Murthy Jatavallabhula, Mohan Sridharan, and Madhava Krishna. Anticipate & Act: Integrating LLMs and Classical Planning for Efficient Task Execution in Household Environments†. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14038–14045, May 2024. doi: 10.1109/ICRA57147.2024.10611164. URL `https://ieeexplore.ieee.org/document/10611164/`.

Andre Barreto, Will Dabney, Remi Munos, Jonathan J. Hunt, Tom Schaul, Hado van Hasselt, and David Silver. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Anand Sontakke, Grecia Salazar, Huong T. Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singh, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 3909–3928. PMLR, 06–09 Nov 2023. URL `https://proceedings.mlr.press/v229/chebotar23a.html`.

Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 447–454, 2016. doi: 10.1109/ICRA.2016.7487165.

Roshan Dhakal, Md Ridwan Hossain Talukder, and Gregory J. Stein. Anticipatory Planning: Improving Long-Lived Planning by Estimating Expected Cost of Future Tasks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11538–11545, May 2023. doi: 10.1109/ICRA48891.2023.10160260. URL https://ieeexplore.ieee.org/document/10160260/.

Roshan Dhakal, Duc M. Nguyen, Tom Silver, Xuesu Xiao, and Gregory J. Stein. Anticipatory Task and Motion Planning, July 2024. URL http://arxiv.org/abs/2407.13694. arXiv:2407.13694 [cs].

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems 35*, NeurIPS 2022, page 35603–35620. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-2580. URL http://dx.doi.org/10.52202/068431-2580.

Patrick Ferber, Malte Helmert, and Jörg Hoffmann. *Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space*, pages 2346–2353. IOS Press, 2020. doi: 10.3233/faia200364. URL http://dx.doi.org/10.3233/FAIA200364.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, December 1971. ISSN 0004-3702. doi: 10.1016/0004-3702(71)90010-5. URL http://dx.doi.org/10.1016/0004-3702(71)90010-5.

Timothy Flavin and Sandip Sen. Revisiting action factorization for complex action spaces, 2026. URL https://arxiv.org/abs/2606.26574.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, July 2006. ISSN 1076-9757. doi: 10.1613/jair.1705. URL http://dx.doi.org/10.1613/jair.1705.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards Continual Reinforcement Learning: A Review and Perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022. doi: 10.1613/jair.1.13673.

Victoria Krakovna, Laurent Orseau, Richard Ngo, Miljan Martic, and Shane Legg. Avoiding Side Effects by Considering Future Tasks. *Advances in Neural Information Processing Systems*, 33:19064–19074, 2020.

Matthew Landers, Taylor Killian, Hugo Barnes, Tom Hartvigsen, and Afsaneh Doryab. Brave: Offline reinforcement learning for discrete combinatorial action spaces. In *Advances in Neural Information Processing Systems 38*, NeurIPS 2025, pages 79505–79534. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2025. doi: 10.52202/085713-2400. URL http://dx.doi.org/10.52202/085713-2400.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL https://arxiv.org/abs/2005.01643.

Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, page 540–545, NLD, 2012. IOS Press. ISBN 9781614990970.

Bo Liu, Yihao Feng, Qiang Liu, and Peter Stone. Metric residual network for sample efficient goal-conditioned reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8799–8806, June 2023. ISSN 2159-5399. doi: 10.1609/aaai.v37i7.26058. URL http://dx.doi.org/10.1609/aaai.v37i7.26058.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5502–5511. International Joint Conferences on Artificial Intelligence Organization, July 2022. doi: 10.24963/ijcai.2022/770. URL https://doi.org/10.24963/ijcai.2022/770.

Andrea Micheli and Alessandro Valentini. Synthesis of search heuristics for temporal planning via reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11895–11902, May 2021. ISSN 2159-5399. doi: 10.1609/aaai.v35i13.17413. URL http://dx.doi.org/10.1609/aaai.v35i13.17413.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4045–4054. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/pardo18a.html.

Maithili Patel and Sonia Chernova. Proactive robot assistance via spatio-temporal object modeling. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=th7GW868Pok.

Maithili Patel, Aswin Gururaj Prakash, and Sonia Chernova. Predicting routine object usage for proactive robot assistance. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=rvh0vkwKUM.

Judea Pearl and Jin H. Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):392–399, 1982. doi: 10.1109/TPAMI.1982.4767270.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, April 1994. ISBN 9780470316887. doi: 10.1002/9780470316887. URL http://dx.doi.org/10.1002/9780470316887.

Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1312–1320, 2015.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/214. URL `https://doi.org/10.24963/ijcai.2021/214`. Main Track.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Satinder P Singh and Richard C Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994. ISSN 0885-6125. doi: 10.1023/A:1022693225949.

Shivam Singh, Karthik Swaminathan, Raghav Arora, Ramandeep Singh, Ahana Datta, Dipanjan Das, Snehasis Banerjee, Mohan Sridharan, and Madhava Krishna. Anticipate & Collab: Data-driven Task Anticipation and Knowledge-driven Planning for Human-robot Collaboration, April 2024. URL `http://arxiv.org/abs/2404.03587`. arXiv:2404.03587 [cs].

Markus Spies, Marco Todescato, Hannes Becker, Patrick Kesper, Nicolai Waniek, and Meng Guo. Bounded suboptimal search with learned heuristics for multi-agent systems. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN

978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33012387. URL https://doi.org/10.1609/aaai.v33i01.33012387.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA, second edition, 2018. URL https://dl.acm.org/doi/10.5555/3312046.

Md Ridwan Hossain Talukder, Raihan Islam Arnob, and Gregory J. Stein. Anticipatory Planning for Performant Long-Lived Robot in Large-Scale Home-Like Environments, November 2024. URL http://arxiv.org/abs/2411.12837. arXiv:2411.12837 [cs].

Md Ridwan Hossain Talukder, Roshan Dhakal, Elizabeth Phillips, and Gregory J. Stein. Courteous anticipation: Improving long-lived task planning in persistent shared environments, 2026. URL https://arxiv.org/abs/2607.20289.

Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Jordan T. Thayer and Wheeler Ruml. Bounded suboptimal search: a direct approach using inadmissible estimates. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 674–679. AAAI Press, 2011. ISBN 9781577355137.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S.

Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, October 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL http://dx.doi.org/10.1038/s41586-019-1724-z.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1995–2003. JMLR.org, 2016.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/bf00992698. URL http://dx.doi.org/10.1007/BF00992698.