Bidirectional End-to-End Framework for Transfer from Abstract Models in Non-Markovian Reinforcement Learning

Mahyar Alinejad[1] MAHYAR.ALINEJAD@UCF.EDU
Precious Nwaorgu[1] PRECIOUSMADUABUCHI.NWAORGU@UCF.EDU
Chinwendu Enyioha[1] CENYIOHA@UCF.EDU
Yue Wang[1] YUE.WANG@UCF.EDU
Alvaro Velasquez[2] ALVARO.VELASQUEZ@COLORADO.EDU
George Atia[1] GEORGE.ATIA@UCF.EDU
[1]*Department of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, USA*
[2]*Department of Computer Science, University of Colorado, Boulder, USA*

Editors: G. Pappas, P. Ravikumar, S. A. Seshia

Abstract

We propose a bidirectional end-to-end reinforcement learning (RL) framework for solving complex non-Markovian tasks in discrete and continuous environments. Instead of directly learning policies in high-dimensional spaces, we first construct a simplified teacher model as a surrogate environment from offline trajectories. Simultaneously, we infer a Deterministic Finite Automaton (DFA) using the RPNI algorithm to capture task dependencies. A policy is learned in the surrogate environment and transferred to the original domain via automaton distillation, which guides policy learning more effectively than direct RL in the original environment. Our framework integrates DQN for discrete tasks and DDPG/TD3 for continuous settings. Empirical results demonstrate that this structured transfer significantly improves learning efficiency, and convergence speed, outperforming standard RL baselines.

Keywords: Reinforcement Learning, Non-Markovian Decision Processes, Automaton Learning, Surrogate Environment Learning, Bidirectional Knowledge Transfer, Automaton Distillation.

1. Introduction

Reinforcement learning (RL) has demonstrated significant success in solving complex decision-making problems across diverse domains, including robotics Jens Kober and Peters (2013), autonomous systems Sutton and Barto (2018), and game-playing Mnih and et al. (2015); Silver et al. (2016). Standard RL algorithms, such as Q-learning Watkins and Dayan (1992) and policy gradient methods Schulman et al. (2017), typically operate under the Markov assumption, where rewards and transition probabilities depend solely on the current state and action. However, many real-world environments exhibit *non-Markovian* properties, where rewards depend on historical sequences of states and actions rather than the current observation alone Bacchus et al. (1997); Littman et al. (2017). Such dependencies introduce challenges for conventional RL approaches, as they require an agent to model and reason over temporal structures instead of relying on memory-less state representations Icarte et al. (2018); Camacho et al. (2019).

The problem becomes even more challenging in environments with continuous state and action spaces, where learning optimal policies requires function approximation, efficient exploration strategies, and generalization over high-dimensional inputs Lillicrap et al. (2016); Haarnoja et al. (2018). Handling non-Markovian rewards in such settings requires additional abstraction mechanisms, as agents must capture and incorporate historical dependencies over continuous representations Doya (2000); Barreto et al. (2017).

1.1. Contributions

In this work, we propose a bidirectional end-to-end RL framework designed for efficient policy learning in complex non-Markovian environments with both discrete and continuous dynamics. Unlike conventional RL approaches that directly optimize policies in complex environments, our framework introduces a structured, two-way information flow between an abstract teacher environment and the real-world target environment. Our approach consists of three key components:
1. Learning an abstract teacher environment: Instead of directly training in a complex environment with unknown reward dynamics, we first construct a simplified teacher environment as a surrogate simulator from an offline dataset of positive and negative trajectories. In continuous settings, we use discretization to transform the environment into a structured, lower-dimensional space, which facilitates learning. Other forms of abstraction will be explored in future work.
2. Passive automaton learning via RPNI: To address non-Markovianity, we employ the Regular Positive and Negative Inference (RPNI) algorithm Oncina and García (1993) to passively infer a Deterministic Finite Automaton (DFA) from observed agent trajectories. The learned DFA serves a threefold purpose: (i) encoding the temporal dependencies of the non-Markovian reward function, enabling structured reasoning over task progression without requiring interactive queries or predefined task specifications Camacho et al. (2019); Icarte et al. (2018), (ii) evaluating policies within the teacher environment, as both student and teacher share the same automaton-defined objective, and (iii) providing a compact, low-dimensional representation based on state labels (features) to facilitate policy transfer through distillation from teacher to automaton to student.
3. Knowledge transfer via automaton distillation: Once a policy is learned in the teacher environment, task knowledge is transferred back to the real target environment through automaton-guided distillation. The learned DFA and value function guide the student agent in policy optimization within the original, more complex environment. This knowledge transfer is seamlessly integrated into discrete RL settings via Q-learning and Deep Q-Networks (DQN) Mnih and et al. (2015) and extended to continuous control via Deep Deterministic Policy Gradient (DDPG) Lillicrap et al. (2016) and Twin Delayed DDPG (TD3) Fujimoto et al. (2018).

To the best of our knowledge, this is the first end-to-end framework that establishes a bidirectional information flow, integrating the learning of a simulator with a logical task representation in the form of a DFA, and bootstrapping knowledge back through automaton distillation. The DFA not only encodes task structure but also serves as a trajectory evaluator within the simulator, ensuring policies are assessed based on task progression.

1.2. Related Work

Non-Markovian Reinforcement Learning. Handling non-Markovian rewards has been widely studied, with various strategies such as augmenting state representations with memory Lin (1993), using recurrent neural networks for history encoding Bakker (2001), or leveraging temporal logic representations Bacchus et al. (1997); Camacho et al. (2019). Automata-based methods provide a structured means to model long-term dependencies in RL, facilitating learning by introducing task structure and guiding exploration Icarte et al. (2018); Giacomo et al. (2019).

Automaton Learning. Automaton learning techniques have been extensively explored in the context of formal verification and RL. Angluin's L^* algorithm Angluin (1987) provides a well-known active learning approach but requires interactive queries, making it unsuitable for passive data-driven learning. Instead, passive algorithms like RPNI Oncina and García (1993) infer DFA struc-

tures from observed sequences without explicit interactions. These methods have been used to extract task structures from RL trajectories Camacho et al. (2019); Icarte et al. (2018).

Automaton-Guided RL. Several approaches integrate automata into RL to enforce task constraints and improve exploration efficiency Hahn et al. (2019); Hasanbeig et al. (2020). While most methods assume a predefined automaton representation, we extend this paradigm by learning the automaton passively from data and using it as an integral component of bidirectional knowledge transfer. Our approach also differs from previous automaton-based RL methods in that we do not assume prior knowledge of teacher-student mappings or handcrafted reward shaping.

Transfer Learning and Automaton Distillation in RL. Transfer learning in RL enables agents to leverage prior knowledge from a source domain to improve learning in a target domain Taylor and Stone (2009). Existing approaches include policy distillation Rusu et al. (2015), successor feature transfer Barreto et al. (2017), and hierarchical skill transfer Konidaris et al. (2018). Automaton distillation Singireddy et al. (2023), a neuro-symbolic transfer learning approach, has also been explored for structured knowledge transfer in deep RL. However, prior methods assume predefined task structures, requiring an explicit automaton and a known mapping between teacher and student environments. Additionally, they typically rely on a manually specified reward function in the teacher environment, limiting their applicability in scenarios where rewards are unknown or sparse.

In contrast, our bidirectional framework removes these assumptions by jointly learning both the automaton and the teacher environment from offline positive and negative trajectories. This allows for structured transfer without predefined task representations or explicit reward engineering, making our approach more broadly applicable to real-world RL problems.

This paper is organized as follows. Section 2 provides necessary background and preliminaries. The problem formulation is presented in Section 3 along with an overview of our framework. Section 4 details our methodology. Section 5 presents experimental results and analysis. Finally, we conclude in Section 6. Additional experimental results and details are deferred to the appendix.

2. Background and Preliminaries

In this section, we provide the necessary background and formal definitions underlying our proposed framework. Our method addresses the challenges of learning in non-Markovian environments with unknown rewards and complex dynamics by constructing an intermediate teacher environment and learning structured representations.

Definition 1 (NMRDP) *A* Non-Markovian Reward Decision Process *(NMRDP) is defined by the tuple* $\mathcal{M} = (S, A, T, r, \gamma)$*, where* S *is a finite set of states,* A *is a finite set of actions, and* $T : S \times A \times S \to [0, 1]$ *specifies the probability* $T(s, a, s') = \Pr(s' \mid s, a)$ *of transitioning to state* $s' \in S$ *from* $s \in S$ *under action* $a \in A$*. Unlike standard MDPs, the reward function* r *in an NMRDP depends on the* history *of states and actions rather than only the current state-action pair. Formally,* $r : (S \times A)^* \to \mathbb{R}$ *assigns rewards based on full trajectories, allowing for richer task specifications that capture long-term dependencies. The parameter* γ *is a discount factor.*

The challenge in solving NMRDPs lies in the need for historical reasoning, which significantly complicates policy learning. This is mitigated in our approach through automaton learning, which extracts temporal dependencies and encodes them into structured representations, allowing agents to operate in an augmented Markovian state space $(S \times \Omega)$, where Ω represents automaton states.

Definition 2 (DFA) *A* Deterministic Finite Automaton *(DFA) is defined as a tuple* $\mathcal{A} = (\Omega, \Sigma, \delta, \omega_0, F)$*, where* Ω *represents a finite set of automaton states,* Σ *is a finite input alphabet,* $\delta : \Omega \times \Sigma \to \Omega$ *defines the transition function,* $\omega_0 \in \Omega$ *is the initial state, and* $F \subseteq \Omega$ *is the set of accepting (final) states. The automaton processes a sequence of symbols* $w = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*$ *by beginning in the initial state* ω_0 *and applying the transition function recursively as* $\omega_i = \delta(\omega_{i-1}, \sigma_i)$ *for* $i = 1, \ldots, n$*. A sequence* w *is accepted if and only if the final state* $\omega_n \in F$*.*

DFAs provide a compact representation of regular languages and are particularly useful for encoding task-specific sequences of events or subgoals in RL tasks. In our framework, DFAs are not assumed to be given but are inferred from positive and negative trajectories collected from an offline dataset. This bidirectional approach allows us to both extract structured knowledge from data, and use this knowledge to enhance policy learning.

Definition 3 (Product MDP) *Given an environment NMRDP* $\mathcal{M} = (S, A, T, R, \gamma)$ *and a DFA* $\mathcal{A} = (\Omega, \Sigma, \delta, \omega_0, F)$*, we construct the* product MDP $\mathcal{M} \otimes \mathcal{A}$ *Baier and Katoen (2008), denoted as* $\mathcal{M}' = (S', A, T', R', \gamma)$*. The state space of the product MDP is* $S' = S \times \Omega$*, augmenting the environment states with the automaton states. The transition function* $T' : S' \times A \times S' \to [0, 1]$ *follows* $T'((s, \omega), a, (s', \omega')) = T(s, a, s')$ *if* $\omega' = \delta(\omega, L(s'))$*, and* 0 *otherwise, where* $L : S \to \Sigma$ *is a labeling function mapping states to automaton symbols. The reward function* $R' : S' \times A \to \mathbb{R}$ *may incorporate automaton state information, and the discount factor* γ *remains unchanged.*

Solving the product MDP $\mathcal{M}'$ enables learning policies that account for both the environment dynamics and automaton-based task progression, effectively handling non-Markovian rewards.

3. Problem Formulation and Framework

3.1. Problem Formulation

We consider an RL problem where an agent interacts with a complex environment modeled as an NMRDP $\mathcal{M} = (S, A, T, R, \gamma)$, recalling that $R : (S \times A)^* \to \mathbb{R}$ is the unknown non-Markovian reward function. We are given an offline dataset $\mathcal{D} = \{(\tau_i, y_i)\}_{i=1}^N$, where each trajectory $\tau_i = (s_0^i, a_0^i, s_1^i, \ldots, s_{n_i}^i)$ consists of a sequence of states and actions, and $y_i \in \{+1, -1\}$ indicates whether the trajectory successfully achieved the task.

To capture the underlying task structure, in our framework we learn a DFA $\mathcal{A} = (\Omega, \Sigma, \delta, \omega_0, F)$ in which each state $s \in S$ in the environment is mapped to an automaton label via $L : S \to \Sigma$. A trajectory τ_i is successful if it induces an automaton state sequence $(\omega_0, \omega_1, \ldots, \omega_T)$ such that $\omega_T \in F$. However, learning a policy directly in $\mathcal{M}$ is challenging, due to the complexity of the environment, the unknown transition dynamics and the absence of a Markovian reward.

3.2. Bidirectional Learning Framework

To address these challenges, we propose a *bidirectional learning framework* that enables structured knowledge transfer between a teacher environment and the original complex environment as illustrated in Fig.1. This framework consists of two complementary information flows:

Student-to-Teacher Flow: We extract information from the student environment $\mathcal{M}$ by learning a surrogate teacher model $\widehat{\mathcal{M}} = (\widehat{S}, \widehat{A}, \widehat{T}, \widehat{R}, \gamma)$, where $\widehat{S}$ and $\widehat{A}$ represent an abstracted state-action

space, and $\widehat{T}$ is a learned approximation of the transition function, constructed from the dataset $\mathcal{D}$. This model provides a structured environment where policies can be learned efficiently.

Learning in the Teacher Environment: To incorporate task structure, we construct a product MDP $\widehat{\mathcal{M}}' = (\widehat{S}', \widehat{A}, \widehat{T}', \widehat{R}, \gamma)$, where $\widehat{S}' = \widehat{S} \times \Omega$. The transition function $\widehat{T}'$ jointly updates the environment and automaton state, $\widehat{T}'((s,\omega), a, (s',\omega')) = \widehat{T}(s,a,s') \cdot \mathbf{1}\{\omega' = \delta(\omega, L(s'))\}$. The automaton serves as an evaluator in this environment, enabling a policy π_{teacher} to be learned via

$$\pi_{\text{teacher}} = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \widehat{R}(s_t, \omega_t, a_t)\right]. \tag{1}$$

Teacher-to-Student Flow: Once a policy is learned in the teacher environment, knowledge is transferred back to the student environment via automaton distillation. This ensures that the student agent leverages structured knowledge from the teacher model while adapting to the complexities of the original environment. The final goal is to learn an optimal policy π^* that maximizes task success in the original environment:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\tau)\right], \tag{2}$$

where $R(\tau)$ is *implicitly learned* through DFA-based evaluation.

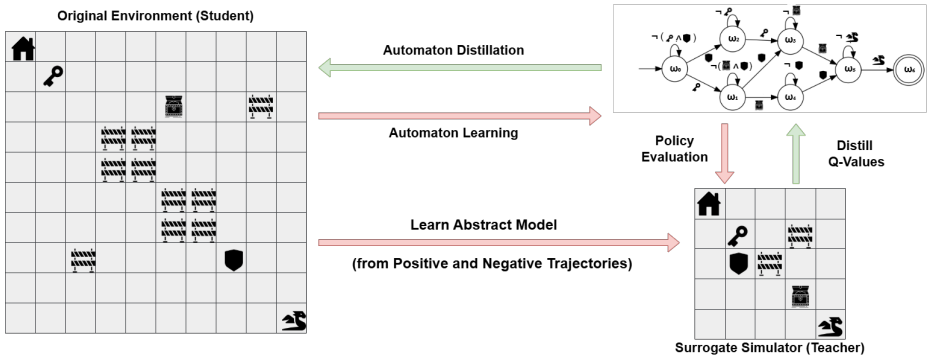

Figure 1: Overview of the Bidirectional Learning Framework. It enables structured knowledge transfer between a teacher and student environment, using automaton distillation for improved policy learning.

Next, we describe how our framework infers the automaton, constructs the teacher model, learns a policy in the teacher environment, and transfers knowledge back through automaton distillation.

4. Methodology

4.1. Passive DFA Learning with RPNI

To handle non-Markovian rewards, we capture temporal dependencies through a DFA. We employ the RPNI algorithm Oncina and García (1993) to infer a minimal DFA from *positive* and *negative* sequences extracted from the offline dataset.

Originally, RPNI was developed for grammar induction and formal language learning, where it passively infers a DFA that generalizes from labeled sequences of symbols. Unlike active learning approaches such as Angluin's L^* algorithm Angluin (1987), which require explicit queries, RPNI

operates passively, making it particularly suitable for offline RL settings, where only trajectory data is available. Our framework extends RPNI beyond its traditional setting by integrating it into an RL framework. Our insight is to leverage RPNI to capture temporal dependencies in tasks with non-Markovian reward structures, allowing us to infer structured representations that guide policy learning in environments with implicit task constraints.

Trajectory Labeling and Symbol Extraction. Each trajectory from the offline dataset $\mathcal{D}$ is mapped to a symbolic sequence using the labeling function $L : S \to \Sigma$. Given a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_n)$ from the original environment, we extract the corresponding symbolic sequence $w = L(s_0) \ldots L(s_n)$. Positive sequences (P) correspond to successful trajectories (ending in an accepting state), while negative sequences (N) correspond to failures.

Building and Merging. RPNI constructs a *Prefix Tree Acceptor* from P, then merges states so as not to accept any sequence in N. The final DFA $\mathcal{A} = (\Omega, \Sigma, \delta, \omega_0, F)$ encodes the essential subgoal or event ordering required for success. This automaton transforms a non-Markovian reward structure into a Markovian product MDP when combined with the NMRDP.

4.2. Surrogate Simulator Construction

We construct a **simplified (surrogate) simulator** from offline data, considering two cases: *(i) Discrete-to-Discrete*, when the original environment is discrete but large or partially unknown; and *(ii) Continuous-to-Discrete*, when the original environment has continuous state (and possibly action) spaces that must be discretized.

Discrete-to-Discrete: In this case, the original environment $\mathcal{M} = (S, A, T, R, \gamma)$ is discrete, but the exact transitions T or reward function R are not fully specified. Given the offline dataset $\mathcal{D}$ of trajectories, we extract a reduced state set $\widehat{S} \subseteq S$ and action set $\widehat{A} \subseteq A$. The transition function is estimated using frequency counts with optional Laplace smoothing:

$$\widehat{T}(s, a, s') \;=\; \frac{|\{(s, a, s') \in \mathcal{D}\}| + \alpha}{\sum_x (|\{(s, a, x) \in \mathcal{D}\}| + \alpha)}. \tag{3}$$

Since the environment is non-Markovian, the reward function is not directly defined in $\widehat{\mathcal{M}}$. Instead, we incorporate task progression information by augmenting the state space with an automaton state from the inferred DFA $\mathcal{A}$. This leads to the construction of a product MDP $\widehat{\mathcal{M}}' = \widehat{\mathcal{M}} \otimes \mathcal{A} = (\widehat{S} \times \Omega, \widehat{A}, \widehat{T}', \widehat{R}', \gamma)$, where Ω is the set of automaton states, and the state representation is now (s, ω), with ω tracking task progression. The reward function $\widehat{R}'$ is then given by

$$\widehat{R}'(s, \omega, a) = \begin{cases} +1, & \text{if } \omega \in F, \\ r(\omega, \omega'), & \text{otherwise,} \end{cases} \tag{4}$$

where $r(\omega, \omega')$ is structured to encourage progress along paths observed in successful trajectories (P) while avoiding transitions commonly associated with failure (N).

Continuous-to-Discrete: When the original environment is continuous, i.e., states $s_c \in \mathbb{R}^n$ and actions $a_c \in \mathbb{R}^m$, we construct a discretized surrogate simulator $\widehat{\mathcal{M}}$. We define discretization mappings $\phi_s : S \to \widehat{S}, \phi_a : A \to \widehat{A}$, which map continuous states and actions onto finite grids or prototype actions. Applying (ϕ_s, ϕ_a) to each transition $(s_c, a_c) \to (s_c')$ in $\mathcal{D}$ yields discrete trajectories $(s_d, a_d) \to (s_d')$. The transition function is estimated as:

$$\widehat{T}_d(s_d, a_d, s_d') \;=\; \frac{|\{(s_d, a_d, s_d') \in \mathcal{D}\}| + \alpha}{\sum_x (|\{(s_d, a_d, x) \in \mathcal{D}\}| + \alpha)}. \tag{5}$$

As in the discrete case, the reward function is not directly defined in $\widehat{\mathcal{M}}$. Instead, the product MDP $\widehat{\mathcal{M}}' = \widehat{\mathcal{M}} \otimes \mathcal{A}$ introduces the automaton state ω, ensuring task dependencies.

4.3. Automaton Distillation for Knowledge Transfer

We train the policy in the simplified teacher agent and use the automaton to distill knowledge from the teacher agent operating in the discretized environment $\widehat{\mathcal{M}}$ to the student agent in the target environment $\mathcal{M}$. In discrete environments, the teacher agent employs either tabular Q-learning Watkins and Dayan (1992) or Deep Q-Network (DQN) Mnih and et al. (2015) to learn an optimal policy in the product MDP $\widehat{\mathcal{M}} \otimes \mathcal{A}$, where the state space is augmented with automaton states.

For Q-learning, the teacher maintains a Q-table $Q_{\text{teacher}}(s_d, \omega, a_d)$, which is updated using the standard Q-learning rule:

$$Q_{\text{teacher}}(s_d, \omega, a_d) \leftarrow Q_{\text{teacher}}(s_d, \omega, a_d) + \alpha\big[r + \gamma \max_{a'_d} Q_{\text{teacher}}(s'_d, \omega', a'_d) - Q_{\text{teacher}}(s_d, \omega, a_d)\big],$$

where α is the learning rate, γ is the discount factor, r is the defined intermediate reward, and (s'_d, ω') is the next state. In the case of DQN, the agent leverages an experience replay buffer $\mathcal{ER}_T$ to store tuples $((s_d, \omega), a_d, r, (s'_d, \omega'))$ and performs batch updates using a neural network.

To distill knowledge into the automaton, we extract Q-values corresponding to automaton transitions. For each transition $(\omega, \sigma) \to \omega'$, the average Q-value is computed as:

$$Q^{\text{avg}}_{\text{teacher}}(\omega, \sigma) = \frac{1}{N_{\omega,\sigma}} \sum_{i=1}^{N_{\omega,\sigma}} Q_{\text{teacher}}(s^i_d, a^i_d),$$

where $N_{\omega,\sigma}$ is the count of occurrences of the transition (ω, σ), (s^i_d, a^i_d) are the corresponding state-action pairs, and $Q_{\text{teacher}}(s^i_d, a^i_d)$ is the estimated Q-value.

The student agent operates in the target environment $\mathcal{M}$ and augments its state with the automaton state ω. The student updates its Q-values by incorporating the distilled knowledge:

$$\begin{aligned} Q_{\text{student}}(s, \omega, a) \leftarrow Q_{\text{student}}(s, \omega, a) + \alpha\big[&\beta(\omega, \sigma) Q^{\text{avg}}_{\text{teacher}}(\omega, \sigma) + \\ &(1 - \beta(\omega, \sigma))(r + \gamma \max_{a'} Q_{\text{student}}(s', \omega', a')) - Q_{\text{student}}(s, \omega, a)\big]. \end{aligned}$$

$\beta(\omega, \sigma) = \rho^{\eta_{\text{student}}(\omega,\sigma)}$ controls the influence of the teacher's knowledge, where $\rho \in (0, 1)$ is a decay parameter and $\eta_{\text{student}}(\omega, \sigma)$ tracks the frequency of the transition (ω, σ) in student training.

For DQN, the student agent employs a neural network to approximate Q-values and optimizes the loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{((s,\omega),a,r,(s',\omega'))}\left[\left(Q'_{\text{student}}((s, \omega), a) - Q_{\text{student}}((s, \omega), a; \theta)\right)^2\right],$$

where $Q'_{\text{student}}((s, \omega), a)$ is the adjusted target Q-value incorporating the distilled automaton knowledge. The student's Q-table or network is initialized with prior knowledge and continuously updated. This automaton-guided knowledge transfer accelerates policy learning and ensures efficient exploration in complex environments. We use a similar approach for transfer from discrete to continuous environments, using Deep Deterministic Policy Gradient (DDPG) or Twin Delayed DDPG (TD3). The details are deferred to an extended version of this work.

5. Experiments

We evaluate the effectiveness of our proposed end-to-end framework on several complex environments with varying levels of task dependencies and complexities. We compare our approach against baseline methods to demonstrate its advantages in handling non-Markovian tasks and facilitating efficient learning through automaton distillation.

Environments. We selected four benchmark environments that embody different challenges related to non-Markovian rewards, sequential dependencies, and continuous dynamics. Given the complexity of the original environments (*student*), we construct simplified (*teacher*) simulators to facilitate structured policy learning before transferring to the original domain. Here, we present one primary environment (*Dungeon Quest*), with additional environments detailed in the Appendix.

The student environment consists of either a 10×10 *discrete* grid (See Fig. 1) or a continuous 10.0×10.0 coordinate space, where the agent must collect a *Key* to unlock the *Chest*, retrieve the *Sword*, and obtain a *Shield* before confronting the *Dragon*. In the continuous case, items are placed in small circular regions, and the agent moves by applying continuous actions. Task dependencies enforce a strict sequence of subgoals regardless of discretization. Additional benchmark environments (*Blind Craftsman*, *OfficeWorld*, and *Minecraft Building Bridge*), follow a similar setup and are detailed in Appendix A.

Surrogate Construction, DFA Learning and Teacher Agent (Teacher Side). In the Dungeon Quest domain, we constructed a teacher model $\widehat{\mathcal{M}}$ as a reduced 5×5 version of the environment while also inferring a DFA from offline trajectories to capture the task structure. To assess generalization, we considered two student settings: a discrete student environment, where the original Dungeon Quest remains a 10×10 discrete grid, and a continuous student environment, where state and action spaces are continuous and require discretization before training.

For both settings, we trained a teacher policy using Q-learning or DQN in the product MDP $\widehat{\mathcal{M}} \otimes \mathcal{A}$, where the DFA augments the state space. The teacher policy learned key subtasks, such as item collection and defeating the dragon, while leveraging structured task information from the automaton. Figure 1 illustrates this process, showing the original discrete 10×10 environment, the inferred DFA, and the learned surrogate model.

We evaluated the agent's performance in the surrogate teacher environment across discrete-to-discrete and continuous-to-discrete settings. The teacher policy was trained in the product MDP $\widehat{\mathcal{M}} \otimes \mathcal{A}$, where the DFA captures task dependencies. Performance was measured using reward per episode, steps per episode, and reward per step. The results in Fig. 2 show that the surrogate simulators effectively model the non-Markovian structure, enabling the teacher agent to learn structured policies for item collection and dragon defeat while achieving stable convergence.

Transfer to the Original Environment (Student Side). After training in the surrogate environment, we transfer the teacher's automaton-aware Q-values to a student agent operating in the original environment. This process enables structured learning by annealing the teacher's estimates with the student's on-policy updates (see Section 4.3). We evaluate this transfer in the Dungeon Quest domain for both discrete and continuous student environments. In the discrete setting, the student agent operates in the full 10×10 Dungeon Quest environment, which includes additional obstacles and unobserved states. It initializes with the distilled Q-values from the teacher's policy, accelerating convergence compared to learning from scratch. The agent then updates its policy using Q-learning or DQN, leveraging the transferred structured knowledge to efficiently complete the task. For the continuous student environment, the agent receives automaton-guided Q-values and

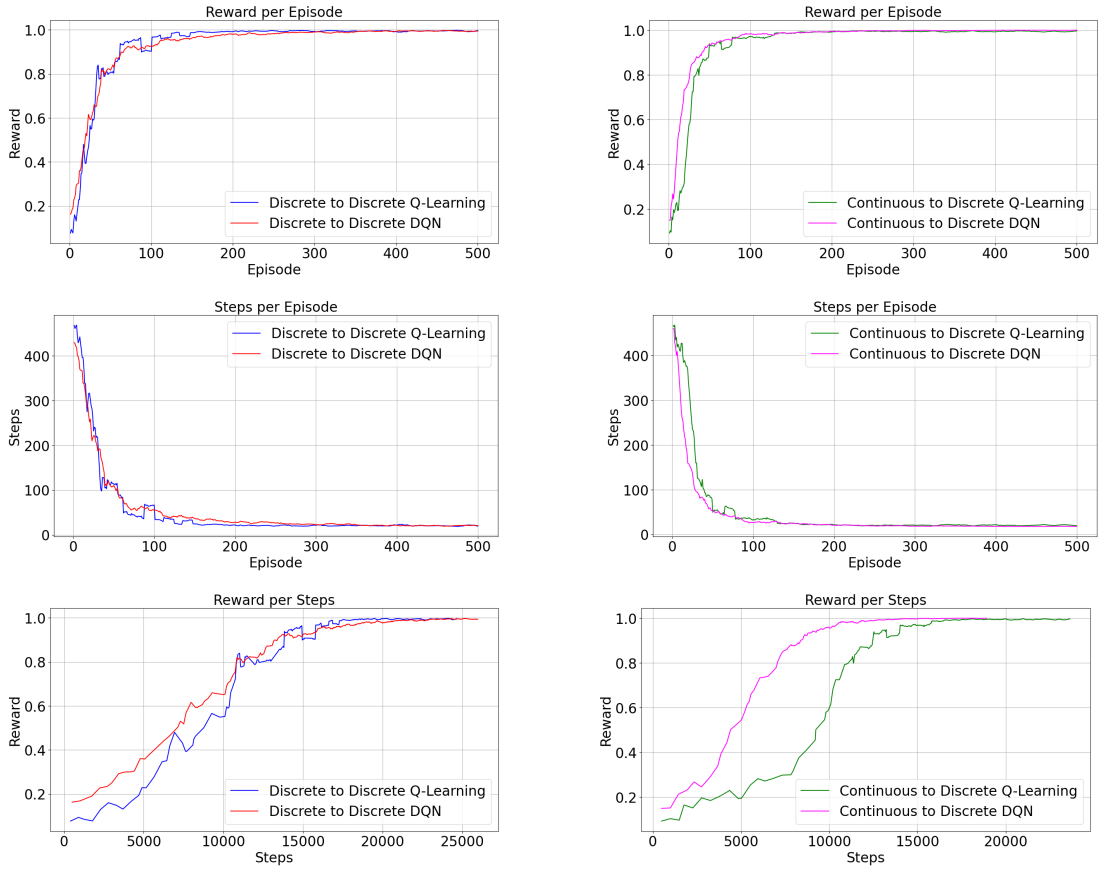

Figure 2: Teacher performance in the surrogate simulators for Dungeon Quest. (**Left**): Discrete-to-discrete. (**Right**): Continuous-to-discrete. Rows show (**top**) reward per episode, (**middle**) steps per episode, and (**bottom**) reward per step. In both modes, the teacher converges to a stable policy capturing key subgoals.

applies DDPG or TD3 to learn an effective control policy. The structured guidance from the teacher model enables the agent to efficiently collect items and defeat the dragon despite the complexity of continuous state and action spaces.

The results in Fig. 3 demonstrate that automaton distillation (AD) effectively enhances learning speed and policy quality in both modes. The structured knowledge transfer from the teacher enables the student agent to efficiently recover subgoal dependencies and optimize decision-making. This bidirectional learning approach ensures that complex non-Markovian dependencies, originally difficult to learn, are successfully incorporated into the final policy. We also compare against Q-learning and DQN in an augmented state space (QAS), where an additional binary vector encodes task-relevant information, increasing state space complexity.

6. Conclusion

We proposed a bidirectional RL framework that integrates surrogate environment construction, passive DFA inference via RPNI, and automaton distillation to handle non-Markovian tasks. By discretizing complex environments into simpler surrogates, we preserved essential task structure while enabling efficient automaton learning without interactive queries. The learned automaton captures

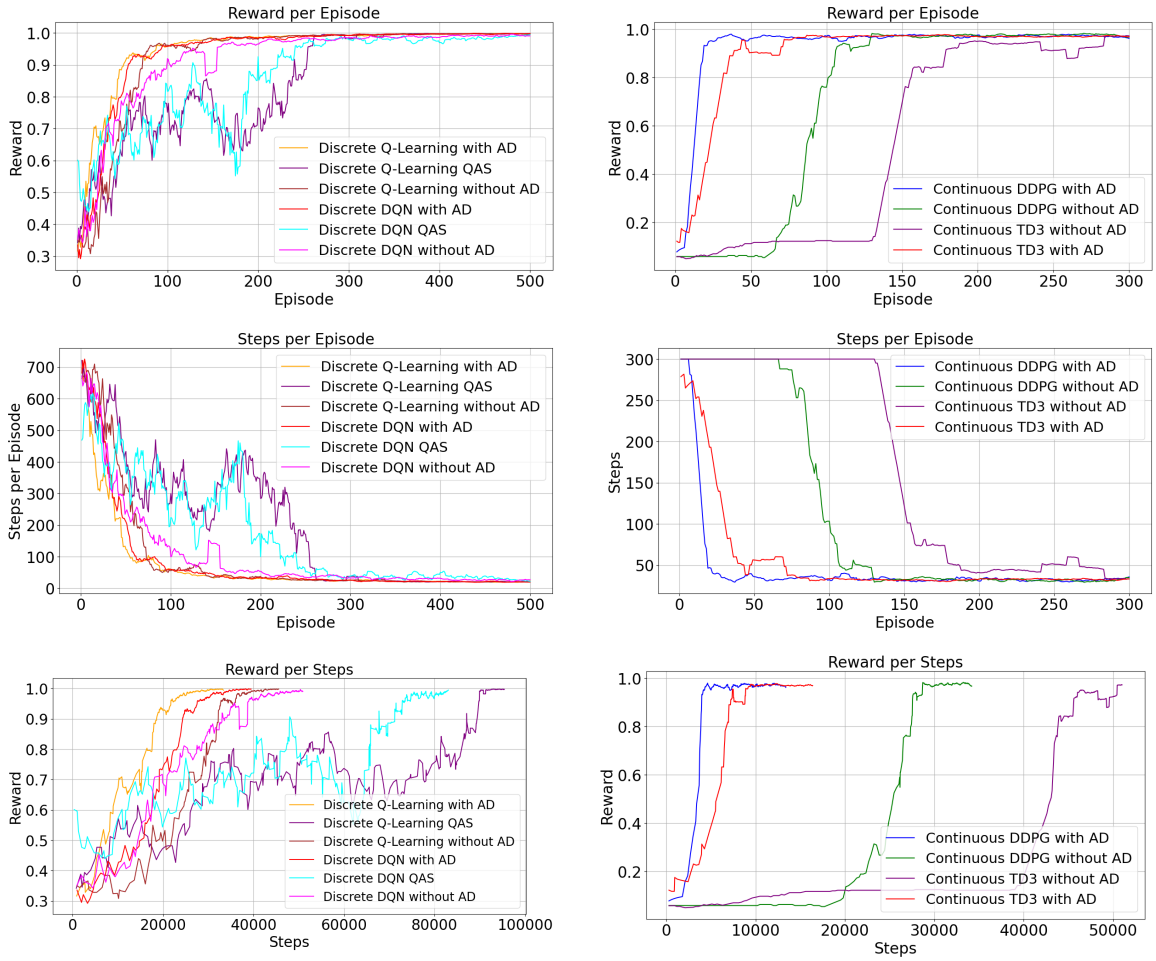

Figure 3: Student performance in the original Dungeon Quest environment after knowledge transfer. (**Left column**): Discrete-to-discrete mode; (**Right column**): Continuous-to-discrete mode. Rows show (**top**) reward per episode, (**middle**) steps per episode, and (**bottom**) reward per step.

sequential dependencies, which we leveraged to distill automaton-guided Q-values back into the original environment, allowing the student agent to refine its policy through a structured blend of teacher estimates and self-learning.

Our approach leverages neurosymbolic methods for transfer, serving multiple purposes: (i) addressing the non-Markovian nature of the reward structure, (ii) utilizing the emerging lower-dimensional automaton representation to assess policy quality within the learned surrogate, and (iii) enabling efficient transfer through distillation techniques by providing a compact, structured representation for knowledge transfer between teacher and student environments.

Empirical results demonstrated that this two-way information flow significantly accelerates convergence while maintaining interpretability. Our framework supports both discrete and continuous state-action spaces, providing a scalable alternative to methods requiring full knowledge of environment dynamics. Future extensions include hierarchical RL, deeper function approximation, and real-time adaptation in partially observable tasks.

Acknowledgments

This work was supported by DARPA under Agreement No. HR0011-24-9-0427 and NSF under Award CCF-2106339 and NSF Cooperative Agreement No. EEC-2133516.

References

Dana Angluin. Learning regular sets from queries and counterexamples. In *Information and Computation*, volume 75, pages 87–106, 1987.

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Structured solution methods for non-markovian decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 112–117, 1997.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008. ISBN 978-0-262-02649-9.

Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*, 2001.

Andre Barreto, Will Dabney, Remi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Reid McIlraith, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6065–6073, 2019. doi: 10.24963/ijcai.2019/840.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1): 219–245, 2000.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 1587–1596. PMLR, 2018.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, pages 128–136. AAAI Press, 2019. doi: 10.1609/icaps.v29i1.3549.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018. arXiv preprint arXiv:1812.05905.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In Tomas Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 11427 of *Lecture Notes in Computer Science*, pages 418–435. Springer, 2019.

Mohammad Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious reinforcement learning with logical constraints. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 483–491. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

J. Andrew Bagnell Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

George Konidaris, Leslie P. Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. Ph.d. dissertation, Carnegie Mellon University, 1993. Order No. 9322750.

Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL, 2017. arXiv preprint arXiv:1704.04341.

Volodymyr Mnih and et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

Josep Oncina and Pedro García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108, 1993.

Andrei A. Rusu, Mohammad Babaeizadeh, Ivo Danihelka, Mohammad Grefenstette, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *arXiv preprint arXiv:1511.06295*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. arXiv preprint arXiv:1707.06347.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

Suraj Singireddy, Precious Nwaorgu, Andre Beckus, Aden McKinney, Chinwendu Enyioha, Sumit Kumar Jha, George K. Atia, and Alvaro Velasquez. Automaton distillation: Neuro-symbolic transfer learning for deep reinforcement learning, 2023. arXiv preprint arXiv:2310.19137.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

Appendix A. Additional Benchmark Environments

A.1. Blind Craftsman

The student environment consists of a 12×12 grid where the agent must collect resources and craft tools before returning home. The environment features *Wood Sources*, where up to two pieces of wood can be collected at a time, a *Factory* where tools are crafted, and *Home*, the final goal. The agent must alternate between gathering wood and crafting tools until three tools are made. Task dependencies enforce that tools cannot be crafted without wood, and the agent cannot proceed home without completing the crafting process.

The teacher environment is a simplified 6×6 version with fewer obstacles, maintaining the essential task dependencies while reducing complexity for faster learning.

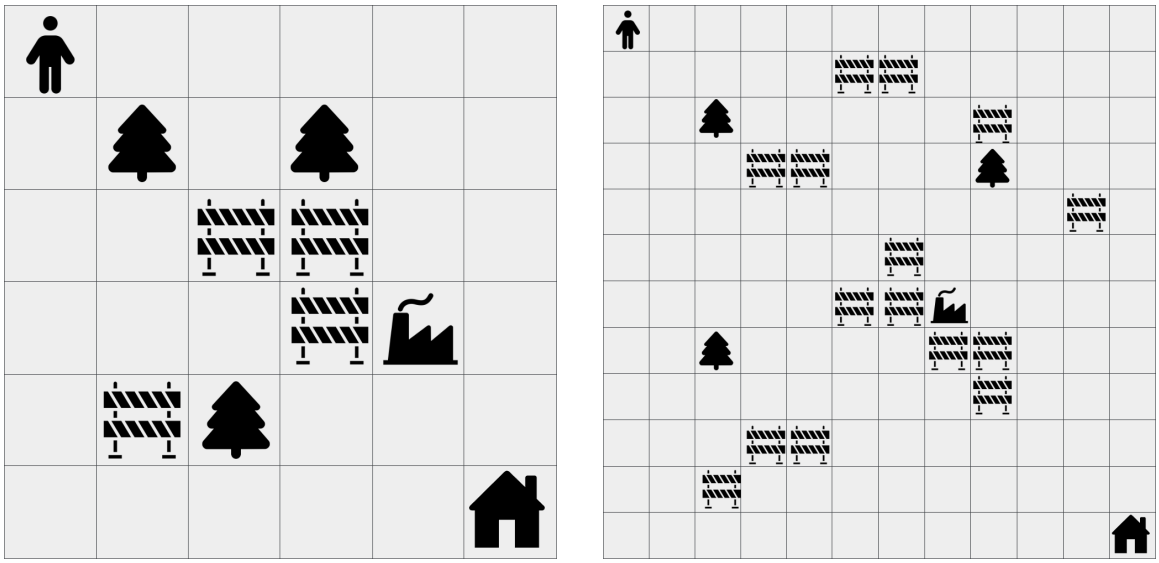

Figure 4: Blind Craftsman Environments: (Left) Teacher Environment (6×6 grid with 4 obstacles); (Right) Student Environment (12×12 grid with 15 obstacles).

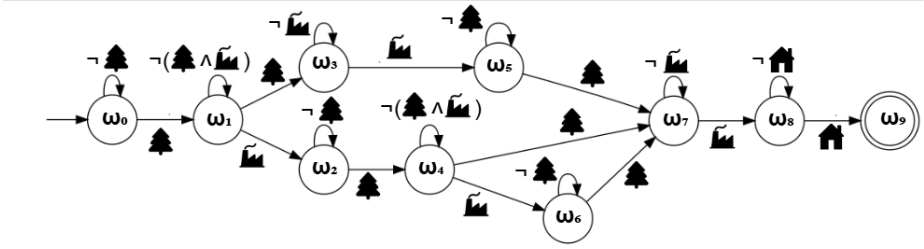

Figure 5: DFA representing the sequential dependencies in the Blind Craftsman environment. This automaton encodes the required subgoal progression, ensuring the agent follows the correct order of tasks in both the Teacher and Student Environments.

A.2. OfficeWorld

The student environment consists of a 9×12 grid representing an office environment with interconnected rooms. The agent must first collect *Coffee* to gain energy, then retrieve the *Mail*, and finally deliver it to the *Office*. Task dependencies enforce that mail cannot be delivered unless both coffee and mail have been collected in the correct order.

The teacher environment is a smaller 6×9 version with fewer rooms and obstacles, maintaining the sequential task structure while simplifying the navigation challenge.

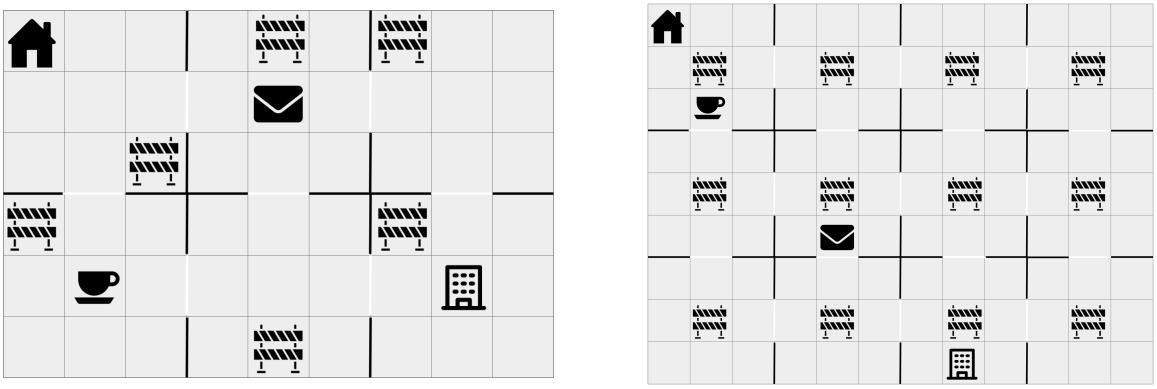

Figure 6: Office Environments: (Left) Teacher Environment (6×9 grid with 6 obstacles); (Right) Student Environment (9×12 grid with 12 obstacles).

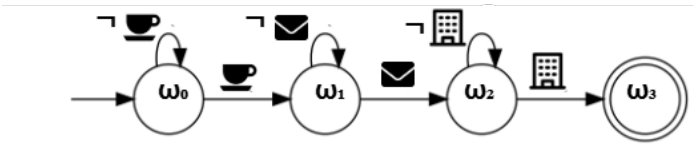

Figure 7: DFA representing the sequential dependencies in the Office environment. This automaton encodes the required subgoal progression, ensuring the agent follows the correct order of tasks in both the Teacher and Student Environments.

A.3. Minecraft Building Bridge

The student environment consists of a 20×20 grid where the agent must gather *Wood* and *Iron* and transport them to a *Factory* for bridge construction. The task requires sequential planning, as construction cannot begin until both materials have been collected.

The teacher environment is a simplified 8×8 grid with fewer obstacles, maintaining the multi-stage structure while reducing complexity for efficient learning.

Appendix B. Performance in Additional Student Environments

After training in the surrogate environment, the learned automaton-aware Q-values are transferred to a student agent operating in the original environment. This structured transfer accelerates learning

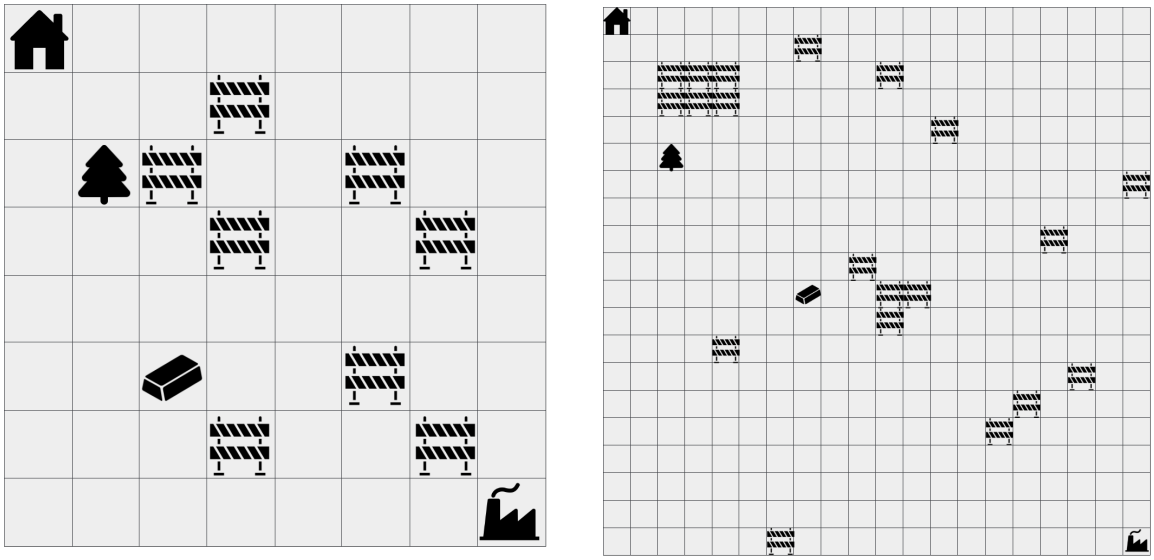

Figure 8: Bridge Environments: (Left) Teacher Environment (8×8 grid with 8 obstacles); (Right) Student Environment (20×20 grid with 20 obstacles).

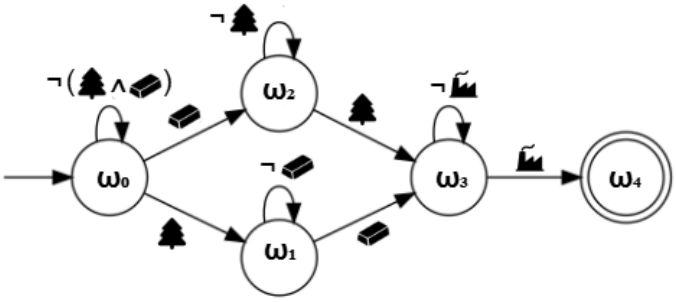

Figure 9: DFA representing the sequential dependencies in the Bridge environment. This automaton encodes the required subgoal progression, ensuring the agent follows the correct order of tasks in both the Teacher and Student Environments.

by leveraging task dependencies, significantly improving reward acquisition and efficiency. We evaluate this transfer in the *Blind Craftsman*, *OfficeWorld*, and *Building Bridge* environments.

B.1. Blind Craftsman

The student agent operates in the full 12×12 Blind Craftsman environment, which introduces additional obstacles and a larger action space. The goal remains to collect wood, craft tools at the factory, and return home. The automaton-guided Q-values allow the student to learn a structured crafting strategy, significantly improving efficiency over direct learning.

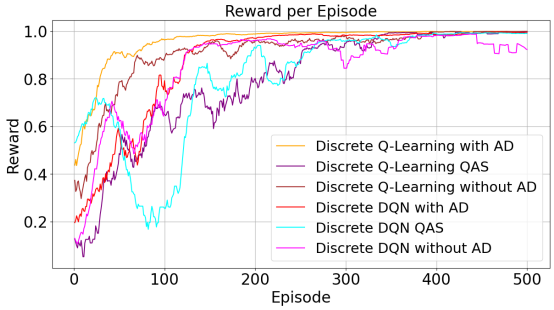

Figure 10: Reward per Episode in Blind Craftsman.

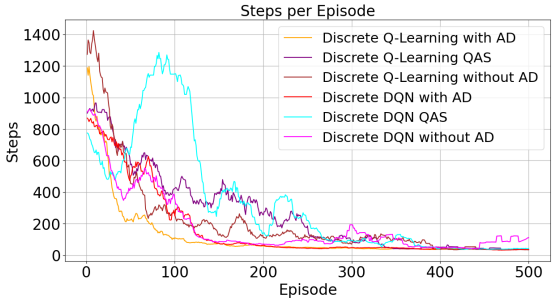

Figure 11: Steps per Episode in Blind Craftsman.

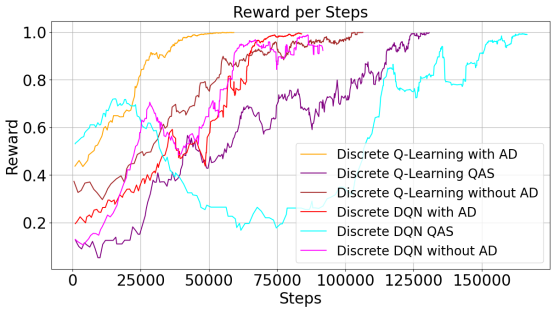

Figure 12: Reward per Step in Blind Craftsman.

B.2. OfficeWorld

In the full 9×12 OfficeWorld environment, the student must collect and deliver items in the correct order. The teacher's automaton-aware Q-values accelerate policy learning, allowing the agent to optimize item collection and delivery efficiently.

B.3. Minecraft Building Bridge

The 20×20 Building Bridge environment requires resource collection and sequential bridge construction. The student agent receives structured Q-values from the teacher, reducing unnecessary exploration and improving task completion efficiency.

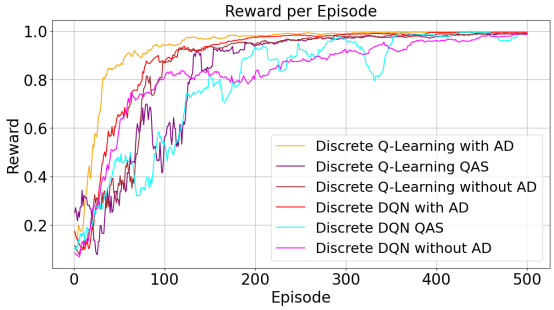

Figure 13: Reward per Episode in OfficeWorld.

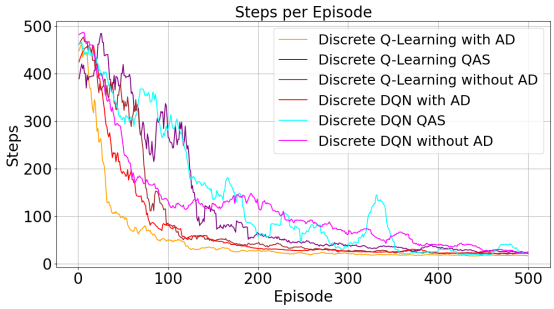

Figure 14: Steps per Episode in OfficeWorld.

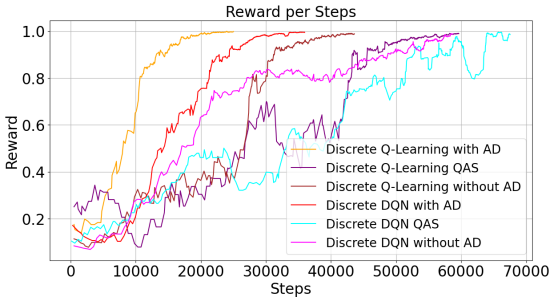

Figure 15: Reward per Step in OfficeWorld.

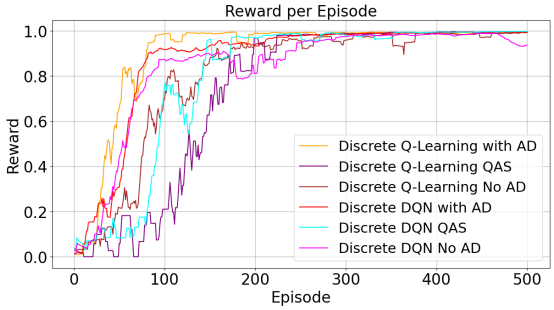

Figure 16: Reward per Episode in Building Bridge.

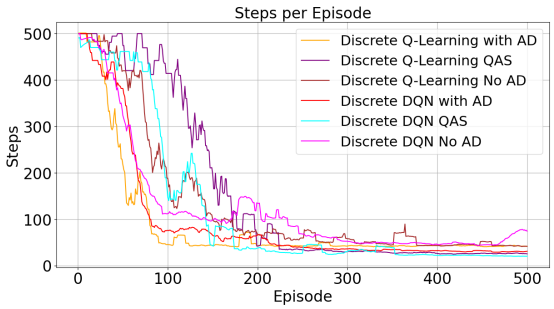

Figure 17: Steps per Episode in Building Bridge.

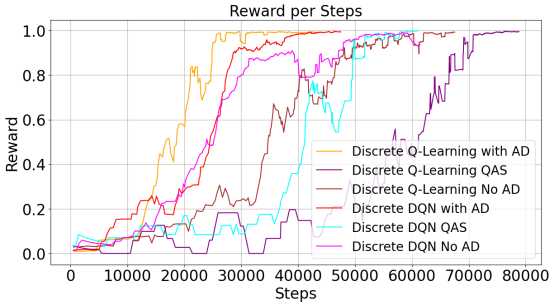

Figure 18: Reward per Step in Building Bridge.