PREFERENCE-GUIDED REINFORCEMENT LEARNING WITH AUTOMATON-BASED SPECIFICATIONS IN NON-MARKOVIAN SETTINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning (RL) in environments with complex, history-dependent reward structures poses significant challenges for traditional methods. In this work, we introduce a novel approach that leverages automaton-based feedback to guide the learning process, replacing explicit reward functions with preferences derived from a deterministic finite automaton (DFA). Unlike conventional approaches that use automata for direct reward specification, our method employs the structure of the DFA to generate preferences over trajectories that are used to learn a reward function, eliminating the need for manual reward engineering. Our framework introduces a static approach that uses the learned reward function directly for policy optimization and a dynamic approach that involves continuous refining of the reward function and policy through iterative updates until convergence. Our experiments in both discrete and continuous environments demonstrate that our approach enables the RL agent to learn effective policies for tasks with temporal dependencies, outperforming traditional reward engineering and automaton-based baselines such as reward machines and LTL-guided methods. Our results highlight the advantages of automaton-based preferences in handling non-Markovian rewards, offering a scalable, efficient, and human-independent alternative to traditional reward modeling. We also provide a convergence guarantee showing that under standard assumptions our automaton-guided preference-based framework learns a policy that is near-optimal with respect to the true non-Markovian objective.

1 INTRODUCTION

Reinforcement Learning (RL) has achieved remarkable success across various domains, ranging from game playing Mnih & et al. (2015); Silver & et al. (2016); Vinyals & et al. (2019) to robotic control Levine & Abbeel (2016); Lillicrap & et al. (2016) and autonomous driving Kendall & et al. (2019). However, most RL algorithms rely on explicit reward functions carefully designed to specify the agent's objectives Sutton & Barto (1998). In practice, designing such reward functions can be difficult, especially in environments where the desired outcomes depend on long-term dependencies or complex sequences of actions Littman & et al. (2017); Camacho & et al. (2019). Such environments exhibit *history-dependent* or *non-Markovian* reward structures, where rewards are not solely determined by the current state but by the trajectory leading to that state Bacchus & Renold (1996); Icarte & et al. (2021). These structures pose significant challenges to traditional RL methods designed to operate within a Markovian framework Icarte & et al. (2018); Sutton & Barto (1998).

To address this gap, we propose a novel approach that bypasses the need for manually specifying explicit reward functions. Instead, we leverage automaton-based feedback to generate preferences over trajectory segments. Our method uses a deterministic finite automaton (DFA) to capture complex temporal dependencies in tasks Oncina & García (1992); Angluin (1987), providing structured feedback to guide the learning process. The agent then scores complete trajectories by their alignment with subgoals, yielding rewards that mirror the automaton's encoded preferences Icarte & et al. (2018); Camacho & et al. (2019); Li & et al. (2017).

Motivation and Challenges. In many real-world scenarios, reward signals are sparse or challenging to define explicitly Christiano & et al. (2017); Andrychowicz & et al. (2017). For instance, robotic

assembly tasks often involve executing a sequence of operations in a specific order, where intermediate actions provide little to no feedback Kaelbling (1993); Hart & Grupen (2009). Such tasks naturally result in non-Markovian reward structures, making it difficult for RL algorithms to infer optimal policies from state-action transitions Littman & et al. (2017); Bacchus & Renold (1996). Traditional methods either assume dense reward functions or rely on manually engineered reward shaping Ng & Russell (1999); Taylor & Stone (2009), both of which can introduce bias or require significant effort Rusu & et al. (2015); Barto & Mahadevan (2003).

Our method addresses this challenge by abstracting task specifications into a formal structure (the DFA), enabling agents to learn policies aligned with the temporal logic encoded in the automaton. *The fundamental distinction of our approach is not that we avoid learning reward functions—indeed, both our method and traditional automaton-based RL ultimately learn rewards—but rather that we eliminate the manual reward engineering bottleneck.* Traditional methods require practitioners to manually translate logical specifications into numeric rewards, a process demanding meticulous calibration of reward magnitudes, transition bonuses, and penalty structures to prevent unintended behaviors or convergence to local optima. Our contribution uses the automaton structure to generate trajectory preferences automatically, from which reward functions are learned without human intervention in the numeric reward assignment process. By using DFAs to provide trajectory preferences, we eliminate the need for explicit reward engineering, making the approach more generalizable.

While reward machines and Linear Temporal Logic (LTL)-based methods also use formal methods to address non-Markovian rewards, they typically require translating logical specifications into numeric rewards—still demanding careful tuning. In contrast, our preference-based method ranks trajectories by their adherence to the automaton structure, avoiding the challenges of numeric reward design. This matters because even with a formal specification, assigning suitable numeric values to transitions or subgoals often requires calibration to prevent unintended behaviors. Moreover, while reward signals from experts are hard to obtain, they can often provide natural language guidance convertible to DFAs via tools like NL2TL Chen et al. (2023). Prior work Neider et al. (2021) used DFAs as advice to resolve conflicting preferences and handle sparse-reward tasks, and Yang et al. Yang et al. (2023) used DFAs for fine-tuning without ranking. In contrast, we use automata to generate preferences for reward learning, aligning policies with task logic without directly shaping rewards.

Contributions. Our work presents a new framework combining formal methods with modern RL techniques to address tasks with complex, history-dependent requirements, connecting task specification and policy optimization. Our contributions are summarized as follows.

Automaton-based preference learning: We introduce a framework for generating preferences over trajectory segments using a DFA, enabling the RL agent to learn reward functions that capture task-specific temporal dependencies without human intervention.

Scoring functions for preference generation: We propose various scoring functions derived from the DFA representation namely, (i) *subtask-based scoring* in which preferences are assessed based on the number of subtasks the agent has completed and a distance metric to the next subtask in a sequence of decomposable tasks, and (ii) *Q-value-based scoring* where we utilize Q-values to assign values to specific transitions within the automaton. The latter approach not only facilitates preference generation but also supports efficient transfer learning between different environments that share the same objective, enhancing the agent's ability to generalize and adapt to new settings.

Static and dynamic learning variants: We present two variants of our method. In the static version, the learned reward function is applied directly for policy optimization. In the dynamic version, the reward function and policy are iteratively refined in a loop until convergence, providing a robust framework for complex tasks Sutton et al. (1999); Bacon et al. (2017).

Theoretical guarantee: We prove that, assuming preference consistency, reward expressivity, and sufficient exploration, our automaton-guided preference-based framework learns a policy that is ε-optimal with respect to the true non-Markovian objective (See Theorem 4.5 and Theorem A.1).

Empirical validation: We evaluate our approach in both grid-based and continuous domains, showing that it learns effective policies under non-Markovian rewards. Results demonstrate that automaton-based preferences provide a scalable, efficient, and human-independent alternative to traditional reward shaping, often outperforming methods like reward machines Ng & Russell (1999); Taylor & Stone (2009) and LTL-based approaches.

2 Related Work

Reward learning from preferences: Preference-based learning offers an alternative to explicit reward signals. The seminal work Christiano & et al. (2017) introduced reward learning from human-ranked trajectory segments, reducing reliance on manual reward design but still requiring human input. Building on this, Zhu et al. (2023) proposed a principled RLHF framework using pairwise or K-wise comparisons. Recent advances include Direct Preference Optimization (DPO) Rafailov & et al. (2023), Contrastive Preference Learning Park et al. (2022), active preference elicitation Biyik et al. (2020), ensemble uncertainty estimation Gleave et al. (2022), and multi-objective alignment Leike et al. (2018). Classic approaches include learning from demonstrations Ho & Ermon (2016) and inverse RL Abbeel & Ng (2004). Our method extends this line by replacing human feedback with automaton-based preferences Oncina & García (1992); Angluin (1987), eliminating human involvement in reward learning while retaining structured feedback. Though DFAs require initial domain knowledge, our approach removes humans from the learning loop itself.

Non-Markovian reward decision processes: Non-Markovian rewards pose challenges in RL, where rewards typically depend only on the current state Sutton & Barto (1998). Solutions include augmenting state representations with history Bakker (2002); Wierstra & et al. (2007) or designing reward functions over action sequences Dupont & et al. (1996); Li & et al. (2017). For instance, Littman & et al. (2017) translated temporal logic into automata for RL integration. Reward machines Icarte et al. (2022) represent reward structures via finite-state machines, with QRM enabling efficient learning through counterfactual reasoning. However, these methods still require manually converting logic into numeric rewards, which can be error-prone and require tuning. Our approach instead learns rewards from automaton-based preferences Walkinshaw et al. (2016); Xu & et al. (2020), avoiding reward translation while leveraging the automaton's structure, eliminating manual reward design.

Logic-based RL: LTL has been widely used to formalize task requirements in RL. SPECTRL Jothimurugan et al. (2019) uses a depth-based measure for sequential tasks, while methods like LPOPL Hasanbeig et al. (2018); Voloshin et al. (2022) and TLTL Li & Belta (2017) embed temporal logic into RL objectives. Eventual Discounting Counterfactual Replays Voloshin et al. (2023) handle task sequences efficiently. Recent work includes goal-conditioned RL with temporal logic Qiu et al. (2023), compositional RL from logic Jothimurugan et al. (2021), and automata embeddings for RL Yalcinkaya & Xu (2024). LLM2LTL Shah et al. (2023) bridges natural language and formal logic, while Boker et al. (2022) explores automata-to-LTL translation. Unlike these methods, which embed logic directly into rewards or acceptance criteria, often causing reward sparsity or scale issues, our approach uses the automaton to generate trajectory preferences. This enables learning reward functions with structured guidance while avoiding manual reward engineering.

Neurosymbolic RL: Symbolic structures like automata have been used to formalize task specifications in RL, particularly under temporal logic constraints Camacho & et al. (2019); Icarte & et al. (2018). They enable agents to reason about action sequences and enforce constraints Baier & Katoen (2008); Li & et al. (2017). Hasanbeig & et al. (2020) integrated automata with deep RL for safety, while shielding methods use automata to block unsafe actions Alshiekh & et al. (2018). Yang et al. Yang et al. (2023) fine-tuned language models with automaton-based controllers guided by natural language, and Neider et al. (2021) used DFAs as advice to resolve conflicting preferences in sparse-reward settings. Reward-conditioning Kumar et al. (2022) guides behavior via reward-to-go, and hierarchical RL Barto & Mahadevan (2003); Bacon et al. (2017) decomposes tasks using formal guidance. While these methods use automata to constrain or guide policies, our approach uniquely leverages automata to generate trajectory preferences for reward learning, aligning policies with task logic without direct reward shaping Hahn & et al. (2019); Icarte & et al. (2021).

Transfer learning in structured environments: Transfer in RL has been studied via policy distillation Rusu & et al. (2015), feature transfer Barreto et al. (2017), and meta-learning Finn et al. (2017). For structured tasks, modular policies guided by task sketches Andreas et al. (2017a) and transfer across LTL-specified tasks Neider et al. (2021) have been explored. Closest to our work, reward machines Icarte & et al. (2021) enable sharing reward structures across tasks. We extend this line by showing that automaton-based preferences can effectively transfer task knowledge across environments using our Q-value-based scoring. Unlike prior methods that transfer full policies or value functions, our approach transfers knowledge via automaton transitions, offering a more modular and interpretable mechanism.

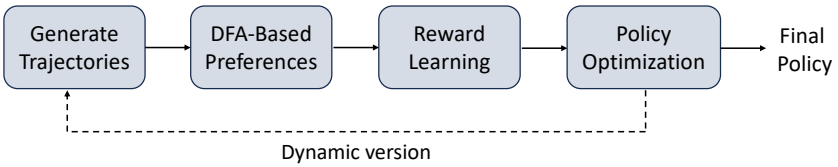

Figure 1: Automaton-based RL framework: The agent generates trajectories, evaluated by the DFA to produce preferences for reward learning via pairwise ranking loss. The learned reward guides policy optimization. In the dynamic version (dashed arrow), this loop repeats until convergence; in the static version, the policy is optimized once after reward learning.

3 Problem Setup

Definition 3.1. A Non-Markovian Reward Decision Process (NMRDP) is a decision process defined by the tuple $\mathcal{M} = (S, s_0, A, T, r)$, where S is a finite set of states, $s_0 \in S$ is the initial state, A is the finite set of actions, $T : S \times A \to \Delta(S)$ is the transition probability function, specifying the probability of reaching state $s' \in S$ given state $s \in S$ and action $a \in A$, and $r : (S \times A)^* \to \mathbb{R}$ is a reward function that depends on the history of states and actions.

The reward signal of an NMRDP incorporates non-Markovian dependencies, allowing it to rely on the full sequence of previous states and actions rather than solely on the current state-action pair.

We also define a labeling function $L : S \to 2^{AP}$, mapping the states of the environment to relevant abstract properties represented by a set of atomic propositions AP.

The agent policy $\pi : S \to \Delta(A)$, where $\Delta(A)$ is the probability simplex on A, determines the probability of selecting a given action in a certain state. The goal is to maximize the expected cumulative reward:

$$V_R(\pi) \;=\; \mathbb{E}_{\tau\sim\pi}\Big[R(\tau)\Big] \;=\; \mathbb{E}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t\, r(\tau_t)\Big]\,,$$

where τ is a trajectory, $\tau_t = (s_0, a_0, \ldots, s_t, a_t)$ is its prefix up to time t, and $0 < \gamma < 1$ is a discount factor.

However, in our framework, the reward function R is not explicitly defined. Instead, the agent relies on automaton-based preferences over trajectory segments, which are used to guide learning. This setup is particularly effective for handling environments with non-Markovian rewards, where task objectives depend on the entire sequence of states and actions rather than just the current state.

Definition 3.2 (Deterministic Finite Automaton (DFA))**.** A DFA is a finite-state machine defined by the tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ is the alphabet of the input language, Q is a finite set of automaton states with starting state q_0, $\delta : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

The DFA defines a high-level structure for evaluating trajectories based on the sequence of state-action pairs. It provides a formalized representation of the task objectives, enabling trajectory evaluation by observing transitions through the DFA states. The sequence $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ results in corresponding DFA transitions, allowing the agent to track progress and alignment with task goals.

Definition 3.3 (Product MDP)**.** The Product MDP is represented by $\mathcal{M}_{\text{prod}} = (S \times Q, A, T_{\text{prod}}, (s_0, q_0), R_{\text{prod}})$, where S is the state space of the NMRDP, Q is the DFA state space, A is the set of available actions, $T_{\text{prod}} : (S \times Q) \times A \to \Delta(S \times Q)$ is the transition function, and $R_{\text{prod}} : Q \times \Sigma \to \mathbb{R}$ is a reward function, where Σ is the alphabet of the input language of the DFA.

The Product MDP integrates the NMRDP and DFA, allowing the agent to act while adhering to task-specific temporal constraints. The transition function T_{prod} combines environment dynamics with DFA transitions, reflecting subgoal progress. The DFA state $q \in Q$ tracks task progression, while the NMRDP state $s \in \mathcal{S}$ represents the environment's physical state. The reward function R_{prod} either reflects subgoal achievements in a known reward setup or is derived through preference learning. Embedding DFA logic into the environment's dynamics enables standard RL methods to handle non-Markovian tasks. Our learning approaches leverage the Product MDP to enforce temporal objectives while learning preference-based rewards.

4 METHODOLOGY

In this section, we describe our automaton-based RL framework, focusing on its core components: (i) Preference elicitation from a DFA, (ii) Reward function learning, and (iii) Policy optimization. We also introduce our static and dynamic learning variants. The pseudocode for the full algorithm is provided in Appendix B. Figure 2 illustrates our framework.

4.1 PREFERENCE ELICITATION

Our method generates preferences over pairs of trajectories (τ_1, τ_2) by evaluating their adherence to task specifications using the DFA. Unlike RLHF, where preferences are derived from human input, our approach leverages the DFA to assess trajectories without direct access to rewards. To rank trajectories, we define a composite score function based on the task structure encoded in the DFA. We present two distinct approaches for generating these preference scores.

Subtask-based scoring. Here, trajectories are scored based on two factors: (i) *subgoal completion*, which is the number N_s of subgoals completed in the correct order as tracked by the DFA, and (ii) *distance to the next subgoal or goal*, which is a distance measure d from the last state of the trajectory to the next required subgoal or goal. Formally, the score for a trajectory τ is computed as:

$$\text{score}(\tau) = w_s \cdot N_s(\tau) - w_d \cdot d(\tau), \tag{1}$$

where w_s and w_d are weighting factors, with $w_s \gg w_d$, prioritizing subgoal completion over distance.

Given two trajectories τ_1 and τ_2, their scores are compared to establish a preference ($\tau_1 \succ \tau_2$ or vice-versa). These preferences are then used to learn a reward function (see Section 4.2).

Remark 4.1. *The DFA plays a central role in scoring by tracking subgoal completion and distance calculation. In particular, the DFA maps the trajectory to a sequence of states, recording transitions as the agent completes subgoals in order, providing $N_s(\tau)$, and is used to determine the next required subgoal, enabling the computation of $d(\tau)$.*

Remark 4.2. *We initially set d as the Manhattan distance for discrete gridworlds, however, our framework is adaptable to various environments. For continuous state spaces, Euclidean distance is a more suitable metric.*

For more complex domains with non-trivial dynamics, learned cost functions or domain-specific metrics can be integrated. For instance, in robotic manipulation tasks, a geodesic distance in configuration space might be more appropriate than direct Euclidean distance.

Q-value-based scoring. In this approach, we integrate *automaton transition values* into the trajectory scoring function. Assume that for each transition (q, σ) in the DFA, we have access to Q-value estimates $\overline{Q}_{\text{dfa}}(q, \sigma)$, representing the desirability of the automaton transition. These values can either be derived directly during training or obtained via a transfer learning process, as detailed later.

Given a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$, the corresponding trace in the DFA is $(q_0, \sigma_0, q_1, \sigma_1, \ldots, q_T)$, where q_0 is the initial automaton state, $q_{t+1} = \delta(q_t, \sigma_t)$, for $t = 0, \ldots, T-1$, and $\sigma_t = L(s_{t+1})$, recalling that δ is the DFA transition function and L is the labeling function which maps a given state to the set of atomic propositions that hold in said state.

The Q-value-based trajectory score is computed as:

$$\text{score}(\tau) = \sum_{t=0}^{T-1} \overline{Q}_{\text{dfa}}\big(q_t, \sigma_t\big)\big). \tag{2}$$

Each automaton transition (q_t, σ_t) represents a partial condition being satisfied in the environment. The value $\overline{Q}_{\text{dfa}}(q_t, \sigma_t)$ reflects how desirable this transition is, based on previously learned or transferred knowledge. Therefore, by summing over all transitions, the scoring function in equation 2 captures the cumulative desirability of the trajectory with respect to the automaton's objectives.

Remark 4.3 (Theoretical Foundation for Q-Value Aggregation)**.** *The aggregation in Equation equation 2 is theoretically grounded in the Markovian nature of the product MDP. When we combine environment state s with automaton state q, the resulting product state (s, q) renders the process Markovian. Consequently, if we can approximate Q-values as functions of this combined*

state—specifically $Q((s,q),a)$ in the product MDP—then trajectory quality can be determined from terminal product states. Our aggregated score $\sum_t \overline{Q}_{dfa}(q_t,\sigma_t)$ approximates this product MDP value function by summing the expected values of logical transitions, effectively capturing the cumulative desirability of the symbolic path traced by the trajectory within the Markovian product space.

As before, preferences between two trajectories, τ_1 and τ_2, are determined by comparing their scores in equation 2. Specifically, $\tau_1 \succ \tau_2$ if $\text{score}(\tau_1) > \text{score}(\tau_2)$, and vice versa. In the case of a tie, they are considered equally preferable. These trajectory-level preferences are then used to train a parametric reward model (see Section 4.2).

But how can we obtain automaton Q-value estimates $\overline{Q}_{dfa}(q,\sigma)$ if these values are not readily available?

Obtaining automaton Q-values: Knowledge distillation. Automaton Q-values can be distilled from a simpler teacher environment in which the agent shares the same task objective. We employ a teacher-student transfer learning framework, where the knowledge encoded in automaton transitions is transferred from the teacher to the student.

Consider an agent trained in a simpler teacher environment using standard RL methods to optimize its behavior according to the task's temporal logic. During training, experiences $\mathcal{D}$ are stored in the form of samples $((s,q),a,r,(s',q'))$, where s and s' are environment states, q and q' are automaton states, and a is the action taken. To distill knowledge, the frequency of automaton transitions is tracked, defined for automaton state q and the atomic propositions σ that label transitions:

$$n_{\text{teacher}}(q,\sigma) = \left|\left\{((s,q),a,r,(s',q')) \in \mathcal{D} \;\middle|\; L(s') = \sigma\right\}\right|.$$

The average Q-value for each automaton transition (q,σ), where $\sigma = L(s')$, is then computed as:

$$\overline{Q}_{\text{teacher}}(q,\sigma) = \frac{\sum_{((s,q),a,r,(s',q'))\in\mathcal{D}, L(s')=\sigma} Q_{\text{teacher}}((s,q),a)}{n_{\text{teacher}}(q,\sigma)}, \tag{3}$$

where $Q_{\text{teacher}}((s,q),a)$ is the Q-value for the state-action pair $((s,q),a)$ in the teacher environment.

Remark 4.4. *We could also use a combined trajectory scoring function that integrates the subtask-based and Q-value-based approaches defined in equation 1 and equation 2, respectively. The combined score function is defined as:*

$$\textit{score}(\tau) = w_s N_s(\tau) - w_d d(\tau) + w_q \sum_{t=0}^{T-1} \overline{Q}_{\textit{teacher}}\big(q_t,\sigma_t\big), \tag{4}$$

where w_s, w_d, and w_q are weighting factors for subgoal completion, distance, and automaton Q-values, respectively, which can be tuned to prioritize different aspects of the task.

4.2 Learning the Reward Function

Using the preferences derived from the DFA, next we aim to learn a reward function $\hat{r}_\theta((s,q),a)$, parameterized by θ, that captures the task's objectives. This reward function is trained via a pairwise ranking loss, ensuring that trajectories preferred by the DFA receive higher cumulative rewards.

Pairwise ranking loss. We learn the reward function $\hat{r}_\theta$ by minimizing the pairwise ranking loss:

$$L(\theta) = \sum_{(\tau_p,\tau_n)} \max\Big(0, m - (\hat{R}_\theta(\tau_p) - \hat{R}_\theta(\tau_n))\Big), \tag{5}$$

where the sum is over pairs of preferred and non-preferred trajectories, τ_p and τ_n, respectively, and m is a margin hyperparameter, ensuring a sufficient difference between the cumulative rewards of these trajectories. By minimizing this loss, we ensure that the learned reward function aligns with the DFA-derived preferences, thereby embedding the task's temporal structure into the reward model. For each pair of trajectories (τ_p,τ_n), the cumulative discounted reward is computed as:

$$\hat{R}_\theta(\tau) = \sum_{t=0}^{T-1} \gamma^t \hat{r}_\theta((s_t,q_t),a_t), \tag{6}$$

where $\hat{r}_\theta((s,q),a)$ is the learned reward function and T is the length of the trajectory.

4.3 Policy Optimization: Static and Dynamic Variants

After learning the reward function $\hat{r}_\theta$, the policy π is optimized using standard RL techniques to maximize the expected cumulative reward. In our experiments, we employ Q-learning for discrete environments and Twin Delayed DDPG (TD3) Fujimoto et al. (2018) for continuous domains.

We propose two learning variants. Specifically, in the **static version**, the reward function is learned once from an initial set of preferences. The policy is then optimized using this fixed reward function. This approach is computationally efficient and well-suited for tasks where the reward structure remains constant over time. In the **dynamic version**, the reward function and policy are refined iteratively. At each iteration, trajectories from the current policy are evaluated by the DFA to produce preferences, which update the reward function. The policy is then re-optimized using the updated reward. This loop continues until convergence and suits tasks needing ongoing refinement due to evolving dynamics or incomplete initial preferences.

We now present our theoretical result informally. See Appendix A for a formal statement and proof.

Theorem 4.5 (Convergence and ε-Optimality (Informal))**.** *Let $\mathcal{M}$ be a finite environment with a non-Markovian reward function R^* encoded via a DFA. If preferences over trajectories align with R^* and Q-learning on the product MDP $\mathcal{M}_{\text{prod}}$ ensures sufficient exploration, then with high probability the learned policy $\hat{\pi}$ satisfies $V_{R^*}(\hat{\pi}) \geq \max_\pi V_{R^*}(\pi) - \varepsilon$.*

5 Experimental Results

We evaluate our framework using six methods: static, dynamic, known reward function, distillation with reward shaping, reward machine, and Logic-Guided Policy Optimization (LPOPL). The goal is to optimize policies in gridworlds with subgoals, obstacles, and a final goal, analyzing performance on tasks with sequential objectives across both discrete and continuous state spaces.

For the *known reward function* method, the agent has access to a hand-crafted reward and directly optimizes its policy, serving as a performance benchmark. Inspired by structured task decomposition Ng & Russell (1999); Singh et al. (2010); Konidaris et al. (2009), the reward encourages completing subgoals in order: $+3$ for correct subgoals, -1 for out-of-order ones, $+9$ for reaching the final goal if all subgoals were completed in order (-3 otherwise), and -0.1 per step to promote efficiency. This design reflects hierarchical RL ideas that leverage prior task structure Dietterich (2000).

We introduce a method termed *distillation with reward shaping*, adapted from automaton-based RL Icarte & et al. (2018; 2021); Xu et al. (2020); Singireddy et al. (2023). It neither relies on preference comparisons nor manual rewards. It uses the given DFA for minimal numeric shaping: $+1$ for successful subgoal transitions and -0.1 otherwise. The agent refines its policy solely from DFA signals. The *reward machine* baseline implements the approach from Icarte et al. (2022), where the automaton directly defines rewards based on state transitions. Each valid transition in the automaton that represents progression toward the goal receives a positive reward ($+1$), while invalid transitions receive no reward. Additionally, a small step penalty (-0.1) encourages efficient solutions. This approach directly translates the automaton structure into a reward function without learning from preferences. The *LPOPL* baseline Hasanbeig et al. (2018) shapes rewards using LTL specifications converted into automata, assigning rewards based on progress toward satisfying the logic formula. In contrast to our preference-based approach, LPOPL directly encodes logical progression into the reward function (see Appendices D, E, and F for further details about these baselines).

Experimental setup. We evaluate our approach in six environments spanning diverse domain types to demonstrate broad applicability and scalability. Four gridworld environments (Minecraft Iron Sword Quest, Dungeon Quest, Blind Craftsman, and Minecraft Building Bridge) are visualized and described in Appendix C along with their DFAs. The first involves a single sequence of subgoals, while the others introduce complexity through multiple valid paths and loops. To validate broader applicability, we extend evaluation to two additional environments representing fundamentally different domain types: Mountain Car Collection, a physics-based terrain navigation task with energy management constraints, and Warehouse Robotics, implementing a realistic 12-dimensional continuous state space that directly addresses scalability concerns for robotic applications. All six tasks require completing subgoals in specific temporal orders, defined by DFAs that encode non-Markovian objectives. These environments highlight the limitations of simple state-based rewards and demonstrate the advantages of DFA-based preferences across discrete navigation, physics-based control, and high-dimensional continuous domains.

Results. First, we report results using the subtask-based scoring in equation 1, evaluated across the four environments on three metrics: 1) reward per episode, 2) steps per episode, and 3) reward per cumulative steps, which best reflects learning efficiency. For brevity, we focus on the third metric in the main text; the others appear in Appendix G.

Figure 2 shows the **reward per cumulative steps**, which highlights overall learning efficiency. In two environments, the dynamic method achieves superior efficiency after sufficient training, balancing exploration with preference refinement. In the other two environments, both the static and dynamic methods maintain competitive performance, often outperforming the other baselines.

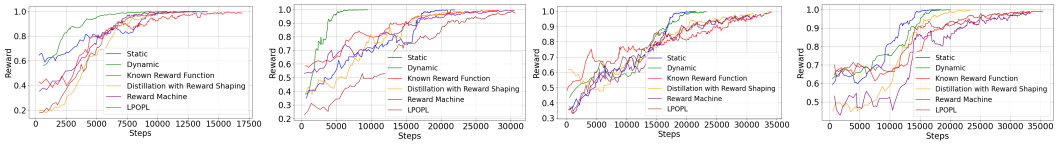

Figure 2: Reward per cumulative steps for all environments. From left to right: Minecraft Iron Sword Quest, Dungeon Quest, Blind Craftsman, and Minecraft Building Bridge.

The dynamic method excels through iterative reward refinement, while the static method performs well with a fixed reward. Both surpass the known reward, reward-shaping, RM, and LPOPL baselines, underscoring the advantage of adaptive, automaton-based alignment in subgoal-driven RL tasks.

Continuous environment results.

To assess generalizability, we tested our approach in continuous versions of the four environments, where agents navigate 2D planes with real-valued coordinates. We replaced Manhattan distance with Euclidean distance and used TD3 for policy optimization.

Figure 3 shows reward per cumulative steps, demonstrating that our approach generalizes well to continuous state spaces. The dynamic method outperforms others as training progresses, while both preference-based methods exceed the known reward and DFA shaping baselines. This highlights the value of preference-based learning in continuous domains, where reward engineering is harder and nuanced feedback is essential. The minimal changes needed (updating distance metrics and policy optimization methods) underscore the flexibility and robustness of our automaton-based preference framework, supporting its scalability to more realistic settings.

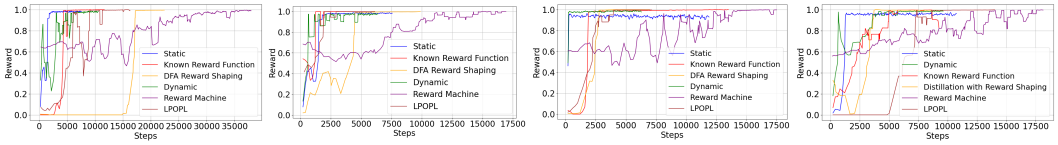

Figure 3: Reward per cumulative steps for all environments in continuous state spaces.

Extended domain validation. Figure 4 shows results for our two additional environments, demonstrating effectiveness across diverse domain types. Both static and dynamic methods outperform all baselines, confirming scalability to realistic applications beyond discrete gridworlds.

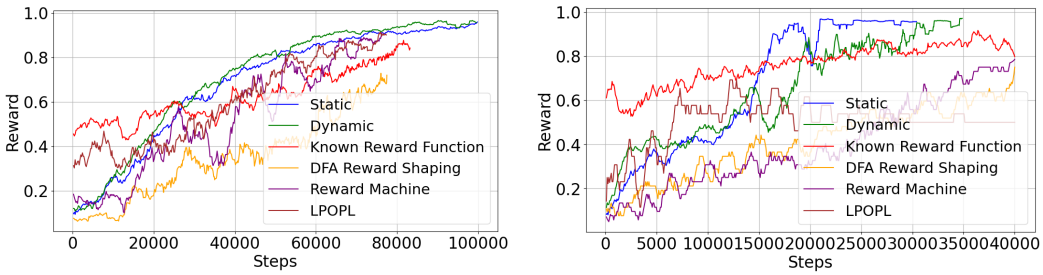

Figure 4: Reward per cumulative steps for extended domain validation. Mountain Car Collection (left) and Warehouse Robotics 12D environment (right).

Combined scoring with transfer. We evaluate the effectiveness of combining subtask-based preferences with Q-value-based knowledge transferred from a teacher agent using the scoring function equation 4. By integrating the teacher's automaton Q-values into preference scoring, we enhance a

student agent's learning in a more complex environment, enabling the student agent to perform well in challenging scenarios where environment-provided rewards are unavailable. The DFA is augmented with Q-values distilled from a teacher agent trained in the original environments (see equation 3). We then leverage these learned DFA Q-values to student variants of the four environments, scaled up to larger grids (10×10, 15×15, 12×12, 20×20) and featuring additional obstacles or modified subgoal placements as shown in Appendix C. Although they retain the same subgoal structure, these expanded layouts pose a greater navigational challenge and demand more complex policy adaptation.

To evaluate our approach, we compare the performance of several student agents. In **Static (Pref)** and **Dynamic (Pref)**, the student leverages distilled teacher preferences under the static and dynamic methods, respectively. We also introduce **Static (Pref+Plan)** and **Dynamic (Pref+Plan)**, where the DFA's Q-values guide both preference scoring and planning. Specifically, the student's Q-learning update rule is modified by an annealing mechanism:

$$Q'_{\text{student}}((s,q),a) = \beta(q, L(s'))\,\overline{Q}_{\text{teacher}}(q, L(s')) + \big(1 - \beta(q, L(s'))\big)\,Q_{\text{target}}, \tag{7}$$

where Q'_{student} is the adjusted Q-value in the product MDP. The weight, $\beta(q, L(s')) = \rho^{\,n_{\text{student}}(q, L(s'))}$, decays over time (with $\rho \in (0,1)$), controlling the degree to which the student relies on the teacher's knowledge, where $n_{\text{student}}(q,\sigma)$ is the frequency of automaton transition (q,σ) in the student environment. The standard target Q-value is $Q_{\text{target}} = r + \gamma \max_{a'} Q_{\text{student}}((s',q'),a')$. By relying more on the teacher's Q-values early in training and gradually shifting to its own experience, the student smoothly transitions from teacher-guided to autonomous learning. Here, r refers to the output of the learned reward model (trained on DFA-derived preferences) rather than environment rewards.

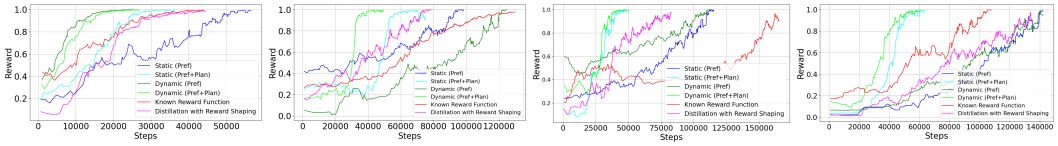

Figure 5: Reward per step for different methods and environments.

Figure 5 shows that distillation agents consistently outperform non-distilled ones in reward per cumulative steps, highlighting the efficiency gained from teacher-derived automaton knowledge. The distillation process transfers subgoal and automaton transition expertise to the student, guiding it via distilled Q-values in both preference generation and policy optimization. This leads to faster convergence by aligning trajectories with the task's temporal structure.

6 Discussion and Limitations

While our automaton-based preference framework performs well across tasks, there are some limitations. Constructing DFAs for large-scale environments can be difficult, motivating future work on learning or refining automata from demonstrations or natural language. Our preference generation uses simple heuristics like subgoal completion and distance, which may not suffice in complex domains with non-Euclidean spaces or interleaved goals; richer models offer a natural extension. Although we generalize to continuous domains, scaling to high-dimensional spaces poses challenges for both DFA representation and preference learning. Appendix I outlines theoretical tools (including automaton-guided attention and transfer learning) with formal guarantees, though empirical validation is pending. Our current framework also assumes full observability; extending to partially observable settings will require belief tracking or state estimation as discussed in Appendix H.

7 Conclusion

We introduced an RL framework that achieves automaton-based alignment, using DFAs for preference rankings to train predictive reward models. We explored static and dynamic variants, benchmarking them against policies using a designed reward function, distillation with reward shaping, reward machines, and LTL-based approaches. Our method demonstrates superior performance across both discrete and continuous environments, highlighting the advantage of preference-based learning over direct reward specification. Our approach captures task-specific temporal dependencies, translating complex sequential objectives into structured, preference-based rewards. The results highlight the advantage of automata-derived reward shaping, particularly in tasks with multiple subgoals and evolving policy requirements. The success of both dynamic and static methods over the baselines underscores the value of integrating structured preferences for more adaptive and efficient RL.

REFERENCES

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 1, 2004.

Mohamad Alshiekh and et al. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2669–2678, 2018.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 166–175, 2017a.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 166–175, 2017b.

Marcin Andrychowicz and et al. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

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

Fahiem Bacchus and Sheila A. K. Renold. Learning continuous time markov processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 703–710, 1996.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1726–1734, 2017.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*, 2002.

André Barreto, Will Dabney, Rémi 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*, pp. 4055–4065, 2017.

Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

Erdem Biyik, Mehmet Palan, Nicholas C. Landolfi, Dylan P. Losey, and Dorsa Sadigh. Active preference-based learning of reward functions. *Proceedings of Robotics: Science and Systems*, 2020.

Udi Boker, Katriel Cohn-Gordon, Denis Kuperberg, Tony Tan, and Moshe Vardi. Translation of ltl to limit deterministic automata. In *Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis*, pp. 57–73, 2022.

Alberto Camacho and et al. Ltlmop: Learning and reasoning about temporal tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Zichuan Chen, Qing Yang, Kai Li, and Zhe Xu. Nl2tl: Transforming natural languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766*, 2023. URL `https://arxiv.org/pdf/2305.07766`.

Paul Christiano and et al. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30:4299–4307, 2017.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Pierre Dupont and et al. Incremental learning of dfa using multiple examples. In *Proceedings of the European Conference on Machine Learning*, 1996.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1126–1135, 2017.

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*, pp. 1587–1596, 2018.

Adam Gleave, Michael D. Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Uncertainty estimation for language reward models. In *Advances in Neural Information Processing Systems*, 2022.

Ernst Moritz Hahn and et al. Omega-regular objectives in model-free reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2019.

Stanley Hart and Richard Grupen. Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development*, 1(1):1–16, 2009.

Mohammad Hasanbeig and et al. Deep reinforcement learning with formal guarantees for safety-critical control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-guided reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 10102–10110, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

Rodrigo Toro Icarte and et al. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

Rodrigo Toro Icarte and et al. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 70:1307–1357, 2021.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, pp. 13021–13030, 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In *Advances in Neural Information Processing Systems*, 2021.

Leslie Pack Kaelbling. Learning to achieve goals. *International Joint Conference on Artificial Intelligence*, 2:1094–1099, 1993.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Alex Kendall and et al. Learning to drive in a day. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254, 2019.

George Konidaris, Scott Kuindersma, Andrew Barto, and Roderic Grupen. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1015–1023, 2009.

Aviral Kumar, Xue Bin Peng, and Sergey Levine. Should i reset? learning when and how to reset for safe reinforcement learning. *arXiv preprint arXiv:2211.03573*, 2022.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

Sergey Levine and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Jing Li and et al. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

Xiao Li and Calin Belta. Temporal logic guided safe reinforcement learning using control barrier functions. In *Proceedings of the 56th IEEE Conference on Decision and Control*, pp. 3110–3115, 2017.

Timothy P. Lillicrap and et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

Michael L. Littman and et al. Environment-independent task specifications via gltl. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

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

Daniel Neider, Jean-Raphael Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. Advice-guided reinforcement learning in a non-markovian environment. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*. AAAI, 2021.

Andrew Y. Ng and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, pp. 278–287, 1999.

Josep Oncina and Pedro García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pp. 99–108, 1992.

Jaekyeom Park, Jihwan Chun, Soyoung Bae, Kyungmin Kim, and Kee-Eung Yoo. Surf: Self-supervised reward function learning with constractive preference comparisons. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 17506–17525, 2022.

Ran Qiu, Yanran Wang, Chao Yu, Jiaming Ji, Haifeng Zhang, and Jun Wang. Instructing reinforcement learning agents via temporal logic specifications. In *Proceedings of the International Conference on Machine Learning*, 2023.

Rafael Rafailov and et al. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Andrei A. Rusu and et al. Policy distillation. In *arXiv preprint arXiv:1511.06295*, 2015.

Narun Shah, Timothy Davison, and David Colturato. Llm2ltl: Extracting, correcting, and verifying temporal logic specifications for reinforcement learning using large language models. *arXiv preprint arXiv:2311.00210*, 2023.

David Silver and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

Satinder Singh, Richard L. Lewis, Andrew G. Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

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. *arXiv preprint arXiv:2310.19137*, 2023.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 1998.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

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

Oriol Vinyals and et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

Cameron Voloshin, Abhinav Verma, Hoang Le, Yisong Yue, Swarat Chaudhuri, and Dorsa Sadigh. Policy optimization for specifications with continuous monitoring. *arXiv preprint arXiv:2207.05185*, 2022.

Cameron Voloshin, Hoang Le, Yisong Yue, Swarat Chaudhuri, and Dorsa Sadigh. Eventual discounting temporal logic counterfactual experience replays. *arXiv preprint arXiv:2304.05692*, 2023.

Neil Walkinshaw, Jose A. Taylor, and Rajesh S. Raja. Inferring finite-state models with temporal constraints. *Empirical Software Engineering*, 21(2):434–467, 2016. doi: 10.1007/s10664-015-9375-2.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565.

Daan Wierstra and et al. Solving deep memory POMDPs with recurrent policy gradients. *Proceedings of the International Conference on Artificial Neural Networks*, pp. 697–706, 2007.

Xuanlin Xu, Xinyun Chen, Piotr Mirowski, Gabriel Synnaeve, Nicolas Usunier, Arthur Guez, and Edward Hughes. Joint reinforcement learning of global and local policies with reward machines. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 10692–10703, 2020.

Zihan Xu and et al. Learning reward machines for partially observable reinforcement learning. In *Proceedings of the Advances in Neural Information Processing Systems*, 2020.

Mert Yalcinkaya and Zhe Xu. Compositional automata embeddings for goal-conditioned reinforcement learning. *arXiv preprint arXiv:2401.13242*, 2024.

Yunhao Yang, Neel P. Bhatt, Tyler Ingebrand, William Ward, Steven Carr, Zhangyang Wang, and Ufuk Topcu. Fine-tuning language models using formal methods feedback, 2023. Available at `https://arxiv.org/abs/2310.18239`.

Banghua Zhu, Michael Jordan, and Jiantao Jiao. Principled reinforcement learning with human feedback from pairwise or k-wise comparisons. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 43037–43067. PMLR, 2023.

A Convergence Theorem for Automaton-Guided Preference-Based RL

In this section, we provide a formal statement and proof of the convergence theorem for our DFA-guided preference-based RL framework, establishing that, under some assumptions, pairwise preference learning guided by a DFA allows an agent to learn a reward function whose optimal policy is close to that of the (unknown) ground-truth objective.

Setup: Consider a finite non-Markovian environment $\mathcal{M} = (S, s_0, A, T, R^*)$, where S is a finite set of states, A is a finite set of actions, $T : S \times A \to \Delta(S)$ is a transition function, and $R^*(\tau)$ is a *trajectory-return* encoding a non-Markovian objective. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a finite DFA that captures the temporal constraints or subgoal structure. We construct the *product MDP* $\mathcal{M}_{\text{prod}}$ with state space $\widetilde{S} = S \times Q$, actions A, and transitions combining T with δ. Since $\widetilde{S}$ and A are finite, $\mathcal{M}_{\text{prod}}$ is a finite MDP.

Theorem A.1 (Convergence and ε-Optimality in Finite Product MDP)**.** *Let Π be the set of all stationary policies on $\mathcal{M}_{\text{prod}}$. Assume:*

- ***(1) Preference Consistency.*** *There exists a real-valued function $R^*(\tau)$ such that if $\tau_1 \succ \tau_2$ (according to DFA-based preferences), then*
$$R^*(\tau_1) \;-\; R^*(\tau_2) \;\geq\; \delta_0 \;>\; 0.$$
- ***(2) Sufficient Expressivity.*** *We can learn a reward $\hat{r}_\theta : \widetilde{S} \times A \to \mathbb{R}$ such that its induced return $\hat{R}_\theta(\tau) = \sum_{t=0}^{T-1} \gamma^t \, \hat{r}_\theta(\widetilde{s}_t, a_t)$ can represent the same preference ordering as R^* on the relevant set of trajectories.*
- ***(3) Correct Preference Training.*** *By sampling a sufficiently large set of labeled trajectory pairs $\{(\tau_p, \tau_n)\}$ from a coverage distribution, we minimize a pairwise ranking loss so that, with probability at least $1 - \zeta$,*
$$\left|\hat{R}_\theta(\tau) \;-\; R^*(\tau)\right| \;\leq\; \varepsilon_r \quad \textit{for most trajectories } \tau \textit{ under that distribution.}$$
(This ensures the ordering $\tau_1 \succ \tau_2$ is preserved except on a small fraction of pairs.)
- ***(4) Persistent Exploration (Tabular).*** *In tabular Q-learning on $\mathcal{M}_{\text{prod}}$ using $\hat{r}_\theta$,* each state–action pair $(\widetilde{s}, a) \in \widetilde{S} \times A$ *is visited infinitely often.*

Let $\hat{\pi}$ be the greedy policy w.r.t. the Q-values learned from $\hat{r}_\theta$. Then, there is an $\varepsilon = \varepsilon_r + \varepsilon_t > 0$ such that, with probability at least $1 - \zeta$,

$$V_{R^*}(\hat{\pi}) \;\geq\; \max_{\pi \in \Pi} V_{R^*}(\pi) \;-\; \varepsilon,$$

i.e. $\hat{\pi}$ is ε-optimal w.r.t. the true non-Markovian objective R^*.

Proof. **Step 1: Consistency of the Learned Return.**

By assumption **(3)**, the pairwise ranking loss is minimized on a sufficiently large set of trajectory pairs $\{(\tau_p, \tau_n)\}$ consistent with $R^*(\tau)$. Hence, with probability at least $1 - \zeta$, we obtain a function $\hat{r}_\theta$ such that its cumulative $\hat{R}_\theta(\tau)$ satisfies:

$$\tau_1 \succ \tau_2 \quad \Longrightarrow \quad \hat{R}_\theta(\tau_1) \;>\; \hat{R}_\theta(\tau_2),$$

except possibly on a small fraction of pairs. Moreover, $\left|\hat{R}_\theta(\tau) - R^*(\tau)\right| \leq \varepsilon_r$ for most τ under the sampling distribution. The margin $\delta_0 > 0$ in **(1)** ensures that small errors in $\hat{R}_\theta(\tau)$ do not flip the preference unless those errors exceed δ_0.

Step 2: Q-Learning Convergence in the Finite Product MDP.

Define the reward in M_{prod} as $\hat{r}_\theta(\widetilde{s}, a)$ for $\widetilde{s} \in \widetilde{S}$, $a \in A$. Since $\widetilde{S}$ and A are finite, tabular Q-learning applies. By assumption **(4)**, *each* pair $(\widetilde{s}, a)$ is visited infinitely often. According to Watkins' theorem

Watkins & Dayan (1992), the Q-values converge almost surely to the unique fixed point $Q^*_{\hat{r}_\theta}$ satisfying the Bellman optimality equation for $\hat{r}_\theta$. Let $\hat{\pi}$ be the greedy policy w.r.t. $Q^*_{\hat{r}_\theta}$; then $\hat{\pi}$ maximizes

$$V_{\hat{R}_\theta}(\pi) \;=\; \mathbb{E}_{\tau\sim\pi}\Big[\hat{R}_\theta(\tau)\Big].$$

Step 3: Relating $\hat{R}_\theta$-Optimality to R^*-Optimality.

Let $\pi^* \in \Pi$ be any policy (potentially optimal) w.r.t. R^*. We show:

$$V_{R^*}(\hat{\pi}) \;\geq\; V_{R^*}(\pi^*) \;-\; (\varepsilon_r + \varepsilon_t).$$

Since $\hat{\pi}$ is optimal for $\hat{R}_\theta$, we have

$$V_{\hat{R}_\theta}(\hat{\pi}) \;=\; \max_{\pi\in\Pi} V_{\hat{R}_\theta}(\pi).$$

On the trajectories visited by $\hat{\pi}$, the difference $\big|\hat{R}_\theta(\tau) - R^*(\tau)\big|$ is at most ε_r (with high probability), so

$$V_{R^*}(\hat{\pi}) \;\geq\; V_{\hat{R}_\theta}(\hat{\pi}) \;-\; \varepsilon_t,$$

where ε_t captures any residual misalignment or low-probability outliers. Next,

$$V_{\hat{R}_\theta}(\hat{\pi}) \;=\; \max_{\pi\in\Pi} V_{\hat{R}_\theta}(\pi) \;\geq\; V_{\hat{R}_\theta}(\pi^*).$$

Again relating $\hat{R}_\theta$ to R^* on π^*'s trajectories,

$$V_{\hat{R}_\theta}(\pi^*) \;\geq\; V_{R^*}(\pi^*) \;-\; \varepsilon_r.$$

Combining,

$$V_{R^*}(\hat{\pi}) \;\geq\; \big[V_{\hat{R}_\theta}(\hat{\pi}) - \varepsilon_t\big] \;\geq\; \big[V_{\hat{R}_\theta}(\pi^*) - \varepsilon_t\big] \;\geq\; \big[V_{R^*}(\pi^*) - (\varepsilon_t + \varepsilon_r)\big].$$

Hence,

$$V_{R^*}(\hat{\pi}) \;\geq\; V_{R^*}(\pi^*) \;-\; (\varepsilon_t + \varepsilon_r).$$

Defining $\varepsilon := \varepsilon_r + \varepsilon_t$ yields the claimed ε-suboptimality bound for $\hat{\pi}$ with high probability $(1 - \zeta)$. Since π^* was arbitrary, in particular

$$V_{R^*}(\hat{\pi}) \;\geq\; \max_{\pi\in\Pi} V_{R^*}(\pi) \;-\; \varepsilon.$$

This completes the proof. □

B ALGORITHMS

In this appendix, we provide the detailed algorithms for our preference-based RL approach. Algorithm 1 summarizes the main procedure, and Algorithm 2 describes the preference computation based on the DFA.

Algorithm 1: RL with Automaton-Based Preferences and Pairwise Ranking Loss

Input: MDP (S, A, P, γ), DFA $\mathcal{A}$, α, β, m, Maximum iterations K (Dynamic) or episodes E (Static)
Output: Optimal policy π^*
Phase 1: Preference Generation & Reward Learning if *Static mode* then
  Initialize random policy π_{rand}
  Generate trajectories $\{\tau_i\}$ via π_{rand}
  Assign preferences $P(\tau_i, \tau_j)$ for sampled trajectory pairs using DFA $\mathcal{A}$ `(Algorithm 2)`
  Train reward model $\hat{r}_\theta$ via pairwise ranking loss
else if *Dynamic mode* then
  Initialize policy π **for** $k = 1$ *to* K **do**
    Collect trajectories $\{\tau_i\}$ via current π
    Assign preferences $P(\tau_i, \tau_j)$ for trajectory pairs using DFA $\mathcal{A}$
    Update reward model $\hat{r}_\theta$ with new preferences
    Optimize policy π via Q-learning on $\hat{r}_\theta$
    if *policy performance is stable* **then**
      break
Phase 2: Policy Optimization (Static Mode Only) if *Static mode* then
  for $e = 1$ *to* E **do**
    Optimize policy π via Q-learning using $\hat{r}_\theta$
return final policy π^*

Algorithm 2: Preference Computation Based on Subtask Completion and Distance

Input: Trajectory pairs $\{(\tau_i, \tau_j)\}$, DFA $\mathcal{A}$, Subgoals $\mathcal{G} = [g_1, g_2, \ldots, g_N]$, Goal state s_{goal}, Weights w_s, w_d
Output: Preferences $P(\tau_i, \tau_j)$ for each trajectory pair
for *each trajectory pair* (τ_i, τ_j) **do**
  for *each trajectory* $\tau \in \{\tau_i, \tau_j\}$ **do**
    Initialize DFA $\mathcal{A}$ to its initial state;
    $N_s(\tau) \leftarrow$ Number of subtasks completed in order by simulating $\mathcal{A}$ with τ;
    $s_T \leftarrow$ Last state of trajectory τ;
    if $N_s(\tau) < N$ **then**
      $s_{\text{next}} \leftarrow$ Next subgoal $g_{N_s(\tau)+1}$;
    else
      $s_{\text{next}} \leftarrow s_{\text{goal}}$;
    $d(\tau) \leftarrow \text{ComputeDistance}(s_T, s_{\text{next}})$;
    $\text{score}(\tau) \leftarrow w_s \cdot N_s(\tau) - w_d \cdot d(\tau)$;
  Assign $P(\tau_i, \tau_j)$ to the trajectory with the higher score, or mark as indifferent if scores are equal;
return $P(\tau_i, \tau_j)$ for all trajectory pairs;

We further provide detailed implementations of our key methodological variants. Algorithm 3 presents our discrete Q-learning approach for grid-based environments, showing both static and dynamic variants with subtask-based scoring. Algorithm 4 extends our framework to continuous domains using Twin Delayed DDPG (TD3), adapting preference generation for continuous state-action spaces with Euclidean distance metrics. Finally, Algorithm 5 illustrates our knowledge distillation approach, which leverages teacher-learned automaton Q-values to guide both preference generation and policy planning in student environments, enabling efficient transfer learning across tasks with shared automaton structures.

Algorithm 3: Discrete Q-Learning with Automaton-Based Preferences (Static and Dynamic)

Input: Grid environment (S, A, P), DFA $\mathcal{A} = (Q, q_0, \delta, F)$, α, γ, ϵ_0, $\epsilon_{\min}$, ϵ_{decay}, Max episodes E, Mode $\in \{\text{Static}, \text{Dynamic}\}$, Weights w_s, w_d

Output: Optimal Q-table and policy π^*

Initialize reward model $\hat{r}_\theta$ with parameters θ

Initialize $Q[(s, q), a] \leftarrow 0$ for all $(s, q) \in S \times Q, a \in A$ $\epsilon \leftarrow \epsilon_0$

if *Mode = Static* **then**

- # Phase 1: Generate preferences and train reward model once
- Initialize random agent and generate trajectory pairs $\{(\tau_i, \tau_j)\}$
- **for** *each trajectory pair* (τ_i, τ_j) **do**
  - $N_{s1} \leftarrow$ Number of subgoals completed in τ_i by DFA $\mathcal{A}$
  - $N_{s2} \leftarrow$ Number of subgoals completed in τ_j by DFA $\mathcal{A}$
  - $s_{T1} \leftarrow$ Last state of τ_i; $s_{T2} \leftarrow$ Last state of τ_j
  - $s_{\text{next1}} \leftarrow$ Next required subgoal or goal for τ_i
  - $s_{\text{next2}} \leftarrow$ Next required subgoal or goal for τ_j
  - $d_1 \leftarrow \text{ManhattanDistance}(s_{T1}, s_{\text{next1}})$
  - $d_2 \leftarrow \text{ManhattanDistance}(s_{T2}, s_{\text{next2}})$
  - $\text{score}_1 \leftarrow w_s \cdot N_{s1} - w_d \cdot d_1$
  - $\text{score}_2 \leftarrow w_s \cdot N_{s2} - w_d \cdot d_2$
  - **if** *score*$_1 >$ *score*$_2$ **then**
    - Add preference $\tau_i \succ \tau_j$
  - **else if** *score*$_2 >$ *score*$_1$ **then**
    - Add preference $\tau_j \succ \tau_i$
- Train reward model $\hat{r}_\theta$ using pairwise ranking loss on preferences
- # Phase 2: Q-learning with fixed reward model
- **for** $e = 1$ *to* E **do**
  - $s \leftarrow$ initial state; $q \leftarrow q_0$
  - **while** *not terminal* **do**
    - probability $\epsilon a \leftarrow$ random action $a \leftarrow \arg\max_{a'} Q[(s, q), a']$ $s' \leftarrow$ next state after taking a
    - $q' \leftarrow \delta(q, L(s'))$; `/* DFA transition */`
    - $r \leftarrow \hat{r}_\theta((s, q), a)$; `/* Query reward model */`
    - $Q[(s, q), a] \leftarrow Q[(s, q), a] + \alpha(r + \gamma \max_{a'} Q[(s', q'), a'] - Q[(s, q), a])$
    - $s \leftarrow s'$; $q \leftarrow q'$
  - $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\min})$

else

- # Dynamic mode: iteratively update reward model and policy
- **for** *iterations* **do**
  - # Generate new trajectories with current policy
  - Generate trajectory pairs $\{(\tau_i, \tau_j)\}$ using ϵ-greedy policy from current Q-table
  - # Compute preferences using DFA
  - Compute preference scores as in Static mode
  - # Update reward model with new preferences
  - Train/update reward model $\hat{r}_\theta$ on new preferences
  - # Policy optimization phase with updated reward model
  - **for** $e = 1$ *to* E_{iter} **do**
    - $s \leftarrow$ initial state; $q \leftarrow q_0$
    - **while** *not terminal* **do**
      - probability $\epsilon a \leftarrow$ random action $a \leftarrow \arg\max_{a'} Q[(s, q), a']$ $s' \leftarrow$ next state after taking a
      - $q' \leftarrow \delta(q, L(s'))$
      - $r \leftarrow \hat{r}_\theta((s, q), a)$
      - $Q[(s, q), a] \leftarrow Q[(s, q), a] + \alpha(r + \gamma \max_{a'} Q[(s', q'), a'] - Q[(s, q), a])$
      - $s \leftarrow s'$; $q \leftarrow q'$
    - $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\min})$
  - **if** *policy performance is stable* **then**
    - **break**

return Q-table and derived policy $\pi(s, q) = \arg\max_a Q[(s, q), a]$

Algorithm 4: Continuous TD3 with Automaton-Based Preferences (Static and Dynamic)

Input: Continuous environment, DFA $\mathcal{A} = (Q, q_0, \delta, F)$, Learning parameters, Mode $\in \{\text{Static}, \text{Dynamic}\}$, Weights w_s, w_d
Output: Optimized policy π^*

```
# Initialize networks and buffers
Initialize actor π_φ, critics Q_θ1, Q_θ2, targets π'_φ, Q'_θ1, Q'_θ2
Initialize reward model r̂_ψ and empty replay buffer D
# Preference generation function
Function GeneratePreferences(τ1, τ2):
    for i ∈ {1, 2} do
        state_i ← Last DFA state reached in τ_i
        pos_i ← Last position in τ_i
        target_i ← Next resource or crafting table
        d_i ← ||pos_i − target_i||_2
        score_i ← w_s · state_i − w_d · d_i
    return preferred trajectory based on higher score
if Mode = Static then
    # Phase 1: Generate preferences once
    Generate random trajectory pairs {(τ_i, τ_j)}
    Compute preferences using GeneratePreferences
    Train reward model r̂_ψ using pairwise ranking loss
    # Phase 2: TD3 training with fixed reward model
    for t = 1 to T do
        s_t, q_t ← current environment and DFA states
        s̃_t ← [s_t, q_t] ;                                  /* Augmented state */
        a_t ← π_φ(s̃_t) + noise ;                            /* Exploration noise */
        Execute a_t, observe s_{t+1}
        q_{t+1} ← δ(q_t, L(s_{t+1})) ;                      /* DFA transition */
        r_t ← r̂_ψ(s_t, a_t, q_t) ;                          /* Query reward model */
        Store (s̃_t, a_t, r_t, s̃_{t+1}, done) in D
        if time to update then
            Sample mini-batch from D
            # TD3 updates (simplified)
            Update critics using target networks
            Update actor (delayed)
            Soft-update target networks
else
    # Dynamic mode: iteratively update reward model and policy
    for i = 1 to num_iterations do
        # Generate trajectories with current policy
        if i = 1 then
            Generate trajectory pairs {(τ_i, τ_j)} using random exploration
        else
            Generate trajectory pairs {(τ_i, τ_j)} using mixed random/policy actions
        # Compute preferences using DFA
        Compute preference scores as in Static mode
        # Update reward model with new preferences
        Train/update reward model r̂_ψ on new preferences
        # Policy optimization with TD3
        for t = 1 to T_iter do
            Collect experience and train TD3 as in Static mode
        # Evaluate agent performance
        Evaluate success rate over test episodes
        if agent performance is stable or sufficient then
            break
return Actor network π_φ (policy)
```

Algorithm 5: Q-Learning with Teacher-Distilled Automaton Knowledge

Input: Student environment, DFA $\mathcal{A}$, Teacher automaton Q-values $\overline{Q}_{\text{teacher}}(q, \sigma)$, Learning parameters α, γ, ϵ_0, ρ, Weights w_s, w_d, w_q, Mode $\in \{\text{Pref}, \text{Pref+Plan}\}$
Output: Optimal policy for student environment
Initialize reward model $\hat{r}_\theta$ with parameters θ
Initialize student Q-table: $Q_{\text{student}}[(s, q), a] \leftarrow 0$ for all $(s, q) \in S \times Q, a \in A$
Initialize transition counts: $n_{\text{student}}(q, \sigma) \leftarrow 0$ for all automaton transitions (q, σ) $\epsilon \leftarrow \epsilon_0$
Phase 1: Generate preferences using distilled knowledge
Generate trajectory pairs $\{(\tau_i, \tau_j)\}$ using random exploration
for *each trajectory pair* (τ_i, τ_j) **do**
  $N_{s1} \leftarrow$ Number of subgoals completed in τ_i by DFA $\mathcal{A}$
  $N_{s2} \leftarrow$ Number of subgoals completed in τ_j by DFA $\mathcal{A}$
  $\text{dist}_1 \leftarrow$ Distance from last state of τ_1 to next subgoal
  $\text{dist}_2 \leftarrow$ Distance from last state of τ_2 to next subgoal
  # Identify automaton transitions for Q-value lookup
  if $N_{s1} > 0$ **then**
    $\text{transition}_1 \leftarrow (N_{s1} - 1, N_{s1})$
  else
    $\text{transition}_1 \leftarrow (0, 0)$
  if $N_{s2} > 0$ **then**
    $\text{transition}_2 \leftarrow (N_{s2} - 1, N_{s2})$
  else
    $\text{transition}_2 \leftarrow (0, 0)$
  $q\text{-value}_1 \leftarrow \overline{Q}_{\text{teacher}}(\text{transition}_1)$
  $q\text{-value}_2 \leftarrow \overline{Q}_{\text{teacher}}(\text{transition}_2)$
  $\text{score}_1 \leftarrow w_s \cdot N_{s1} - w_d \cdot \text{dist}_1 + w_q \cdot q\text{-value}_1$
  $\text{score}_2 \leftarrow w_s \cdot N_{s2} - w_d \cdot \text{dist}_2 + w_q \cdot q\text{-value}_2$
  if *score*$_1 >$ *score*$_2$ **then**
    Add preference $\tau_i \succ \tau_j$
  else if *score*$_2 >$ *score*$_1$ **then**
    Add preference $\tau_j \succ \tau_i$
# Phase 2: Train reward model $\hat{r}_\theta$ using pairwise ranking loss on preferences
# Phase 3: Q-learning with teacher knowledge
for $e = 1$ *to* E **do**
  $s \leftarrow$ initial state; $q \leftarrow q_0$
  while *not terminal* **do**
    probability $\epsilon a \leftarrow$ random action $a \leftarrow \arg\max_{a'} Q_{\text{student}}[(s, q), a']$
    $s' \leftarrow$ next state after taking a
    $q' \leftarrow \delta(q, L(s'))$; /* DFA transition */
    $\sigma \leftarrow L(s')$; /* Atomic proposition labeling s' */
    # Update transition count and compute annealing factor
    $n_{\text{student}}(q, \sigma) \leftarrow n_{\text{student}}(q, \sigma) + 1$
    $\beta(q, \sigma) \leftarrow \rho^{n_{\text{student}}(q, \sigma)}$; /* Annealing weight */
    # Compute reward from trained reward model
    $r \leftarrow \hat{r}_\theta((s, q), a)$
    if *Mode = Pref* **then**
      # Use reward model only for preferences, standard Q-learning update
      $Q_{\text{student}}[(s, q), a] \leftarrow$
       $Q_{\text{student}}[(s, q), a] + \alpha(r + \gamma \max_{a'} Q_{\text{student}}[(s', q'), a'] - Q_{\text{student}}[(s, q), a])$
    else if *Mode = Pref+Plan* **then**
      # Combined teacher Q-value with TD target (planning)
      $Q_{\text{target}} \leftarrow r + \gamma \max_{a'} Q_{\text{student}}[(s', q'), a']$
      $Q_{\text{combined}} \leftarrow \beta(q, \sigma) \cdot \overline{Q}_{\text{teacher}}(q, \sigma) + (1 - \beta(q, \sigma)) \cdot Q_{\text{target}}$
      $Q_{\text{student}}[(s, q), a] \leftarrow Q_{\text{student}}[(s, q), a] + \alpha(Q_{\text{combined}} - Q_{\text{student}}[(s, q), a])$
    $s \leftarrow s'$; $q \leftarrow q'$
  $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\min})$
return Derived policy $\pi(s, q) = \arg\max_a Q_{\text{student}}[(s, q), a]$

C Description and DFA Visualizations for Each Environment

In this section, we provide the description and visualizations of the four environments along with their DFA representations. Each DFA encodes the logical structure of subgoal dependencies and task constraints, guiding the agent in task completion. The states represent distinct progress levels, with transitions triggered by specific subgoal achievements. These automata serve as an interpretable abstraction of the environment dynamics, ensuring structured exploration and efficient policy learning.

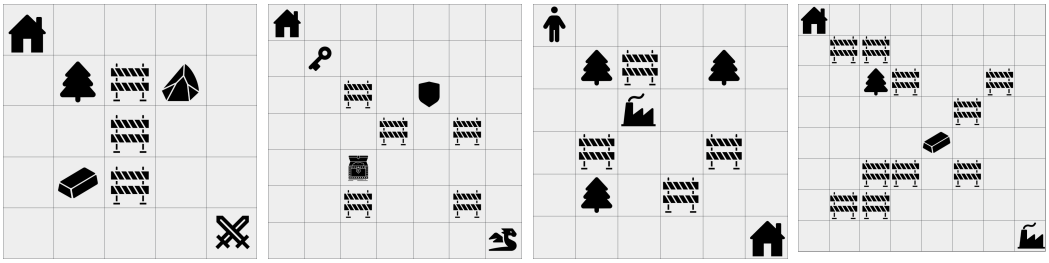

Figure 6: Visualization of the four environments: Minecraft Iron Sword Quest, Dungeon Quest, Blind Craftsman, and Minecraft Building Bridge (left to right).

1) Minecraft Iron Sword Quest (5×5). The agent starts at a *Home Base* and must reach a *Crafting Table*, completing subgoals in a strict order while navigating obstacles. It must gather wood to craft a Wooden Pickaxe, use it to mine stone and craft a Stone Pickaxe, and finally mine iron ore to craft the sword at the Crafting Table. This type of structured environment has been widely explored Andreas et al. (2017b), including tasks like *Make Bed* and *Make Axe*.

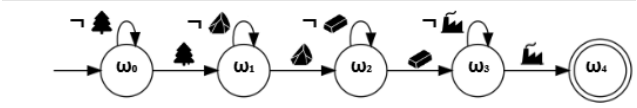

Figure 7: DFA for *Minecraft Iron Sword Quest*. The automaton enforces the correct order of resource collection before crafting the sword.

2) Dungeon Quest (7×7). The agent navigates a grid to collect items and defeat a dragon, following a strict sequence of subgoals Singireddy et al. (2023). The agent must first obtain a *Key* to unlock the *Chest*, retrieve the *Sword* from the Chest, and collect the *Shield* for protection. The Dragon can only be defeated with both the Shield and Sword. These dependencies are captured in the DFA, with transitions triggered by item acquisition.

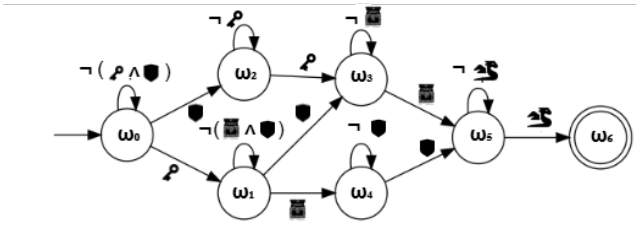

Figure 8: DFA for *Dungeon Quest*. The sequence requires acquiring the Key, Shield, and Chest before facing the Dragon.

3) Blind Craftsman (6×6). This grid world features multiple paths and potential loops within its subgoal structure. The agent must gather wood, craft tools, and return home. The environment includes wood sources (up to two pieces collected at a time), the factory (for crafting tools), and the home (the final goal, accessible only after crafting all tools). The agent alternates between wood collection and factory visits to craft three tools before returning home. Task dependencies ensure tools require sufficient wood, and the mission is incomplete until all three tools are crafted. The DFA encodes these dependencies.

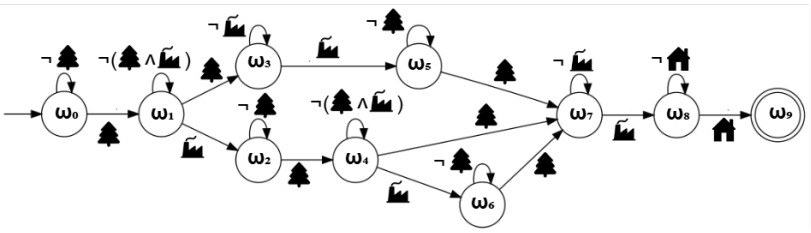

Figure 9: DFA for *Blind Craftsman*. The agent must alternate between collecting wood and visiting the Factory to craft tools before returning Home.

4) Minecraft Building Bridge (8×8). The agent must build a bridge by collecting *Wood* and *Iron* and using them at the *Factory*, the final subgoal Andreas et al. (2017b); Icarte & et al. (2018). Starting at a *Home Base*, the agent navigates the grid, overcoming obstacles like rivers, rocks, and trees. Unlike previous tasks, *Wood* and *Iron* can be collected in any order before reaching the *Factory*, allowing multiple valid sequences. The DFA encodes these dependencies, ensuring the task is completed only after both materials are collected and the bridge is constructed.

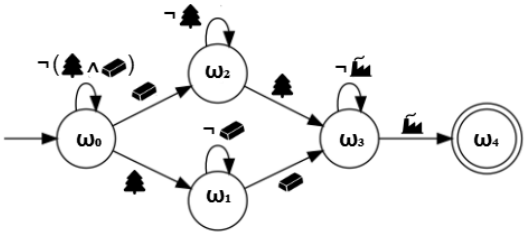

Figure 10: DFA for *Minecraft Building Bridge*. The agent can collect Wood and Iron in any order before utilizing the Factory to complete the bridge.

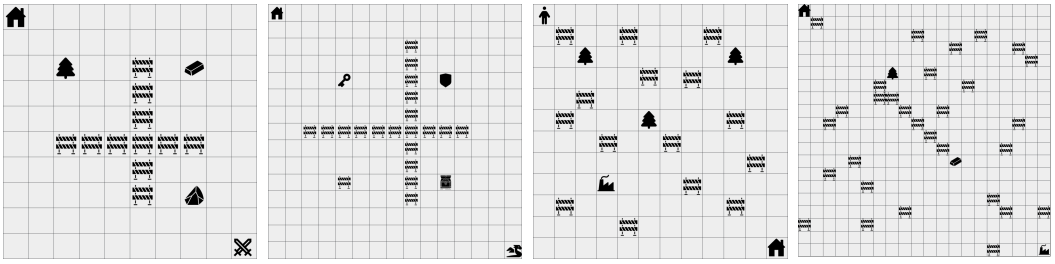

Figure 11: Visualization of the four student environments.

5) Mountain Car Collection (Physics-Based Navigation). The agent operates in a 1D mountainous terrain with 20 discrete positions, where movement is constrained by energy levels and terrain difficulty. The state space is 9-dimensional, comprising normalized position (1D), energy level (5D one-hot encoding), and inventory status for three collectible items (3D binary). The environment features a challenging landscape with varying heights that affect energy consumption during navigation. The agent must sequentially collect four items: Power Cell, Sensor Array, Data Crystal, and Base Station, following strict temporal dependencies encoded in the DFA. The agent has five energy states (depleted, low, medium, high, max) that determine movement capabilities, with energy consumption based on terrain elevation changes and obstacle encounters. Obstacles at positions 5, 10, and 15 impose additional energy penalties. The agent's movement distance is limited by its current energy level, and the rest action provides energy recovery. This environment validates our approach's effectiveness in physics-based domains with resource management constraints, extending beyond discrete navigation tasks to multi-dimensional state spaces with continuous constraint satisfaction problems.

Figure 12: DFA for *Mountain Car Collection*. The automaton enforces sequential collection of power cell, sensor array, data crystal, and base station return in strict order.

6) Warehouse Robotics (High-Dimensional Continuous). This environment implements a realistic 12-dimensional continuous state space representing a warehouse robotics scenario, comprising normalized robot coordinates (2D), scanner possession status (1D), normalized scanner battery level (1D), individual task completion flags for zone scanning, item scanning, item pickup, and delivery (4D), normalized overall task progress (1D), normalized step count (1D), and binary proximity indicators for scan zone and shipping dock (2D). The robot operates in a 6×8 grid with continuous position coordinates, requiring precise navigation and state management. The task involves a 5-step sequential operation: obtaining a scanner from the station, navigating to the scan zone to perform inventory scanning, returning the scanner to the charging station for battery replenishment, collecting the identified item from its location, and delivering it to the shipping dock. Each subtask must be completed in the correct order as tracked by the automaton state progression. This high-dimensional representation mirrors real-world robotic systems where agents must simultaneously track position, equipment status, battery levels, individual task completion states, and environmental proximity—creating a challenging continuous control problem. This environment directly addresses scalability concerns for robotic applications, demonstrating our framework's capability to handle realistic industrial automation scenarios with the complex temporal constraints and multi-modal state representations typical of modern warehouse management systems.

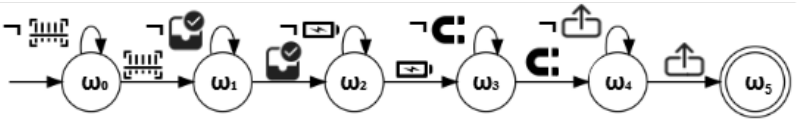

Figure 13: DFA for *Warehouse Robotics*. The 6-state automaton encodes the complete warehouse workflow: scanner acquisition, zone scanning, scanner return, item pickup, and delivery completion.

D DISTILLATION WITH REWARD SHAPING

Here we provide more details on the *Distillation with Reward Shaping* method. In contrast to the preference-based algorithms or known reward function method, this approach relies on a given automaton structure to shape a minimal numeric reward. It neither compares trajectories pairwise nor uses a handcrafted reward function. Instead, it leverages automaton transitions to provide positive feedback upon subgoal completion.

D.1 METHOD OVERVIEW

We assume the agent must satisfy a sequence of subgoals described by a DFA $\mathcal{A}$. Instead of assigning custom numeric rewards for each subgoal or comparing pairs of trajectories, the agent obtains:

- $+1$ reward whenever it triggers an automaton transition associated with completing a subgoal step;
- -0.1 penalty per action step otherwise, discouraging wandering.

No additional reward shaping is introduced. Thus, once the agent transitions from automaton state ω to ω', it knows it has *correctly* advanced the subgoal sequence. If q' is an accepting automaton state, the episode terminates successfully.

Remark D.1. *If the automaton is* learned *(e.g., via RPNI Oncina & García (1992)), that step happens* outside *this RL loop. In our experiments, we treat the automaton as* given*, so the policy training procedure focuses on the environment plus this minimal numeric shaping.*

D.2 KNOWLEDGE TRANSFER (TEACHER–STUDENT SETUP)

To accelerate learning in more complex environments, we adopt a teacher–student paradigm:

1. **Teacher phase (simpler or well-understood environment).** We run Q-learning in a product MDP $\mathcal{M}_d \otimes \mathcal{A}$, where $\mathcal{M}_d$ is a discrete "teacher" environment with the same subgoal structure. The teacher records Q-values for each automaton transition, effectively capturing how "valuable" it was to advance subgoals from one automaton state ω to another.
2. **Distillation of automaton transitions.** We aggregate the teacher's Q-values over transitions (ω, σ), where σ is the atomic proposition labeling the next subgoal. This yields an *average* or *representative* Q-value, $Q^{\text{avg}}_{\text{teacher}}(\omega, \sigma)$, for each automaton transition.
3. **Student phase (target environment).** We then train a student agent in the product MDP $\mathcal{M}_s \otimes \mathcal{A}$, where $\mathcal{M}_s$ is the new (potentially more complex) environment but *with the same* automaton $\mathcal{A}$. The student still uses minimal numeric shaping ($+1$ on subgoal transitions, -0.1 otherwise), *but* it incorporates the teacher's knowledge in its Q-updates:

$$Q'_{\text{student}}((s,\omega),a) = \beta(\omega,\sigma)Q^{\text{avg}}_{\text{teacher}}(\omega,\sigma) + \big(1-\beta(\omega,\sigma)\big)\Big(r+\gamma \max_{a'} Q_{\text{student}}((s',\omega'),a')\Big),$$

   where $\beta(\omega,\sigma) = \rho^{\eta(\omega,\sigma)}$ controls the balance between the teacher's guidance and the student's own experience, annealing as the agent visits each automaton transition more frequently ($\eta(\omega,\sigma)$ is the visit count).
4. **Policy optimization.** The student updates its Q-table as:

$$Q((s_t,\omega_t),a_t) \leftarrow Q((s_t,\omega_t),a_t) + \alpha\Big(r(\omega_t,\sigma_t) + \gamma \max_{a'} Q((s_{t+1},\omega_{t+1}),a') - Q((s_t,\omega_t),a_t)\Big),$$

   gradually converging to a policy π^* that balances teacher guidance with environment-specific exploration.

D.3 RELATION TO OUR MAIN APPROACHES

Unlike the **static** and **dynamic** preference-based methods, which *learn* a reward model from trajectory comparisons, *Distillation with Reward Shaping* simply exploits the automaton structure to produce local numeric signals. It also differs from the **known reward function** approach in that we do not

hand-specify the reward terms for each subgoal or final goal. Instead, the DFA transitions themselves indicate subgoal completions, providing minimal positive increments to guide the agent. Experiments illustrate how this method can serve as a teacher for subsequent knowledge transfer, ultimately enabling efficient learning in new or larger environments with the same subgoal logic.

E REWARD MACHINE BASELINE

Here we provide more details on the *Reward Machine (RM)* baseline method. This approach directly defines rewards based on automaton state transitions, offering a formalized alternative to preference-based learning for non-Markovian tasks.

E.1 METHOD OVERVIEW

RMs provide an interpretable mechanism for encoding non-Markovian reward functions Icarte et al. (2022); Icarte & et al. (2018). An RM is defined as a tuple $\mathcal{R} = (U, u_0, F, \delta_u, \delta_r)$, where:

- U is a finite set of states representing task progression
- $u_0 \in U$ is the initial state
- $F \subseteq U$ is the set of terminal states
- $\delta_u : U \times \mathcal{L} \to U$ is the state-transition function triggered by environmental events in $\mathcal{L}$
- $\delta_r : U \times \mathcal{L} \times U \to \mathbb{R}$ is the reward-transition function

Here, $\mathcal{L}$ denotes the set of high-level events or labels that can be detected in the environment. For example, in our tasks, $\mathcal{L}$ includes events such as "wood" (for collecting wood), "key" (for picking up a key), or "factory" (for reaching the factory location). These events serve as triggers for state transitions in the reward machine and are extracted from the environment state through a labeling function.

In our implementation, the agent obtains:

- $+5.0$ reward for each valid transition that represents subgoal completion in the correct order
- -1.0 penalty for attempting to complete subgoals out of order
- -0.1 penalty per action step to encourage efficiency
- $+10.0$ reward upon reaching the terminal state after all subgoals are completed in order

This approach explicitly encodes the temporal structure of the task, creating a more informative reward landscape while maintaining the logical constraints of sequential subtask completion Xu & et al. (2020); Icarte & et al. (2021).

E.2 IMPLEMENTATION DETAILS

Learning proceeds in the product MDP $\mathcal{M}_{\text{prod}} = \mathcal{M} \times \mathcal{R}$, where the agent's state is augmented with the current RM state. This creates a Markovian decision process from the originally non-Markovian task, as the RM state captures the necessary history information.

For policy learning in this product MDP, we use Q-learning with the following update rule:

$$Q((s,u),a) \leftarrow Q((s,u),a) + \alpha \left(r + \gamma \max_{a'} Q((s',u'),a') - Q((s,u),a) \right)$$

where (s,u) is the current state-RM state pair, a is the action, r is the reward received from the RM's reward-transition function δ_r, and (s',u') is the next state-RM state pair with $u' = \delta_u(u, L(s'))$, where $L(s')$ maps the environment state to relevant atomic propositions.

E.3 RELATION TO OUR APPROACH

While both the RM baseline and our preference-based method leverage automaton structures to handle non-Markovian rewards, they differ fundamentally in their approach:

- **Reward Specification:** RMs directly map automaton transitions to specific numeric rewards, requiring careful engineering of reward values. Our approach learns these values from preferences, eliminating the need for manual reward tuning.

- **Learning Mechanism:** RMs create a product MDP where learning occurs in an augmented state space. Our method separates preference generation from reward learning, enabling more flexible adaptation to task structure.
- **Transferability:** While both approaches can leverage automaton structures for transfer learning, our method's preference-based nature provides additional flexibility when transferring across environments with different dynamics but the same logical structure.

The RM baseline represents a state-of-the-art approach for structured reward specification in complex non-Markovian tasks, providing a formal alternative to our preference-based method.

F LTL-GUIDED REINFORCEMENT LEARNING (LPOPL) BASELINE

Here we provide more details on the *LTL-Guided RL (LPOPL)* baseline method. This approach leverages Linear Temporal Logic (LTL) to guide RL through formal task specifications, providing theoretically-grounded task guidance without relying on preference elicitation.

F.1 METHOD OVERVIEW

Logic-Guided Policy Optimization Hasanbeig & et al. (2020); Li & Belta (2017) uses LTL formulas to encode temporal task requirements. For sequential subgoal tasks, we encode the requirement as an LTL formula:

$$\varphi = \Diamond(g_1 \wedge \Diamond(g_2 \wedge \Diamond(\ldots \wedge \Diamond g_n)))$$

where $\Diamond$ is the "eventually" operator and g_i represents the i-th subgoal. This formula is then translated into a Deterministic Finite Automaton (DFA) $\mathcal{A}_\varphi = (Q, q_0, F, \Sigma, \delta)$, where states in Q track progress toward satisfying φ.

The key innovation in LPOPL is the use of potential-based reward shaping Ng & Russell (1999) derived from the automaton structure:

- A potential function $\Phi : Q \to \mathbb{R}$ assigns higher values to states closer to acceptance
- The shaped reward is defined as $R'(s, a, s') = R(s, a, s') + \gamma\Phi(\delta(q, L(s'))) - \Phi(q)$
- A small negative base reward $R(s, a, s') = -0.1$ encourages efficient paths
- Terminal states that satisfy the LTL formula receive a large positive reward (+10.0)

This approach preserves optimality guarantees while providing denser feedback aligned with the LTL specification Camacho & et al. (2019); Shah et al. (2023).

F.2 IMPLEMENTATION DETAILS

We implement the potential function Φ based on the shortest path distance from each automaton state to an accepting state:

$$\Phi(q) = \begin{cases} \max_{q' \in F}\{d(q, q')\} & \text{if } q \notin F \\ 0 & \text{if } q \in F \end{cases}$$

where $d(q, q')$ is the minimum number of transitions required to reach q' from q in the automaton. This creates a potential gradient that guides the agent toward satisfying the LTL specification.

The agent learns in the product MDP between the environment and the DFA, with the following Q-learning update:

$$Q((s, q), a) \leftarrow Q((s, q), a) + \alpha \left(R'(s, a, s') + \gamma \max_{a'} Q((s', q'), a') - Q((s, q), a) \right)$$

where (s, q) is the current state-automaton state pair, a is the action, R' is the shaped reward, and (s', q') is the next state-automaton state pair with $q' = \delta(q, L(s'))$.

F.3 RELATION TO OUR APPROACH

Both LPOPL and our preference-based method use automaton structures to address non-Markovian rewards, but they differ in several key aspects:

- **Formal Specification:** LPOPL requires LTL formulas to be manually translated into reward signals through potential functions. Our approach learns reward functions directly from automaton-generated preferences without requiring explicit reward engineering.
- **Reward Shaping:** LPOPL uses potential-based reward shaping to create dense rewards that preserve optimality. Our method learns a reward model directly from pairwise preferences, avoiding the need for careful shaping design.
- **Theoretical Properties:** LPOPL provides theoretical guarantees on optimality preservation through proper potential-based shaping. Our approach offers convergence guarantees based on preference consistency and sufficient exploration.

G ADDITIONAL EXPERIMENTAL RESULTS

This appendix provides additional experimental results that complement the findings presented in the main text. While the main paper focuses on reward per cumulative steps, here we present the complete set of metrics: reward per episode and steps per episode.

Figure 14 shows **the reward per episode** for the four environments considered. The *dynamic method* achieves stronger long-term gains through iterative refinement, though it starts with less stability. The *static method* converges rapidly with a fixed learned reward, often matching or slightly outperforming the *known reward function* baseline initially. Ultimately, the dynamic method surpasses all once it completes enough preference-driven updates.

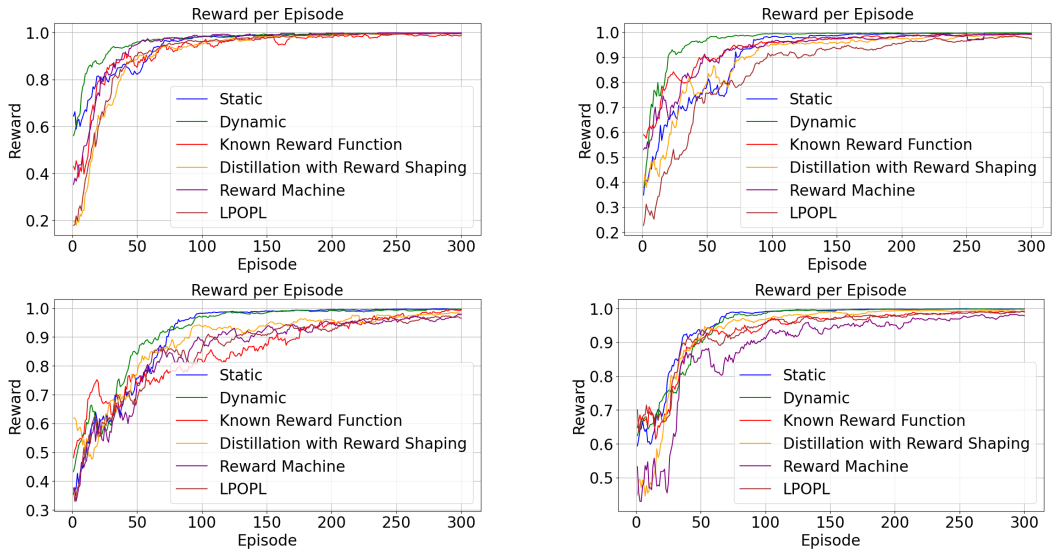

Figure 14: Reward per episode for all environments.

The **steps per episode** for each environment are shown in Figure 15. The static method tends to require fewer steps early on once the reward function is learned, but lacks further refinement. The dynamic method initially takes extra steps during its exploratory updates, yet ultimately converges to more efficient paths in two environments. However, in the other two environments, both methods still outperform the known reward function baseline. The *distillation-based , reward machine and LTL based methods* do not converge as quickly as the dynamic method but maintains stable performance.

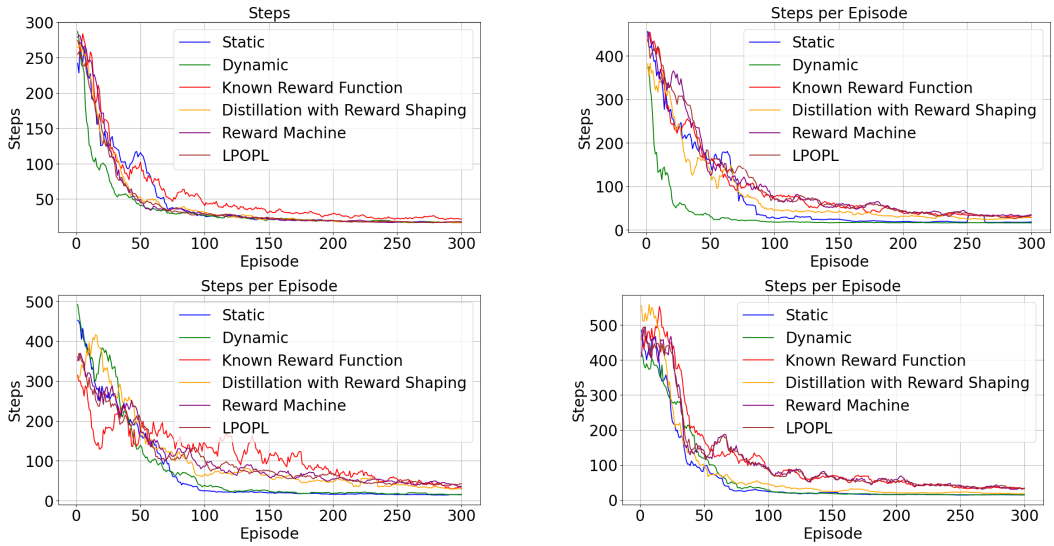

Figure 15: Steps per episode for all environments.

Combined scoring with transfer. We evaluate the effectiveness of combining subtask-based preferences with Q-value-based knowledge transferred from a teacher using the scoring function.

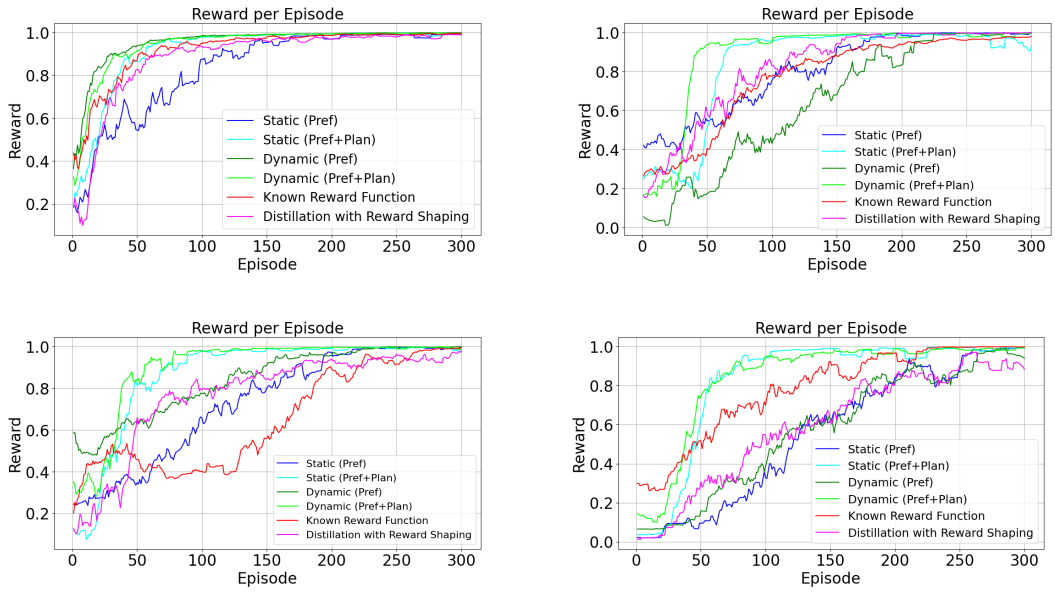

Figure 16: Reward per episode for all methods and environments.

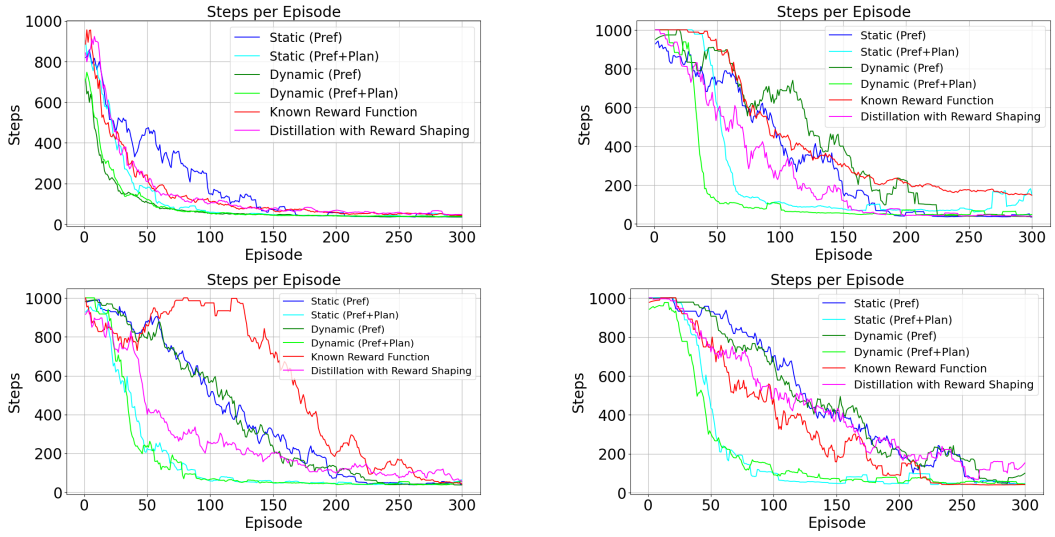

Figure 17: Steps per episode for all methods and environments.

In all environments, distillation agents consistently outperform non-distilled methods, as indicated by higher episode rewards (Figure 16), fewer steps per episode (Figure G), and a steeper reward increase over total steps (shown in the main text). These comprehensive results demonstrate the enhanced performance achieved by leveraging teacher-derived automaton knowledge across evaluation metrics.

H BELIEF-STATE BASED PREFERENCE ELICITATION FOR PARTIALLY OBSERVABLE ENVIRONMENTS

Agents may operate with limited information about their environment, in which case the fully observable assumption is does not hold. We extend our framework to address this challenge by introducing a belief-state based preference elicitation approach for partially observable environments.

Belief-State Based Scoring. In partially observable environments, the agent has limited information about the true state of the environment. We formalize a partially observable environment as a tuple $\mathcal{M}_{\text{po}} = (S, s_0, A, T, \Omega, O, R^*)$, where S, s_0, A, and T are defined as before, Ω is the set of observations, $O : S \times A \to \Delta(\Omega)$ is the observation function mapping state-action pairs to distributions over observations, and R^* is the unknown reward function.

Given that the agent cannot directly observe the full state, we maintain a belief state $b \in \Delta(S)$, representing a probability distribution over possible environment states. For each unobserved subgoal $g \in G$ (where G is the set of subgoals defined by the DFA), we define a belief value $b(g) \in [0, 1]$ indicating the agent's confidence that g is at a particular location.

The belief-state trajectory scoring function integrates three components: (i) *observed subgoal completion*, (ii) *expected progress toward unobserved subgoals*, and (iii) *information gain*. Formally, for a trajectory $\tau = (s_0, a_0, o_0, b_0, s_1, a_1, o_1, b_1, \ldots, s_T)$, where o_t represents the observation and b_t the belief state at time t, the score is computed as:

$$\text{score}(\tau) = w_c \cdot N_c(\tau) + w_b \cdot P_b(\tau) + w_i \cdot I(\tau), \tag{8}$$

where $N_c(\tau)$ is the number of subgoals confirmed to be completed (directly observed), $P_b(\tau)$ is a measure of expected progress based on belief states, and $I(\tau)$ is the information gain achieved throughout the trajectory. The weights w_c, w_b, and w_i prioritize different aspects, with typically $w_c > w_b > w_i$.

The expected progress term $P_b(\tau)$ is defined as:

$$P_b(\tau) = \sum_{g \in G} \sum_{t=0}^{T-1} \max(0, b_{t+1}(g) - b_t(g)), \tag{9}$$

which captures improvements in belief confidence for each subgoal over time. The information gain $I(\tau)$ quantifies the reduction in state uncertainty:

$$I(\tau) = \sum_{t=0}^{T-1} \big(H(b_t) - H(b_{t+1})\big), \tag{10}$$

where $H(b) = -\sum_{s \in S} b(s) \log b(s)$ is the entropy of belief state b.

Remark H.1. *The belief-state approach effectively handles partial observability by: (1) rewarding confirmed subgoal completions, (2) encouraging trajectories that improve belief accuracy about unobserved subgoals, and (3) promoting systematic exploration to reduce state uncertainty. This results in more efficient exploration in environments where complete state information is unavailable.*

The belief state b_t is updated after each action and observation using standard Bayesian inference:

$$b_{t+1}(s') = \eta \cdot O(o_{t+1}|s', a_t) \sum_{s \in S} T(s'|s, a_t) b_t(s), \tag{11}$$

where η is a normalization constant ensuring $\sum_{s' \in S} b_{t+1}(s') = 1$.

For applications in automaton-guided environments, we augment the belief state to incorporate confidence in DFA states. For each DFA state $q \in Q$, we maintain a confidence value $c(q) \in [0, 1]$, representing our degree of certainty that the automaton is in state q given partial observations. The DFA state confidence is updated based on observed evidence of subgoal completion and the structure of the automaton.

Given two trajectories τ_1 and τ_2, their scores under Equation equation 8 are compared to establish preferences. These belief-aware preferences enable the agent to learn effective policies even with limited observability, by focusing exploration on regions with high expected information gain and by intelligently tracking progress through the automaton's states even when direct confirmation is not available.

H.1 Belief-State Policy Optimization

Building upon our belief-state preference elicitation method, we extend the policy optimization approach to handle partially observable environments. Since the agent cannot directly observe the full state, we optimize over belief states rather than raw environment states.

Belief-State Value Function Approximation. We parameterize the reward function as $\hat{r}_\theta(s, b, q, c, a)$, where s is the directly observable component of the state, b is the belief state over unobserved elements, q is the observed part of the current DFA state, c is the confidence level in the unobserved part of the DFA state, and a is the action. This reward function is trained using the same pairwise ranking loss as in Equation equation 5, but operating over belief-augmented trajectories.

The Q-function for the belief-state MDP is learned as:

$$Q((s, b, q, c), a) = \mathbb{E}\left[\hat{r}_\theta(s, b, q, c, a) + \gamma \max_{a'} Q((s', b', q', c'), a')\right], \tag{12}$$

where (s', b', q', c') is the next belief-augmented state after taking action a in state (s, b, q, c).

Intrinsic Exploration Bonuses. To encourage efficient exploration under partial observability, we can augment the learned reward function with intrinsic motivation terms:

$$r_{\text{total}}(s, b, q, c, a) = \hat{r}_\theta(s, b, q, c, a) + \beta_1 r_{\text{info}}(b, a) + \beta_2 r_{\text{conf}}(c), \tag{13}$$

where $r_{\text{info}}(b, a)$ is an information gain bonus for actions that are expected to reduce uncertainty in the belief state, and $r_{\text{conf}}(c)$ rewards increases in DFA state confidence. The hyperparameters β_1 and β_2 control the relative importance of exploration versus exploitation.

The information gain bonus is formally defined as:

$$r_{\text{info}}(b, a) = \mathbb{E}_{o \sim O(\cdot|s,a)}\left[H(b) - H(b')\right], \tag{14}$$

where b' is the updated belief after taking action a and receiving observation o, and $H(\cdot)$ is the entropy function as defined earlier.

Belief-Augmented Dynamic and Static Variants. Similar to our fully observable setting, we propose two learning variants for the belief-state case:

In the **belief-based static variant**, the belief-augmented reward function is learned once from an initial set of preferences generated using Equation equation 8. The policy is then optimized using this fixed reward function, operating over belief states rather than raw states.

In the **belief-based dynamic variant**, both the belief-augmented reward function and the policy are refined iteratively. At each iteration, belief-augmented trajectories are generated using the current policy, and preferences for these trajectories are derived using our belief-state scoring approach. The reward function is updated based on these new preferences, and the policy is re-optimized accordingly. This iterative process continues until convergence, allowing for progressive refinement of both the reward model and policy under partial observability.

H.2 Theoretical Properties

The belief-state approach extends our framework to partially observable settings while preserving key theoretical properties. Here, we analyze the relationship between the original non-Markovian task specification and our belief-state solution.

Proposition H.2 (Belief-State Optimality)**.** *Let π_b^* be the optimal policy for the belief-state MDP under the learned reward function $\hat{r}_\theta$. If the belief state tracking is accurate and the learned reward function satisfies $|\hat{R}_\theta(\tau) - R^*(\tau)| \leq \varepsilon_r$ for trajectories τ generated by π_b^*, then π_b^* is ε-optimal with respect to the true non-Markovian objective, where $\varepsilon = \varepsilon_r + \varepsilon_b$ and ε_b is the error introduced by belief state approximation.*

Proof Sketch. The belief-state MDP is a sufficient statistic for optimal decision-making in POMDPs Kaelbling et al. (1998). By incorporating the DFA state and confidence into the belief representation,

we ensure that the non-Markovian aspects of the task are captured in the augmented state space. The error term ε_b accounts for imperfect belief updating due to approximation errors or model misspecification. Given accurate belief tracking and a well-learned reward function, the resulting policy will be near-optimal for the original non-Markovian objective. □

Remark H.3 (Information-Directed Exploration)**.** *The belief-state scoring function in Equation equation 8 induces preferences that favor trajectories which efficiently gather information relevant to task completion, balancing exploitation (subgoal completion) with exploration (information gain). This is because the scoring function rewards three components: confirmed subgoal completion (N_c), expected progress via improved beliefs (P_b), and information gain (I). Trajectories that maximize this score will necessarily balance immediate task progress with exploration that reduces uncertainty about unobserved subgoals. This balance is optimal for information-directed exploration Russo et al. (2018), which prioritizes reducing uncertainty about task-relevant aspects of the environment.*

H.3 Algorithmic Implementation

Here, we present the algorithm for belief-state based preference learning in partially observable environments. Algorithm 6 details the procedure for generating preferences using belief states, and Algorithm 7 outlines the full belief-state reinforcement learning procedure.

Algorithm 6: Belief-State Based Preference Computation

Input: Trajectory pairs $\{(\tau_i, \tau_j)\}$, DFA $\mathcal{A}$, Subgoals $\mathcal{G} = [g_1, g_2, \ldots, g_N]$, Goal state s_{goal}, Weights w_c, w_b, w_i
Output: Preferences $P(\tau_i, \tau_j)$ for each trajectory pair
for *each trajectory pair* (τ_i, τ_j) **do**
  for *each trajectory* $\tau \in \{\tau_i, \tau_j\}$ **do**
    Initialize DFA $\mathcal{A}$ to its initial state;
    Initialize belief state b_0 with uniform distribution over unobserved elements;
    $N_c(\tau) \leftarrow 0$; // Confirmed subgoals
    $P_b(\tau) \leftarrow 0$; // Belief progress
    $I(\tau) \leftarrow 0$; // Information gain
    for *each step* t *in trajectory* $\tau = (s_0, a_0, o_0, \ldots, s_T)$ **do**
      if *step* $t > 0$ **then**
        Update belief state b_t based on b_{t-1}, a_{t-1}, o_t;
        for *each subgoal* $g \in \mathcal{G}$ **do**
          if g *newly observed in* o_t *AND matches next required subgoal* **then**
            $N_c(\tau) \leftarrow N_c(\tau) + 1$;
          $P_b(\tau) \leftarrow P_b(\tau) + \max(0, b_t(g) - b_{t-1}(g))$;
        $I(\tau) \leftarrow I(\tau) + (H(b_{t-1}) - H(b_t))$;
      Update DFA state and confidence based on observation o_t and belief b_t;
    score$(\tau) \leftarrow w_c \cdot N_c(\tau) + w_b \cdot P_b(\tau) + w_i \cdot I(\tau)$;
  if *score*$(\tau_i) >$ *score*(τ_j) **then**
    $P(\tau_i, \tau_j) \leftarrow \tau_i \succ \tau_j$;
  else if *score*$(\tau_j) >$ *score*(τ_i) **then**
    $P(\tau_i, \tau_j) \leftarrow \tau_j \succ \tau_i$;
  else
    $P(\tau_i, \tau_j) \leftarrow$ Indifferent;
return $P(\tau_i, \tau_j)$ for all trajectory pairs;

The key components of our belief-state preference learning approach are:

1. Belief State Tracking: We maintain a probability distribution over unobserved environment elements, particularly focusing on potential subgoal locations.

2. Belief-Aware DFA: The DFA not only tracks logical progression through the task but also maintains confidence levels in its current state based on partial observations.

Algorithm 7: Belief-State RL with Automaton-Based Preferences

Input: POMDP $\mathcal{M}_{\text{po}}$, DFA $\mathcal{A}$, Learning rates α, β, Margin m, Mode {Static, Dynamic}
Output: Optimal policy π^*

```
// Phase 1: Belief-State Preference Generation and Reward
   Learning
```

Initialize belief state representation b_0;
if *Static mode* **then**
  Generate trajectories $\{\tau_i\}$ using random policy with belief tracking;
  Compute preferences using Algorithm 6;
  Train belief-augmented reward model $\hat{r}_\theta(s, b, q, c, a)$ using pairwise ranking loss;
else
  Initialize policy π;
  for *iterations* $k = 1, 2, \ldots$ *until convergence* **do**
    Generate trajectories $\{\tau_i\}$ using current policy π with belief tracking;
    Compute preferences using Algorithm 6;
    Update belief-augmented reward model $\hat{r}_\theta(s, b, q, c, a)$ using pairwise ranking loss;
    Add intrinsic motivation terms to reward: $r_{\text{total}} = \hat{r}_\theta + \beta_1 r_{\text{info}} + \beta_2 r_{\text{conf}}$;
    Optimize policy π using belief-state Q-learning with r_{total};
    if *policy performance converges* **then**
      break;

```
// Phase 2: Policy Optimization (Static mode only)
```

if *Static mode* **then**
  Add intrinsic motivation terms to reward: $r_{\text{total}} = \hat{r}_\theta + \beta_1 r_{\text{info}} + \beta_2 r_{\text{conf}}$;
  for *training iterations* **do**
    Optimize policy π using belief-state Q-learning with r_{total};
return final policy π^*

3. Information-Directed Scoring: The preference scoring mechanism balances confirmed progress (subgoals directly observed), belief-based progress (improved confidence in unobserved subgoals), and information gain (reduction in state uncertainty).

4. Intrinsic Motivation: The reward function incorporates exploration bonuses for actions that efficiently gather task-relevant information, promoting systematic exploration of the environment.

This framework can enable effective learning in partially observable environments by guiding exploration toward task completion even when direct observation of all subgoals is not possible. The belief state tracking provides robustness against observation noise and occlusion, while the automaton structure ensures that the learned policy respects the temporal constraints of the task. Future work will address empirical validation.

I Scalability Analysis for High-Dimensional State Spaces

In this section, we provide a formal analysis of how our automaton-based preference-guided RL framework scales to high-dimensional problems, establishing both theoretical guarantees and practical considerations.

I.1 Theoretical Complexity Analysis

Definition I.1 (Product MDP Construction)**.** Given an MDP $\mathcal{M} = (S, s_0, A, T, r)$ and a DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, the product MDP is defined as $\mathcal{M}_{\text{prod}} = (S \times Q, A, T_{\text{prod}}, (s_0, q_0), R_{\text{prod}})$.

For each state-action pair $(s, q, a) \in S \times Q \times A$, we must compute transitions to all possible next states $(s', q') \in S \times Q$. Computing each transition probability requires $\mathcal{O}(1)$ time for the environment transition and $\mathcal{O}(|Q|)$ time in the worst case for the DFA transition function. Therefore, the construction of the product MDP $\mathcal{M}_{\text{prod}}$ has time complexity $\mathcal{O}(|S|^2 \cdot |Q|^2 \cdot |A|)$. For storing the transition function T_{prod}, the space complexity $\mathcal{O}(|S|^2 \cdot |Q|^2 \cdot |A|)$ in the worst case.

This reveals that the product MDP construction exhibits polynomial scaling in all relevant parameters, but the quadratic dependence on both $|S|$ and $|Q|$ can become problematic in high-dimensional spaces. Fortunately, our function approximation approach mitigates this theoretical complexity.

Theorem I.2 (Sample Complexity for Preference Learning)**.** *Let $\mathcal{H}$ be a hypothesis class of reward functions with VC dimension $d_{\mathcal{H}}$. To learn a reward function that achieves error at most ϵ with probability at least $1 - \delta$, the required number of preference pairs is:*

$$N = \mathcal{O}\left(\frac{d_{\mathcal{H}} + \log(1/\delta)}{\epsilon^2}\right) \tag{15}$$

The proof follows from standard results in statistical learning theory, applying PAC learning bounds to the preference learning setting. Specifically, we can reduce preference learning to binary classification where each pair of trajectories forms a training instance, and the goal is to predict the preferred trajectory. The sample complexity then follows from the VC dimension bound for binary classification.

This theorem demonstrates that the sample complexity depends primarily on the complexity of the reward function class rather than the dimensionality of the state space directly, suggesting better scaling properties than approaches that must learn value functions over the entire state space.

I.2 Function Approximation for High-Dimensional Spaces

To practically address high-dimensional state spaces, we introduce a formal extension using function approximation:

Definition I.3 (Function Approximator)**.** Let $\phi : S \times Q \times A \to \mathbb{R}^d$ be a feature mapping that projects state-automaton-action tuples into a d-dimensional feature space, where $d \ll |S| \cdot |Q| \cdot |A|$. The reward function is parameterized as $\hat{r}_\theta(s, q, a) = f_\theta(\phi(s, q, a))$, where $f_\theta : \mathbb{R}^d \to \mathbb{R}$ is a function approximator (e.g., neural network) with parameters θ.

This representation allows us to establish:

Proposition I.4 (Dimensionality Reduction)**.** *The computational complexity of policy optimization in the product MDP using function approximation scales with $\mathcal{O}(d)$ rather than $\mathcal{O}(|S| \cdot |Q|)$, where d is the dimension of the feature space.*

Proof. With function approximation, policy updates require computing gradients with respect to θ, which scales with the parameter count rather than state space size. Specifically, the computational complexity of a gradient update is $\mathcal{O}(|\theta|)$, where $|\theta|$ is the number of parameters in the function approximator. For a neural network with fixed architecture, $|\theta| = \mathcal{O}(d)$, where d is the input dimension. □

I.3 Automaton-Guided Dimensionality Reduction

We introduce techniques specifically designed to leverage the automaton structure for more efficient learning in high-dimensional spaces:

Definition I.5 (Automaton-Guided Attention)**.** For a given automaton state $q \in Q$, we define an attention mask $M_q : \{1, ..., \dim(S)\} \to [0, 1]$ that highlights state dimensions relevant to the current automaton state. The attended state representation is:

$$\tilde{s}_q = s \odot M_q \tag{16}$$

where $\odot$ denotes element-wise multiplication.

Proposition I.6 (Attention Efficiency)**.** *Using automaton-guided attention reduces the effective dimensionality of the state space from* $\dim(S)$ *to* $\|M_q\|_0$ *on average, where* $\|M_q\|_0$ *is the number of non-zero elements in* M_q*.*

Proof. By focusing only on dimensions with non-zero attention weights, the computational operations scale with the number of attended dimensions rather than the full state dimension. This is because the gradient updates will be zero for dimensions with zero attention, effectively reducing the dimensionality of the optimization problem. □

Definition I.7 (Automaton-State Conditional Independence)**.** Let X_i be the i-th dimension of the state space S. Given an automaton state $q \in Q$, we say that dimensions X_i and X_j are conditionally independent given q if

$$P(X_i, X_j|q) = P(X_i|q) \cdot P(X_j|q) \tag{17}$$

Remark I.8 (Factorized Representation)**.** *If the state dimensions exhibit conditional independence given the automaton state, then the Q-function can be decomposed as:*

$$Q((s, q), a) = \sum_{i=1}^{K} Q_i((s_i, q), a) \tag{18}$$

where K is the number of independent components and s_i is the i-th component of the state. In particular, if the state dimensions are conditionally independent given the automaton state, then the transition dynamics can be factorized. For factorized transition dynamics, the Q-function can be decomposed into a sum of component Q-functions, each operating on a lower-dimensional subspace, reducing the curse of dimensionality.

I.4 Transfer Learning with Automaton Transitions

Our Q-value-based scoring mechanism offers significant advantages for scaling to complex tasks through curriculum learning.

In particular, let $\mathcal{M}_1$ and $\mathcal{M}_2$ be environments with state spaces of dimension d_1 and d_2 respectively ($d_2 > d_1$), sharing the same DFA structure $\mathcal{A}$. The sample complexity to learn a near-optimal policy in $\mathcal{M}_2$ after transfer from $\mathcal{M}_1$ is reduced by a factor of $\Omega\left(\frac{d_2}{d_1}\right)$ compared to learning from scratch.

To see that, note that without transfer, learning in $\mathcal{M}_2$ requires exploration in a d_2-dimensional space, which has sample complexity exponential in d_2 in the worst case. With transfer, the Q-value-based scoring transfers knowledge at the level of automaton transitions, which are invariant to the specific state representation. This means that the agent must only learn the mapping between state features and automaton transitions, rather than the full value function. The sample complexity is thus dominated by the complexity of learning this mapping, which scales linearly with the state dimension, yielding the stated improvement factor.

I.5 Exploration Efficiency in High-Dimensional Spaces

One of the key challenges in high-dimensional spaces is efficient exploration. We establish theoretical guarantees for our automaton-guided exploration approach:

Theorem I.9 (Exploration Hardness)**.** *In an n-dimensional continuous state space with sparse rewards defined by an m-state DFA, the expected time to discover a trajectory that reaches an accepting state through random exploration is $\Omega(c^n)$ for some constant $c > 1$, while our automaton-guided approach reduces this to $\mathcal{O}(m \cdot poly(n))$.*

Proof Sketch. Random exploration suffers from the curse of dimensionality, with the volume of the state space growing exponentially with the dimension n. In contrast, automaton-guided exploration decomposes the problem into subgoals, as defined by the DFA states. The number of samples needed scales with the number of automaton states m and a polynomial function of the state dimension n for each transition between automaton states, rather than exponentially with the state dimension. □

Proposition I.10 (Reward Density)**.** *Let R_{env} be the original sparse reward function of the environment, and $\hat{R}_\theta$ be our learned reward function from automaton-guided preferences. The density of non-zero rewards under $\hat{R}_\theta$ is $\Theta(m/|S|)$, while under R_{env} it is $\Theta(1/|S|)$, where m is the number of DFA states.*

Proof. The original sparse reward function typically provides a non-zero reward only upon reaching the final goal, which is a single state out of $|S|$, giving a density of $\Theta(1/|S|)$. Our learned reward function, guided by automaton transitions, provides meaningful rewards at each transition between DFA states, which occurs $m - 1$ times in a successful trajectory, resulting in a density of $\Theta(m/|S|)$. □

I.6 Limitations and Theoretical Bounds

Despite the advantages of our approach, we acknowledge several fundamental limitations:

1. DFA State Explosion: For certain complex tasks, the number of DFA states can grow exponentially with the number of subtasks k, i.e., $|Q| = \Omega(c^k)$ for some $c > 1$.
   In particular, if subtasks have complex dependencies and ordering constraints, the DFA must represent all valid combinations and sequences, which can grow exponentially with the number of subtasks in the worst case.
2. Preference Generation Cost: The computational cost of generating preferences using our method scales with $\mathcal{O}(|\tau|^2)$, where $|\tau|$ is the number of trajectories,
   since we must compute and compare the scores for each pair of trajectories, resulting in a quadratic scaling with the number of trajectories.

Theorem I.11 (Approximation Error Bound)**.** *Let π^* be the optimal policy for the true reward function R^*, and $\hat{\pi}$ be the policy learned using our approach. Then, under mild assumptions:*

$$\|V^{\pi^*} - V^{\hat{\pi}}\|_\infty \leq \frac{2\epsilon_r}{(1-\gamma)^2} + \frac{2\gamma^H}{1-\gamma} \tag{19}$$

where ϵ_r is the maximum error in the learned reward function, γ is the discount factor, and H is the horizon length of the DFA (the maximum number of transitions required to reach an accepting state).

Proof Sketch. The performance difference can be decomposed into two terms: (1) the error due to imperfect reward learning, bounded by $\frac{2\epsilon_r}{(1-\gamma)^2}$ using standard results from approximate dynamic programming, and (2) the error due to finite-horizon approximation of the DFA-guided rewards, bounded by $\frac{2\gamma^H}{1-\gamma}$. □

I.7 Conclusion on Scalability

Our theoretical analysis demonstrates that the proposed automaton-based preference-guided RL approach exhibits favorable scaling properties for high-dimensional problems. The key advantages include:

1. The decomposition of complex tasks via automaton structures, reducing the effective complexity of the learning problem.

2. The use of function approximation for reward modeling, which scales with the parameter count rather than the state space size.

3. The efficient transfer of knowledge through Q-value-based scoring, enabling curriculum learning from simpler to more complex environments.

4. The implementation of automaton-guided attention mechanisms, which reduce the effective dimensionality by focusing on task-relevant features.

These properties enable our approach to theoretically scale to high-dimensional state spaces while maintaining its core advantage in handling non-Markovian rewards without manual reward engineering. While practical implementation challenges remain, particularly for very high-dimensional problems, our theoretical analysis provides a solid foundation for extending the approach to such settings in future work.