Hybrid Offline Passive Grammatical Inference and Online Planning for Non-Markovian Tasks

Mahyar Alinejad
Dept. of Electrical and Computer Engineering
University of Central Florida
Orlando, Florida
mahyar.alinejad@ucf.edu

Alvaro Velasquez
Dept. of Computer Science
University of Colorado
Boulder, Colorado
alvaro.velasquez@colorado.edu

Yue Wang
Dept. of Electrical and Computer Engineering
Dept. of Computer Science
University of Central Florida
Orlando, Florida
yue.wang@ucf.edu

George Atia
Dept. of Electrical and Computer Engineering
Dept. of Computer Science
University of Central Florida
Orlando, Florida
george.atia@ucf.edu

***Abstract*—Planning in non-Markovian environments often requires inferring task structures, such as reward machines, through interactions with the environment. Traditional active grammatical inference methods, like Angluin's L^* algorithm, depend on continuous querying to learn task structures for the underlying planning objectives. In contrast, we propose a hybrid approach that combines passive grammatical inference, using the Regular Positive and Negative Inference (RPNI) algorithm, with online planning. By leveraging pre-collected positive and negative trajectories, RPNI learns a deterministic finite automaton (DFA) that captures the task structure, significantly reducing the need for real-time interactions. Subsequently, online planning is conducted over the product MDP, which integrates the environment with the learned DFA. This hybrid methodology minimizes the cost of online interactions and improves learning efficiency in complex environments. Our approach outperforms baseline algorithms in terms of runtime and sample complexity, and is well-suited for real-world scenarios where task structures are implicit, and interactions with the environment are expensive.**

***Index Terms*—Passive Grammatical Inference, RPNI Algorithm, Automaton Learning, Non-Markovian Planning**

I. Introduction

In reinforcement learning (RL), solving complex tasks often involves defining non-Markovian objectives, where the rewards depend on the history of states rather than the current state alone [1]. Traditional RL methods struggle with such problems due to sparse or delayed reward signals. To address this, automaton-based approaches have been developed, where task specifications are encoded in a deterministic finite automaton (DFA). However, many of these methods rely on active learning, requiring extensive online interaction with the environment, which can be resource-intensive and time-consuming.

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

A. Relation to Prior Work

Our method builds on several key advancements in RL such as hierarchical RL (HRL), non-Markovian planning, and grammatical inference.

Hierarchical RL. Classical approaches such as MAXQ [2] and Options [3] focus on task decomposition and sub-goal learning but are limited by their dependence on predefined hierarchies, which can constrain the policy space and hinder convergence to optimal solutions. In contrast, we use DFAs to capture task sequences directly in the reward structure, avoiding HRL's policy restrictions.

Non-Markovian planning. Recent works on reward machines, such as [4] and [5], use finite-state automata to represent non-Markovian reward functions. Unlike Q-learning for Reward Machines (QRM) [4], which integrates standard Q-learning with a known reward machine to guide the agent's learning process, we adopt the Regular Positive and Negative Inference (RPNI) algorithm [6] to passively learn DFAs from pre-collected data. This reduces the need for real-time querying, making our approach more practical for environments where direct interaction is costly or impractical.

Grammatical inference. Active grammatical inference methods, such as the L^* algorithm [7], have been widely used to learn automata by querying the environment through membership and equivalence queries. Techniques such as [8], [9], [10], and [11] apply L^* to learn automata in RL and planning settings by proposing policies to answer these queries. The rise of foundation models, particularly large language models, has also led to renewed interest in grammatical inference by using these models as oracles or sources of knowledge from which DFAs can be extracted [12]–[14]. However, active inference often requires continuous interaction with the environment, which may not be feasible in many real-world scenarios. Our method, on the contrary, focuses on passive grammatical

inference, as explored by [15] and [16], to learn the task structure from static datasets. This makes our hybrid approach applicable in environments with implicit task structures that must be inferred without active querying.

Contribution. In this paper, we introduce the Hybrid Passive Grammatical Inference and Online Planning (HiPO) algorithm for planning with non-Markovian tasks. HiPO employs a passive grammatical inference approach, leveraging the RPNI algorithm to learn a DFA from a pre-collected, fixed dataset of positive and negative trajectories gathered via an offline behavior policy, eliminating the need for additional online interactions. By utilizing this passive method, we significantly reduce the reliance on costly real-time interactions with the environment, resulting in substantial computational savings. Following this, we learn an optimal policy by planning over a product MDP, which integrates the non-Markovian environment with the learned DFA. Our experimental results demonstrate that HiPO achieves faster convergence compared to existing active learning algorithms, effectively reducing both training time and interaction costs.

II. Background and Notation

In this paper, we assume a non-Markovian reward decision process (NMRDP) for the agent-environment dynamics, i.e., the reward function depends on the entire history of the agent's interaction with the environment and not just the current state. The learning algorithm is assumed to have access to this NMRDP model, except for the reward function R.

Definition 1 (Non-Markovian Reward Decision Process): An NMRDP is defined as a tuple $\mathcal{M} = (S, s_0, A, P, \mathsf{AP}, L, R)$, where S is a finite set of states, $s_0 \in S$ is the initial state, A is a finite set of actions, $P : S \times A \times S \to [0, 1]$ is a probabilistic transition function, AP is a finite set of atomic propositions, $L : S \to 2^{\mathsf{AP}} \cup \{\varepsilon\}$ is a labeling function, and $R : S^* \to \{0, 1\}$ is a non-Markovian reward function.

The labeling function $L : S \to 2^{\mathsf{AP}} \cup \{\varepsilon\}$ maps states to the alphabet $\Sigma = 2^{\mathsf{AP}}$, which specifies the language corresponding to the underlying goal. This alphabet represents meaningful events that occur in the states. The empty string ε is included to label states that are semantically insignificant, to be ignored during the process of grammatical inference. The non-Markovian reward function R assigns a reward of 1 to a trace if it is part of the DFA's language. A sequence of states $s_1, s_2, \ldots, s_k$ generates a trace $L(s_1)L(s_2)\ldots L(s_k)$.

Definition 2 (Deterministic Finite Automaton): A DFA is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of automaton states, Σ is a finite alphabet of task-related labels, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of accepting states.

The task sequence that the agent must complete is modeled using a DFA, which captures valid sequences of subtasks to be achieved (e.g., items to be collected in a given order). The DFA tracks the agent's progress toward completing tasks by transitioning between automaton states based on interactions with labeled states. A trace is in the language of the DFA if it leads to a reward of 1.

The combination of the environment $\mathcal{M}$ and the task specification captured by the DFA $\mathcal{A}$ results in a product MDP $\mathcal{M}^{\otimes} = \mathcal{M} \otimes \mathcal{A}$, which we define next.

Definition 3 (Product MDP): Given an NMRDP $\mathcal{M} = (S, s_0, A, P, \mathsf{AP}, L, R)$ and a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, their product MDP is given by $\mathcal{M}^{\otimes} = (S^{\otimes}, x_0, A, P^{\otimes}, Q, L^{\otimes}, R)$, where $S^{\otimes} = S \times Q$, the initial state $x_0 = (s_0, q_0)$, $P^{\otimes} : (S \times Q) \times A \times (S \times Q) \to [0, 1]$ is defined in (1), and $L^{\otimes}((s, q)) = \{q\}$.

$$P^{\otimes}((s', q')|(s, q), a) = \begin{cases} P(s'|s, a) & q' = \delta(q, L(s')) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The transition function $T^{\otimes} : S^{\otimes} \times A \to S^{\otimes}$ updates both the environment and automaton states based on the agent's actions and label interactions.

III. Proposed Approach

In this section, we describe our hybrid learning approach HiPO, which combines passive automaton learning using the RPNI algorithm with online policy learning via planning over the product MDP. This hybrid method combines the strengths of passive offline learning with active online adaptation, making it suitable for complex, non-Markovian environments.

A. Automaton Learning Using RPNI

The initial stage of our approach involves learning a DFA from a static dataset consisting of positive and negative task execution traces. Using the RPNI algorithm, we can infer the underlying structure of tasks without needing real-time interaction with the environment, which is convenient in offline learning scenarios.

Algorithm 1 describes the DFA learning process. It takes sets of positive and negative trajectories, P_{traj} and N_{traj}, collected through some behavior policy, and outputs a DFA $\mathcal{A}$ that represents valid task sequences from the pre-collected data. A positive trajectory $p \in P_{traj}$ is a sequence of states $s_0, s_1, \ldots, s_k$ that successfully completes the task (such as collecting items in the correct order), corresponding to a trace of labels $w \in \Sigma^*$, where $w = L(s_0)L(s_1)\ldots L(s_k)$. These traces are split into segments, each representing specific subtasks or labels in the correct order, forming the set of positive examples P, used in the learning process. Negative trajectories N_{traj} represent sequences that fail to meet the tasks, such as incorrect item order. These form the set of negative examples N, ensuring the DFA rejects invalid sequences.

The algorithm proceeds by passively learning a DFA from these positive and negative traces in two main steps. First, a Prefix Tree Acceptor (PTA) is constructed from the set of positive examples P, where each sequence forms a distinct path from the root to a leaf. Initially, the PTA accepts only the sequences in P without any generalization. In the second step, the algorithm iteratively merges similar states in the PTA to generalize the DFA. During this state-merging process, the

algorithm ensures that the resulting DFA rejects all negative examples from N. If a merge would cause the DFA to accept a negative example, that merge is discarded. The algorithm continues this process of merging and validation against negative examples until no further merges are possible, resulting in a minimal DFA $\mathcal{A}$ that accepts all positive examples while rejecting all negative ones.

This approach differs from active learning methods, such as L^*, which rely on interactive membership and equivalence queries. Instead, it operates passively, constructing the DFA solely from the pre-collected dataset of examples. This makes it particularly well-suited for environments where it is impractical or costly to interact directly with the environment to gather traces.

Algorithm 1 Passive DFA Learning in NMRDP

Input: NMRDP $\mathcal{M}$, Positive trajectories P_{traj}, Negative trajectories N_{traj}
Output: Learned DFA $\mathcal{A}$
Step 1: Extract label sequences from trajectories
Extract positive label sequences $P \leftarrow \{\text{EXTRACTLABELSEQUENCE}(\mathcal{M}, traj) \mid traj \in P_{traj}\}$
Extract negative label sequences $N \leftarrow \{\text{EXTRACTLABELSEQUENCE}(\mathcal{M}, traj) \mid traj \in N_{traj}\}$
Step 2: Build Prefix Tree Acceptor (PTA)
Build PTA $\mathcal{T}$ from positive label sequences P
Initialize DFA $\mathcal{A}$ from the PTA $\mathcal{T}$
Step 3: State Merging with Negative Sequences
for each pair of states q_1, q_2 in DFA $\mathcal{A}$ **do**
    if ISCONSISTENT(q_1, q_2, N) **then**
        Merge states q_1 and q_2 in DFA $\mathcal{A}$
    end if
end for
Continue merging states until no further merges are possible
Return the learned DFA $\mathcal{A}$

function EXTRACTLABELSEQUENCE($\mathcal{M}$, $traj$)
    Given $\mathcal{M}$, extract sequence of labels $\ell_1\ell_2\cdots\ell_n$ from trajectory $traj$
    return label sequence
end function

function ISCONSISTENT(q_1, q_2, N)
    Check if merging states q_1 and q_2 contradicts any sequence in N
    return `True` if consistent, `False` otherwise
end function

B. Planning over the Product MDP

After learning the DFA $\mathcal{A}$, it is combined with the environment to form a product MDP, where planning and policy learning are performed as described in Step 3 of Algorithm 2. The product MDP captures both the dynamics of the environment and the task's structural constraints, guiding the agent to follow valid action sequences that lead to task completion.

Here, we use Q-learning over the product MDP, which enables the agent to learn an optimal policy by interacting with the environment and updating its knowledge of the task, represented by the learned DFA, through experience. The agent dynamically updates the Q-values associated with state-action pairs, progressively improving the policy over time. The Q-value update rule is defined as follows:

$$Q(s^{\otimes}, a) \leftarrow Q(s^{\otimes}, a) + \alpha \left[r + \gamma \max_{a'} Q(s'^{\otimes}, a') - Q(s^{\otimes}, a) \right] \tag{2}$$

where, for simplicity, we have used $s^{\otimes} := (s, q)$ and $s'^{\otimes} := (s', q')$, with $q' = \delta(q, L(s'))$, and $Q(s^{\otimes}, a)$ represents the expected cumulative reward for taking action a in state $s^{\otimes} \in S^{\otimes}$, α is the learning rate, γ is a discount factor, and r is the immediate reward received after transitioning to state $s'^{\otimes}$. The Q-values are used by the agent to learn an optimal policy $\pi : S^{\otimes} \to \mathcal{P}(A)$ that maximizes the discounted cumulative reward, where $\mathcal{P}(A)$ is the probability simplex over A.

Algorithm 2 HiPO: Hybrid Passive Inference and Online Planning

Input: NMRDP $\mathcal{M}$, Positive trajectories P_{traj}, Negative trajectories N_{traj}, Number of episodes T
Step 1: Learn DFA from trajectories
Use Algorithm 1 with P_{traj}, N_{traj} to learn DFA $\mathcal{A}$
Step 2: Form Product MDP
Construct product MDP $\mathcal{M}^{\otimes} = \mathcal{M} \otimes \mathcal{A}$
Step 3: Planning over Product MDP
Perform Q-learning on $\mathcal{M}^{\otimes}$ using the update rule (2)
Return the learned policy π

IV. EXPERIMENTAL RESULTS

A. Environment Setup

We conducted experiments in two distinct gridworld environments: the Office Labeled GridWorld and the Minecraft Labeled Environment, depicted in Fig. 1 and 2, respectively. Both environments test the agent's ability to navigate and complete tasks in a specific sequence, modeled as an NMRDP.

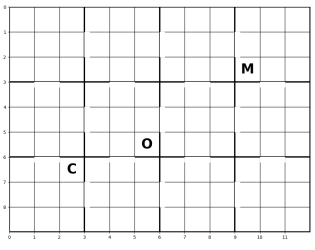

Fig. 1. Office Labeled GridWorld environment.

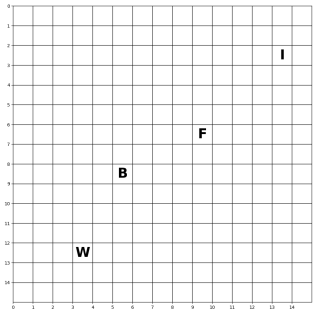

Fig. 2. Minecraft Labeled GridWorld Environment.

1) Office Labeled Gridworld Environment: In this environment, the grid consists of rooms labeled with tasks such as "coffee," "mail," and "office." The agent's goal is to collect these items in the specified order, with rewards dependent on following the correct sequence. The structured sequence of tasks adds temporal dependencies, which traditional Markovian models cannot capture.

2) Minecraft Labeled Gridworld Environment: In this environment, the agent must also collect items in a specific order: wood (`w`), iron (`i`), factory (`f`), and bridge (`b`). As in the Office Labeled GridWorld, rewards are given only when the sequence is completed correctly, and the environment is modeled as an NMRDP. The Minecraft environment is more complex, with four tasks instead of three, and a larger grid (15×15 vs. 9×12), requiring more time and planning for completion.

B. Comparison with Active Grammatical Inference

We compare the performance of HiPO, which leverages passive grammatical inference via RPNI, with the Active Grammatical Inference method AGI [11], JIRP [16], and AFRAI [9]. AGI relies on the L^* algorithm for automata learning, which requires interactive membership and equivalence queries. In contrast, our method uses offline data for DFA learning, reducing the need for real-time interaction with the environment. JIRP jointly infers reward machines and policies by iteratively refining the automaton from RL episodes, updating its hypothesis when inconsistencies arise. This enables simultaneous improvement of both the automaton and the policy. AFRAI employs L^* for active automaton learning, introducing an additional policy to answer membership queries, guiding exploration and improving performance in non-Markovian environments.

Evaluation Metrics: The primary evaluation metric is the accumulated reward over time, which reflects the efficiency of task completion. This captures how quickly the agent learns to complete tasks in the correct sequence across both environments. We also consider the number of samples used. In HiPO, the total sample count includes both the learning and execution phases. During the learning phase, samples correspond to the number of transitions in the positive and negative example trajectories. In the planning phase, the number of samples refers to the total number of training steps across all episodes.

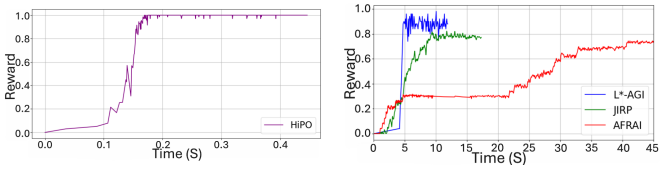

Fig. 3. (Left) Reward versus time for HiPO in the Office. (Right) Reward over time for AGI, JIRP, and AFRAI in the Office.

As shown in Figs. 3 and 4, HiPO achieves higher rewards per unit time and converges faster compared to the other approaches. This improvement stems from the offline learning of the DFA, which reduces the need for real-time interaction with the environment. The results are consistent across both

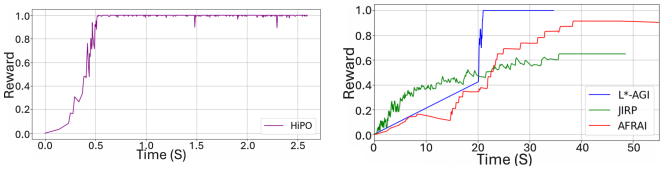

Fig. 4. (Left) Reward versus time for HiPO in Minecraft. (Right) Reward over time for AGI, JIRP, and AFRAI in Minecraft.

the Office and Minecraft environments. The results are plotted separately due to the differences in time scales and to ensure fairness, as the DFA learning phase of HiPO is passive, while the other algorithms rely on active learning.

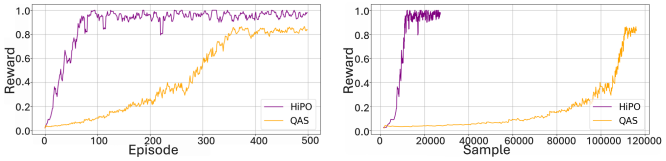

Fig. 5. (Left) Reward per episode for HiPO and QAS in Office. (Right) Samples vs reward for HiPO and QAS in Office.

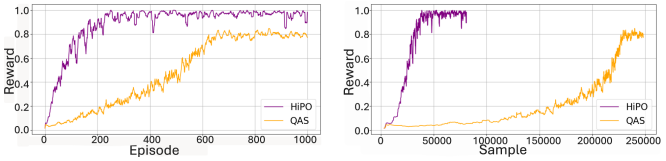

Fig. 6. (Left) Reward per episode for HiPO and QAS in Minecraft. (Right) Samples vs reward for HiPO and QAS in Minecraft.

In Figs. 5 and 6 we also compare against QAS [17] which performs Q-learning in an augmented state space by incorporating an additional binary vector to represent whether each high-level event has been encountered. This augmentation adds complexity by expanding the state space with extra bits of information. HiPO demonstrates significantly better performance in terms of reward per episode and sample complexity. This can be attributed to our structured use of the learning phase, which involves targeted task learning through sequential label achievements, allowing more efficient and robust adaptation to the environment's dynamics.

V. Conclusion

We developed HiPO, a method combining offline automaton learning with online planning in a product MDP. Using RPNI, we learned task structures from static datasets, reducing the need for real-time interaction. This structured knowledge guided planning, enabling the agent to adapt and optimize its policy in real-time. Our evaluation shows that this approach leads to faster convergence and reduced sample complexity compared to methods relying solely on continuous querying, improving learning efficiency by separating task learning from policy learning. HiPO strikes a balance between offline learning and online adaptability, providing an efficient solution for reinforcement learning in non-Markovian settings.

REFERENCES

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[2] Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13(1):227–303, 2000.

[3] 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.

[4] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[5] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Hierarchies of reward machines. In *International Conference on Machine Learning*, pages 10494–10541. PMLR, 2023.

[6] Jose Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern Recognition and Image Analysis: Selected Papers from the IV-th Spanish Symposium*, pages 49–61. World Scientific, 1992.

[7] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[8] Matan Gaon and Ronen Brafman. Reinforcement learning with non-Markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3980–3987, 2020.

[9] Zhe Xu, Bo Wu, Aditya Ojha, Daniel Neider, and Ufuk Topcu. Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In *Machine Learning and Knowledge Extraction, CD-MAKE 2021*, pages 115–135. Springer, 2021.

[10] Taylor Dohmen, Noah Topper, George Atia, Andre Beckus, Ashutosh Trivedi, and Alvaro Velasquez. Inferring probabilistic reward machines from non-markovian reward signals for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 574–582, 2022.

[11] Noah Topper, George Atia, Ashutosh Trivedi, and Alvaro Velasquez. Active grammatical inference for non-Markovian planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 32, pages 647–651, 2022.

[12] Yiyou Sun, Junjie Hu, Wei Cheng, and Haifeng Chen. DFA-RAG: Conversational semantic router for large language model with definite finite automaton. In *Forty-first International Conference on Machine Learning*, 2024.

[13] Lekai Chen, Ashutosh Trivedi, and Alvaro Velasquez. LLMs as probabilistic minimally adequate teachers for dfa learning. *arXiv preprint arXiv:2408.02999*, 2024.

[14] Marcell Vazquez-Chanlatte, Karim Elmaaroufi, Stefan J Witwicki, and Sanjit A Seshia. L^*LM: learning automata from examples using natural language oracles. *arXiv preprint arXiv:2402.07051*, 2024.

[15] Mohammad Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[16] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 590–598, 2020.

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