Learning Policies by Learning Rules

Brandon Araki, Jeana Choi, Lillian Chin, Xiao Li, and Daniela Rus, Fellow, IEEE

Abstract—Efficiently learning interpretable policies for complex tasks from demonstrations is a challenging problem. We present Hierarchical Inference with Logical Options (HILO), a novel learning algorithm that learns to imitate expert demonstrations by learning the rules that the expert is following. The rules are represented as linear temporal logic (LTL) formulas, which are interpretable and capable of encoding complex behaviors. Unlike previous works, which learn rules from high-level propositions, HILO learns rules by taking both propositions and low-level trajectories as input. It does this by defining a Bayesian model over LTL formulas, propositions, and low-level trajectories. The Bayesian model bridges the gap from formula to low-level trajectory by using a planner to find an optimal policy for a given LTL formula. Stochastic variational inference is then used to find a posterior distribution over formulas and policies given expert demonstrations. We show that by learning rules from both propositions and low-level states, HILO outperforms previous work on a rule-learning task and on four planning tasks while needing less data. We also validate HILO in the real world by teaching a robotic arm a complex packing task.

Index Terms—Imitation learning, learning from demonstration, probabilistic inference, task and motion planning.

I. INTRODUCTION

IN THE imitation learning (IL) problem, desired behaviors are learned by imitating expert demonstrations [1]. IL methods have been successful in tackling a wide range of diverse tasks, largely due to model-free policy representations [2]. While these model-free policies have high expressive power, they face significant issues in real-world deployment. These policies are often difficult or impossible to interpret [3], require significant amounts of expert data to train, struggle to learn long-time horizon tasks due to vanishing gradients, and cannot be reconfigured to solve new tasks.

The shortcomings of current IL methods are exemplified when learning complex rule-based tasks. Imagine a chef cooking an old family dish. The robot can observe the kitchen and the chef’s movements, but it cannot observe the recipe that the chef is implicitly following. A model-free IL agent might require many demonstrations to arrive at a robust, high-performing policy. The robot would also struggle to learn the task due to its long-time horizon nature. And even if the robot learns how to cook the dish, it would not be able to modify the task according to a diner’s request.

To address these limitations, many approaches use symbolic reasoning to learn rule-based policies [4]. Symbolic reasoning decomposes complex environments into a collection of symbols and associated subtasks, which can be related via interpretable rules to specify tasks such as cooking a dish or driving a car. Learning a policy by learning rules enables agents to learn long-term plans much more efficiently than by learning a policy over the low-level environment alone. Meanwhile, the subtasks, which have shorter time horizons and a simpler structure than the overall task, are much more amenable to model-free learning algorithms. Therefore, once a rule-based policy is learned, low-level model-free policies can be used to execute the symbolic plan [5].

In most previous IL work that uses symbolic planning, rule learning occurs over sequences of symbols without taking the expert’s low-level trajectories into account. As we demonstrate in Section VI-A, this results in a loss of information that makes rule learning less efficient and in some cases prevents the learning algorithm from converging to the expert’s rules. Previous work does not take low-level trajectories into account because, until recently, there was no planning algorithm that could plan over both the rules and the low-level states in an optimal and efficient manner. However, our recently introduced planning algorithm called Value Iteration with the Logical Options Framework (LOF-VI) can find optimal plans over both the rules and the low-level states extremely efficiently [6]. Incorporating LOF-VI into the learning process could therefore make rule learning more accurate and efficient.

In this work, we address the challenge of enabling robotic agents to learn to imitate complex tasks in an interpretable and efficient manner by introducing Hierarchical Inference with Logical Options (HILO), an IL algorithm that learns a policy by learning the rules that the expert is following. In HILO, the symbols are propositions that define true/false events in the environment, and the rules are expressed as interpretable linear temporal logic (LTL) formulas. Given a low-level environment including propositions; a set of pre-trained low-level policies; and expert demonstrations, HILO learns a distribution over LTL formulas and policies that characterize the task the expert is performing.

HILO improves upon previous work by enabling low-level trajectories to be used as input in the learning process. HILO achieves this by defining a hierarchical Bayesian model that relates LTL formulas to propositions and low-level trajectories.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
The model bridges the gap from formula to low-level trajectory by using the LOF-VI planner to find an optimal policy for a given LTL formula. Therefore the model defines a distribution over LTL formulas, and each formula can be transformed by the LOF-VI planner into a policy that defines a distribution over trajectories of propositions and low-level states. Stochastic variational inference is used to find a posterior distribution over formulas and policies given expert demonstrations. We show that HILO achieves better performance with less data than previous work, and we demonstrate HILO’s utility in a real-world packing task.

In summary, this paper makes the following contributions:

1) We introduce a hierarchical Bayesian model that incorporates the LOF-VI planner to relate LTL formulas to policies, thereby defining a joint distribution over LTL formulas, propositions, and low-level trajectories.

2) We use the Bayesian model to define a stochastic variational inference problem that infers a posterior distribution over interpretable LTL formulas and policies given a set of expert demonstrations.

3) We evaluate the rule-learning performance of HILO on a set of 100 different tasks, and we evaluate HILO’s policy-learning performance on 4 planning domains, showing that HILO’s planner-in-the-loop allows it to outperform other rule-learning and policy-learning algorithms while requiring less data. We also validate HILO in a real-world setting by teaching a robotic arm a complex grocery-packing task.

II. RELATED WORK

Most rule-learning algorithms take sequences of propositions (binary variables) as input and search over the space of automata/LTL formulas to find the set of rules most likely to generate or accept the observed data. [7] defines likelihood functions over proposition sequences for inferring LTL specifications from demonstrations. [8] uses discrete optimization via Tabu search to learn rules represented as Reward Machines. [9] records the trajectories of an agent and semantically segments the environment to produce proposition traces. They search for automata using DPLL search. [10] learns an LTL formula and proposition mapping from demonstrations. Their approach relies on counterexample generation and testing. There is also a large literature beyond the IL setting on learning automata and LTL formulas from proposition traces. HILO differs from much of this literature in that it does not assume access to counterexamples or an oracle, and it assumes that the expert demonstrations consist of only positive examples [11], although there are also many works that construct automata from positive traces [12].

The main distinction between HILO and the previous methods is that the other methods infer automata or formulas over only traces of propositions, whereas HILO includes a planner-in-the-loop that allows HILO to find an optimal policy for a given LTL formula and evaluate the likelihood of the expert trajectories with respect to the policy using both low-level trajectories and propositions. HILO therefore takes advantage of nuanced environmental data to improve inference (see Section VI-A).

The work in this paper extends our prior work [13], which also includes a planner in the inference loop. However, the planner in [13], called Logical Value Iteration (LVI) [14], is limited to planning over discrete state and action spaces and is not composable. The work in this paper incorporates a new planner introduced in [6] called Value Iteration with the Logical Options Framework (LOF-VI), which learns composable policies and allows our method to be applied to environments with continuous state and action spaces. [13] also learns rules as probabilistic automata, whereas our method learns rules as LTL formulas. The distribution over LTL formulas is a more natural representation of task space than the distribution over probabilistic automata, meaning that our algorithm requires significantly less data to converge to a good solution (Section VI-C).

There is also a large body of work that learns symbolic groundings, along with policies, from a set of expert demonstrations [4], [15], [16]. However, these works focus on the problem of representation learning, as it is very difficult to learn interpretable representations. Our work assumes that interpretable propositions are provided, either by a human user or a representation learning algorithm, and then uses the given propositions to learn LTL formulas and policies.

III. PRELIMINARIES

Linear Temporal Logic: We formally specify tasks with LTL, which expresses tasks and rules using temporal operators such as “eventually” and “always” [17]. Formulas \( \phi \) have the syntax grammar:

\[
\phi := p \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid X \phi \mid \phi_1 \land \phi_2
\]

where \( p \) is a proposition (a boolean-valued truth statement that corresponds to objects or events in the environment), \( \neg \) is negation, \( \lor \) is disjunction, \( X \) is “next,” and \( \land \) is “until”. Derived rules include conjunction \( (\land) \), “eventually” \( (F \phi \equiv \text{true} \lor \phi) \) and “always” \( (G \phi \equiv \neg F \neg \phi) \) [17]. \( \phi_1 \land \phi_2 \) means that \( \phi_1 \) is true until \( \phi_2 \) is true. \( F \phi \) means that there is a time where \( \phi \) is true and \( G \phi \) means that \( \phi \) is always true. LTL formulas can be translated into Büchi automata using translation tools such as SPOT [18]. In this work, we only consider formulas that have a finite state automaton (FSA) representation, as the LOF-VI planner assumes that the rules are expressed as an FSA. This restriction means that HILO cannot learn never-ending tasks such as persistent surveillance tasks; however, all finite tasks can be learned.

The Logical Options Framework: The Logical Options Framework (LOF) is a framework for formulating and planning over rule-based environments [6]. HILO uses LOF to find optimal policies for LTL-specified tasks (see the red box in Fig. 2). In LOF, propositions are divided into three types: subgoals \( \mathcal{P}_G \), events \( \mathcal{P}_E \), and safety propositions \( \mathcal{P}_S \). Subgoals must be achieved in order to satisfy the task. They are associated with goals such as “the agent has picked up the banana”. Each subgoal is associated with a logical option, which is a subpolicy whose purpose is to achieve the subgoal. Events affect the control flow of the task specification, e.g., whether or not a goal is cancelled.
Safety propositions are events that the agent must avoid, e.g., hitting an obstacle.

The LOF algorithm has two steps. Step 1: is to learn the logical options. Each option is associated with a subgoal and must be trained to achieve its subgoal. Training an option involves learning a subpolicy \( \pi_o : S \to A \), a reward model \( R_o : S \to \mathbb{R} \), and a transition model \( T_o : S \times S \to [0, 1] \). The reward model gives the expected return of initiating an option at a given state \( s \). and the transition model gives the probability of the option ending at a state \( s' \) given it is initiated at state \( s \). In practice, model-free policy-learning algorithms that learn both a policy and a value function, such as value iteration or PPO, can be used to learn the options, as the value function is equivalent to the reward model \( R_o \). The transition model can be simplified by assuming that the option will always achieve its subgoal [6]. Because logical options can be trained offline using generic learning algorithms, HILO takes trained logical options as inputs.

Step 2: is to find an optimal policy over the rules. The LTL formula is translated into an FSA using SPOT [18]. The FSA, the low-level environment, and the logical options form a construct called a semi-Markov decision process (SMDP) [19]. SMDPs differ from regular MDPs in that actions are replaced by options, which can act over multiple time steps. A Q-function can be found over the SMDP using a form of value iteration called Value Iteration with the Logical Options Framework (LOF-VI). The learned Q-function \( Q : F \times S \times O \to [0, 1] \) specifies a distribution over FSA states, low-level states, and logical options, thus linking the LTL formula to the low-level trajectory in HILO’s Bayesian model. LOF-VI can operate on continuous state/action spaces because it only has to plan over the potential start and stop states of the options, which form a finite set of states. As shown in [6], LOF-VI typically takes only 10–50 training iterations to converge to the optimal policy for a given LTL formula, making it an ideal planner-in-the-loop for inference.

IV. PROBLEM STATEMENT

Given: HILO takes a low-level environment MDP \( \mathcal{E} \), options \( \mathcal{O} \), and expert trajectories \( \mathcal{D} \) as input. The MDP \( \mathcal{E} = (S, A, P, T_E \times T_F, R_E, \gamma) \). The state space \( S \) and the action space \( A \) can be discrete or continuous. \( T_E \) is the transition function, \( R_E \) is the reward function, and \( \gamma \) is the discount factor. In addition to the normal components of an MDP, \( \mathcal{E} \) also includes information about the propositions, \( \mathcal{P} \) is the set of propositions and is divided into subgoals \( \mathcal{P}_G \), event propositions \( \mathcal{P}_E \), and safety propositions \( \mathcal{P}_S \). \( T_F \) is a proposition labeling function that relates each state to the propositions that are true at that state. Each subgoal \( p \in \mathcal{P}_G \) is associated with a logical option \( o_p = (\pi_o, R_o, \gamma) \), the sub-policy trained to achieve the associated subgoal; \( R_o \) is the reward model, and \( \gamma \) is the transition model. The set of options is \( \mathcal{O} \). Lastly, HILO assumes access to a set of \( N \) expert trajectories \( \mathcal{D} = \{d_0, \ldots, d_N\} \). Each trajectory is a sequence of states, options, and propositions of length \( T_i \) so that \( d_i = (s_{i0}^t, o_{i0}, p_{i0}) \ldots (s_{iT_i}^t, o_{iT_i}, p_{iT_i}) \). Note that the trajectory is not over the entire sequence of low-level states and actions but rather over the sequence of high-level options that the expert performs. Also included in the trajectory at each time step are the agent’s low-level state and the proposition that is true when option \( o_t \) is initiated.

Problem 1: Given an environment MDP \( \mathcal{E} \), a set of options \( \mathcal{O} \), and a dataset of \( N \) trajectories \( \mathcal{D} \), learn a distribution over LTL formulas \( q(\phi | \mathcal{D}, \alpha) \) parametrized by \( \alpha \) that induces a distribution over policies that best imitates the expert demonstrations by minimizing the KL divergence between the true posterior over formulas \( p(\phi | \mathcal{D}, \alpha) \) and the variational approximation \( q(\phi | \mathcal{D}, \alpha) \).

V. HIERARCHICAL INFERENCE WITH LOGICAL OPTIONS

HILO solves Problem 1 by formulating a Bayesian model that describes the problem and then using stochastic variational inference (SVI) to find an approximate posterior distribution over LTL formulas given the data (Algorithm 1).

A. Bayesian Model

The Bayesian model is a hierarchical hidden Markov model (HMM). In the high level of the HMM, an LTL formula \( \phi \) is sampled given parameters \( \alpha \), subgoals \( \mathcal{P}_G \), and proposition sequences \( PS \) (discussed in Section V-B). The LTL formula is converted using SPOT into an FSA with transitions \( T_F \) and a reward function \( R_F \). \( R_F \) assigns a reward of \(-1\) to all FSA states except the goal state, which has a reward of \( 0 \). Given \( T_F \), \( R_F \), environment \( \mathcal{E} \), and the initial state of the agent \((s_0, o_0, p_0)\), LOF-VI can be used to find an optimal policy. The policy can then be rolled out, giving a sequence of states, options, propositions, and FSA states \((s_t, o_t, p_t, f_t)\). \((s_t, o_t, p_t)\) corresponds to the observed expert trajectories, and the FSA state \( f_t \) is the hidden state of the HMM. We assume that every trajectory begins

---

**Algorithm 1: HILO Algorithm.**

1. **procedure** HILO
2. \( \text{Training Inputs: MDP } \mathcal{E}, \text{ options } \mathcal{O}, \text{ trajectories } \mathcal{D} \)
3. \( \text{To learn: Variational parameter } \alpha^* \)
4. \( \text{Stochastic variational inference on the learning objective } L \) (Section V-A)
5. \( \text{return } \alpha^* = \arg\min_\alpha L \)
6. **end procedure**
HILO Diagram: The inputs to HILO are a low-level environment, a set of trained options, and expert data (trajectories and proposition sequences). The output is a distribution over LTL formulas that approximates the true posterior distribution.

The generative model: (a) A set of rule templates is used to generate LTL formulas. (b) An example of a generated parse tree for \( \phi \). (c) The sampled LTL formula is converted to an FSA. (d) LOF-VI finds a satisfying policy over the FSA and the environment.

The novelty of this distribution is that it is over both \( \phi \) and the variational distribution \( q(\phi|D, \alpha) \):

\[
\mathcal{L} = KL(q(\phi|D, \alpha)||p(\phi|D, \alpha)), \quad \hat{\alpha}^* = \arg \min_\alpha \mathcal{L}
\]

\( \hat{\alpha}^* \) defines an approximate posterior distribution over \( \phi \). The SVI problem was solved using Pyro [20] and Pytorch.

B. Sampling LTL Formulas

Since LTL is a regular language, LTL formulas can be expressed as trees. Fig. 3(a) shows the formal grammar we use to generate trees, and Fig. 3(b) shows the tree associated with the formula \( F((a \lor d) \land P_c) \). For a given tree depth, the space of possible trees grows exponentially with the number of propositions and operators, and trees can have an infinite depth. The problem of finding the optimal LTL formula in this vast space is therefore very challenging. We use a number of heuristics and assumptions to guide inference and reduce the search space while maintaining an expressive grammar. We include pseudo-code in the supplement that describes our model in detail; we give an outline here.

Our grammar constructs formulas from rule templates to reduce the search space. We use six templates: Or and Eventually correspond to the \( \lor \) and \( \mathsf{F} \) operators. EColl and SColl are collections of event and subgoal propositions, respectively. They work by sampling the number of propositions to sample, then sampling the given number of propositions from a list of available propositions and joining them as a disjunction.
templates in order to reduce the search space. The prior is a recursive distribution for the variational approximation.

defines prior values for all possible sampling operations, and is the set of subgoals used by Sequential

can distinguish between the two formulas with this single data point. HILO also cannot distinguish between the two formulas when a is closer to the agent than b, since the behavior of going to a instead of b when a is closer is consistent with both Fa and F(a ∨ b). However, when a second data point is added where the agent goes to a even though b is closer, HILO converges to Fa as the most likely formula, because the behavior of going to a even though b is closer is consistent only with Fa. NOLO, by contrast, cannot take advantage of this nuanced trajectory information. This is a fundamental limitation that NOLO shares with almost all other prior work. In the next section, we show that HILO has superior rule-learning performance not just in this simple case but on 100 other rules as well.

A. Inference Over Low-Level States

HILO differs from other rule-learning methods in that it takes not just proposition sequences as input but also low-level trajectories. We compare HILO to a baseline that only takes proposition sequences as input. The baseline, which we call NOLO for “No Logical Options,” has the same fundamental structure of the prior works discussed in Section II, inferring the formulas that best match the input sequences without taking the low-level states into account. NOLO has the same architecture as HILO as outlined in Fig. 2, except that most of the elements in the red box have been removed. NOLO does not take the environment E nor the options C as inputs and does not use LOF-VI to find an optimal policy. Instead, it proposes LTL formulas, converts them into FSAs, and evaluates the likelihood of the expert proposition sequences (not trajectories) based on their likelihood of being accepted by the proposed FSA.

Fig. 5 illustrates how HILO uses low-level trajectory data to its advantage. In this example, the environment has two subgoals, a and b, and the expert is following the rule Fa, “go to a,” which produces the proposition sequence [a]. NOLO and most other prior work can only use the proposition sequence [a] as input. Given this single data point, it is impossible to distinguish between the formulas F(a ∨ b) and Fa because the input [a] satisfies both of them. As shown in Fig. 5, NOLO cannot distinguish between the two formulas with this single data point.

B. Rule-Learning & Interpretability

We next demonstrate HILO’s ability to learn interpretable LTL formulas from low-level trajectories. We defined a grid-world with four subgoal propositions a, b, c, and d, and event proposition f. We generated 100 random formulas of string-length less than 70 (listed in the supplement). They vary in complexity from as simple as Fb to as complex as (¬f ∧ F (a ∨ d) ∧ F ((b ∨ c) ∧ Fd)) ∨ (Ff ∧ F (a ∨ b) ∧ F (c ∨ d))). For each formula, we generated 10 environments with randomly placed subgoals and obstacles (such as Fig. 6(a)), along with simulated expert demonstrations. We ran HILO, NOLO, HILO without the LTL prior, and NOLO without the LTL prior, for 100 training steps on an increasing number of samples (1, 2, 4, 6, 8, and 10 samples). Fig. 6(b) shows the average posterior probability of the true formulas as the sample size increases. HILO becomes much more certain than NOLO as it is given more data. The reason the posterior probability increases with the number of samples.

VI. EXPERIMENTS & RESULTS

We performed four experiments to show that HILO is an effective tool for learning and planning with LTL formulas. The first illustrates how HILO improves over prior work by incorporating planning into inference. The second demonstrates HILO’s ability to learn rules from low-level trajectories. The third shows that LOF can learn successful policies using a fraction of the data of other methods. The final applies HILO to a packing task on a real-world system.
true formula is $F(a)$ or $F(a \lor b)$. With only one data point where $a$ is closer to the agent than $b$, HILO also cannot infer which formula is the true formula. When HILO sees that the agent chooses to go to $a$ even though $b$ is closer, it converges to the true formula $F(a)$.

Fig. 6. The performance of HILO vs NOLO over 100 formulas, with and without the LTL prior. (a) A randomly generated gridworld environment. (b) The average posterior probability of the true formulas over number of samples. (c) The proportion of experiments where the true formula is in the top 5 most likely posterior formulas.

is because many of the formulas have disjunctions, so there are many trajectories that can satisfy them. More samples enable both HILO and NOLO to better infer the disjunctions. Fig. 6(c) shows the proportion of experiments for which the true formula is in the top five most likely posterior formulas. HILO performs slightly better than NOLO up until 10 samples, where HILO is slightly lower. We believe that for some of the more complex formulas, 10 samples is not enough to cover all of the possible satisfying sequences of propositions. Therefore, HILO becomes more certain about overly simple versions of the true formula, so that the true formula does not make the top five. Both HILO and NOLO without the LTL prior perform very poorly. This is likely because of the size of the search space over LTL formulas; without a strong prior to guide inference, neither model is able to converge to the true formula. Our results show that the LTL prior is crucial for learning, and that the strength of HILO is that it can take advantage of the environment’s low-level information to arrive at a more precise posterior distribution and learn more complex formulas.

Since LTL formulas are interpretable, the rules that HILO learns can be directly converted to unambiguous natural language. Here is an example of a formula that our model learned in the experiments: ($G \neg f \land F(a \land Fd) \lor (F \land F(d \land F((b \lor c) \land Fa)))$). These rules are interpretable models of the high-level behavior of the learned policies.

### C. Planning

We tested HILO on four planning domains, which were defined as discrete 2-D gridworlds with eight actions defined as moving one grid space in one of the cardinal or ordinal directions. The first domain is a **gridworld** (Fig. 6(a)) with true formula $F(a \land F(b \land Fc))$ (“go to $a$, then $b$, then $c$”). The second is a **lunchbox** packing domain (Fig. 7(a)) that represents a robot arm packing a lunchbox. The true formula is $F((a \lor b) \land F(d \land F(c \land Fd)))$; “pick up $a$ or $b$ and pack it into lunchbox $d$, and then pick up $c$ and pack it into $d$.” The third is a **cabinet** opening domain (Fig. 7(b)). This domain represents the process of opening a cabinet: checking the cabinet ($cc$), seeing if it is unlocked ($uo$) or locked ($nuo$), and then, if it is unlocked, opening it ($op$), and if it is locked, getting the key ($gk$), unlocking it ($uc$), and then opening it ($op$). The formula is $Fcc \land F((uo \land Fop) \lor (nuo \land F(gk \land F((uc \land Fop))))). Lastly, the **dungeon** domain (Fig. 7(c)) is a gridworld with four doors $da$, $db$, $dc$, $dd$, four keys that unlock their corresponding doors $ka$, $kb$, $kc$, $kd$, and a goal $g$. The formula is $Fg \land (\neg da U ka) \land (\neg db U kb) \land (\neg dc U kb) \land (\neg dd U kb)$. These domains are also used in [13] and [14] to evaluate the algorithms introduced in those papers, giving us baselines to compare against.
be applied to more sophisticated tasks, templates such as $\text{Xor}$ or $\text{Until}$ could easily be defined.

We compare HILO’s planning performance to two baselines. The first baseline, Logical Value Iteration (LVI) [13], differs in two ways from HILO: 1) It is limited to use on discrete state and action spaces. 2) LVI infers a probabilistic automaton, whereas HILO infers an LTL formula. The second baseline is an LSTM network, which is a model-free method for dealing with time-series data. The first layer of the network is a 3D CNN with 1024 channels. The second layer is an LSTM with 1024 hidden units. The LSTM baseline takes as input trajectories in which each timestep is an image of the gridworld, including the agent. The LSTM is trained to output the action that the agent takes at each timestep. For each domain, random training and test data points were generated and used for training/evaluation.

The results present two takeaways (Table I). First, HILO requires far fewer data points than LVI and LSTM due to its model-based approach. Since LSTM can only interpolate between trajectories, it needs a significant amount of data to achieve good performance. Although LVI is model-based like HILO and could theoretically require far less data, in practice, high variance in its gradient estimates means that more data must be used for its model to converge [13]. Another reason why HILO has better performance than LVI is because HILO represents task space using LTL formulas rather than probabilistic automata like LVI. A distribution over probabilistic automata is a poor representation of task space because the distribution is over the edge weights of a single automaton, whereas a distribution over LTL formulas is a true distribution over discrete tasks. HILO is therefore able to evaluate the posterior probabilities of individual tasks, whereas LVI can only evaluate the posterior probabilities of the edges of an automaton. The second takeaway of the results is that HILO and LVI inference both outperform the model-free LSTM baseline. In particular, increasing the complexity of the formula does not degrade HILO’s performance, whereas it degrades the LSTM baseline’s performance. This is because unstructured deep models struggle with long-term planning tasks, since they do not have a structured memory or ability to plan, unlike HILO and LVI.

### D. Real-World Packing

To demonstrate the viability of HILO in a real-world application, we implemented HILO on a robotic arm and taught it a complex packing task (Fig. 1). The arm was a UR5 robot with a continuous 3-D state space $S \in \mathbb{R}^3$ and 12-D action space $A \in [0, 2\pi]^6 \times \mathbb{R}^6$, representing the angles and angular velocities of the six joints of the robot. The arm was equipped with a soft gripper [21] and was located at a packing station with a conveyor belt.

The task of the robot was to pack groceries from the moving conveyor belt. Specifically, the task is that if there is an item on the conveyor belt (oob for “object on belt” and oobb for “objects on belt and in buffer”), the robot should pick it up (pick-from-belt). Each item has an April Tag that identifies if it is delicate or not; if the item is delicate (del), the robot should place it in the buffer zone (place-in-buffer). Otherwise, the robot should pack it in the bin (place-in-bin). If there are no items on the conveyor belt but there are items in the buffer (oib for “object in buffer”), the robot should pack any items in the buffer zone (pick-from-buffer) into the bin (place-in-bin). After any of these branches of the program are finished, the robot should return to its home position (h). These rules ensure that non-delicate items will be packed first, and delicate items will be set aside and packed only after all of the non-delicate items have been packed.

The LTL formula is $F((\text{oib} \land F(\text{pick-from-buffer} \land F(\text{place-in-bin} \land \text{Fh}))) \lor ((\text{oob} \lor \text{oobb}) \land F(\text{pick-from-belt} \land F((\text{place-in-bin} \land \text{Fh}) \lor (\text{del} \land F((\text{place-in-buffer} \land \text{Fh}))))))).$ There are four event propositions: oob, oobb, oib, and del. There are five subgoals: pick-from-belt, pick-from-buffer, place-in-buffer, place-in-bin, and h. Options were calculated using RRT Connect, and costs were defined as the length of a path in Euclidean space.

---

**Table I**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Training HILO</th>
<th>Training LVI</th>
<th>Training LSTM</th>
<th>Test HILO</th>
<th>Test LVI</th>
<th>Test LSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gridworld</td>
<td>Set size: 1</td>
<td>600</td>
<td>1000</td>
<td>1200</td>
<td>1200</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Success Rate</td>
<td>100.0%</td>
<td>99.50%</td>
<td>99.20%</td>
<td>99.91%</td>
<td>99.33%</td>
</tr>
<tr>
<td>Lunchbox</td>
<td>Set size: 2</td>
<td>500</td>
<td>9000</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td>Success Rate</td>
<td>100.0%</td>
<td>99.60%</td>
<td>91.44%</td>
<td>100.0%</td>
<td>99.94%</td>
</tr>
<tr>
<td>Cabinet</td>
<td>Set size: 2</td>
<td>550</td>
<td>6000</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td>Success Rate</td>
<td>100.0%</td>
<td>93.60%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>89.58%</td>
</tr>
<tr>
<td>Dungeon</td>
<td>Set size: 1</td>
<td>800</td>
<td>6000</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td>Success Rate</td>
<td>100.0%</td>
<td>0.00%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
This formula is fairly complex as it involves three sequential paths initiated by different event propositions and joined by disjunctions. However, HILO was able to learn this formula after observing just five demonstrations. The first demonstration was picking up a soup can (non-delicate) from the conveyor belt and packing it in the bin. The second demonstration was the same except there was an item in the buffer. The third demonstration was picking up grapes (delicate) from the belt and placing them in the buffer. The fourth demonstration was the same except that there was another item already in the buffer. The last demonstration was packing the grapes from the buffer into the bin when there were no items on the conveyor belt.

After these five demonstrations, HILO inferred a distribution over rules where the most likely rule corresponded exactly to the true rule. The resulting policy was used to pack three bins of groceries consisting of delicate and non-delicate items, where the non-delicate items were packed first and the delicate items were stored in the buffer area and packed later. A video in the supplement shows the experiments.

VII. CONCLUSION & FUTURE WORK

In this work, we introduced HILO, a method for inferring and planning with LTL formulas given low-level trajectory demonstrations. We showed how HILO improves over other work by incorporating planning in the inference loop. Next, we demonstrated that HILO can learn a wide variety of interpretable LTL formulas. In the planning experiments, we showed that HILO needs very little data to learn a high-level policy, and that its performance does not degrade as the task becomes more complex. Finally, we showed how HILO’s strengths – its interpretability and data-efficiency – make it applicable to real-world settings. One of HILO’s main shortcomings is that the space of LTL formulas increases exponentially with the number of propositions and allowed operators. As the number of LTL templates used by HILO increases, the LTL search space increases exponentially. The hand-crafted LTL prior will also become increasingly burdensome to create. Future work could use natural language instructions as a prior by learning a mapping from natural language to LTL prior parameters $\alpha$. Another avenue of future work could be to learn an embedding of LTL formulas over a large dataset of tasks, so that inference could occur over a small latent space.

ACKNOWLEDGMENT

Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors, and not of the funding agencies.

REFERENCES