Learning to Plan with Logical Automata

Brandon Araki\textsuperscript{1, *}, Kiran Vodrahalli\textsuperscript{2, *}, Thomas Leech\textsuperscript{1, 3}, Cristian-Ioan Vasile\textsuperscript{1}, Mark Donahue\textsuperscript{3}, Daniela Rus\textsuperscript{1}

\textsuperscript{1}MIT CSAIL, Cambridge, MA 02139, \textsuperscript{2}Columbia University, New York City, NY 10027
\textsuperscript{3}MIT Lincoln Laboratory, Lexington, MA 02421, *Authors contributed equally

Abstract—This paper introduces the Logic-based Value Iteration Network (LVIN) framework, which combines imitation learning and logical automata to enable agents to learn complex behaviors from demonstrations. We address two problems with learning from expert knowledge: (1) how to generalize learned policies for a task to larger classes of tasks, and (2) how to account for erroneous demonstrations. Our LVIN model solves finite gridworld environments by instantiating a recurrent, convolutional neural network as a value iteration procedure over a learned Markov Decision Process (MDP) that factors into two MDPs: a small finite state automaton (FSA) corresponding to logical rules, and a larger MDP corresponding to motions in the environment. The parameters of LVIN (value function, reward map, FSA transitions, large MDP transitions) are approximately learned from expert trajectories. Since the model represents the learned rules as an FSA, the model is interpretable; since the FSA is integrated into planning, the behavior of the agent can be manipulated by modifying the FSA transitions. We demonstrate these abilities in several domains of interest, including a lunchbox-packing manipulation task and a driving domain.

I. INTRODUCTION

In the imitation learning (IL) problem, desired behaviors are learned by imitating expert demonstrations [1, 11, 35]. IL has had success in tackling tasks as diverse as camera control, speech imitation, and self-driving for cars [42, 10, 19, 46]. However, an IL model trained to imitate a specific task must be re-trained on new expert data to learn a new task. Additionally, in order for a robot to correctly learn a task, the expert demonstrations must be of high quality: most imitation learning methods assume that experts do not make mistakes. Therefore, we ask

1) How can expert demonstrations for a single task generalize to much larger classes of tasks?

2) What if the experts are unreliable and err?

This paper provides answers to these questions by applying elements of formal logic to the learning setting. We require our policies to be derived from learned Markov Decision Processes (MDP), a standard model for sequential decision making and planning [5, 38]. We assume these MDPs can be factored into a large MDP that describes the motion of the robot in the physical environment, and, more importantly, a small finite state automaton (FSA) that corresponds to the rules the agent follows. After learning the transition and reward functions of the MDP and FSA, it is possible to manually change the FSA transitions to make the agent perform new tasks and to correct expert errors. Additionally, the FSA provides a compact symbolic representation of the policy.

For example, imagine the robotic arm in Fig. 1 packing first a sandwich and then a banana into a lunchbox. The physical environment and the motions of the robotic arm can be described by a “low-level” MDP. The rules the robot follows are described using FSAs. In the FSA, transitions are dependent on logical truth statements called propositions. In this environment there are three propositions – “robot has grasped sandwich”, “robot has grasped banana”, and “robot has dropped whatever it is holding into the lunchbox”. The truth values of these propositions control transitions between the FSA’s states, which we also refer to as logic states. For example, when “robot has grasped sandwich” is true, the FSA transitions from being in an initial state to being in a state in which “the robot has grasped the sandwich.” When it is in this new state and “robot has dropped whatever it is holding into the lunchbox” is true, it transitions to the next state, “the robot has placed the sandwich into the lunchbox.” We assume that the propositions correspond to locations in 2D space (e.g., we assume that the manipulator has a pre-programmed behavior to grasp a banana when it is in the vicinity of the banana and “robot has grasped banana” becomes true). This assumption enables us to factor the unknown MDP as the product of the high-level FSA and the low-level MDP. A simpler example of a product MDP is illustrated in Fig. 2.

The agent then learns approximate transitions and rewards associated with this product MDP, and generates a policy by running a planning algorithm. This approach has two benefits: 1) the learned policy is interpretable in the sense of learned FSA representations of rules, and 2) the behavior of the agent is manipulable because the rules that the agent follows can be changed in a predictable way by modifying the FSA’s transitions. These benefits address the questions posed before: performing new tasks without re-learning and correcting faulty behavior.
A. Outline of Our Approach

![Diagram of MDP and FSA integration](image)

Fig. 2: An illustration of how an MDP and an FSA create a product MDP. The MDP is a 2D gridworld with propositions \(a\), \(b\), and \(o\). The FSA describes the rules "go to \(a\), then \(b\), and avoid \(o\). The resulting product MDP represents how these rules interface with the 2D gridworld.

Planning over Approximate MDPs Our model assumes that the MDP of the robot’s behavior, also called a product automaton (PA), factors into a small, high-level FSA and a large, low-level MDP. When the FSA and MDP are known, one can find the optimal policy over the PA with standard planning methods [4, 26, 16, 40]. The model in this paper extends this approach to the IL setting where the transition and reward functions of the PA are unknown by learning an approximate MDP and then planning over the resulting MDP model. Exclusively during training we assume a logic oracle that the agent can query to learn its current FSA state. This assumption is more plausible and efficient to simulate compared to related works, which require knowledge of the full FSA [32, 45]. Our model, the Logic-based Value Iteration Network (LVIN), learns the relevant part of the transition matrix (TM) describing the FSA and directly integrates it into a differentiable recursive planning algorithm generalizing the Value Iteration Network (VIN) proposed in [41]. The key idea is to add a VIN module at each state of the FSA and link them together appropriately.

Logic Formalism In the robotics and control community, temporal logic languages such as Linear Temporal Logic (LTL) are used to unambiguously specify complex tasks, and a large and versatile class of these specifications can be directly translated into FSAs [22, 43, 6, 24, 25, 44, 30, 36]. In this paper, we assume that the high-level FSA is generated by an unknown LTL specification, although our methodology generalizes to any formal grammar that specifies FSAs. In fact, our model does not require that the expert demonstrations be generated from an FSA — it simply finds the best explanation that can be expressed as a product of an FSA and an MDP.

Experiments In the experiments, we demonstrate that for several gridworld tasks of varying complexity and a robot picking task (Fig. 1), our methodology allows us to efficiently understand and modify a robot’s behavior. Our approach also solves tasks requiring long sequences of accurate actions, where we demonstrate standard learning approaches often fail. As another application, we fix expert mistakes without re-training on new data.

B. Contributions

1) We improve learning by attaching a logic oracle to the environment during training in the form of an FSA. Our logic oracle is required to produce less information than previous work (only FSA state, not the full transition matrix) and is thus more feasible to implement.

2) We introduce a differentiable planning model called the Logic-based Value Iteration Network (LVIN) which integrates an FSA into the recursive planning step of VIN. Using an imitation learning objective, we report considerable improvements over baselines in four different domains, including a robot picking task with real-world experiments.

3) We show that our framework can learn the transition matrix between FSA states, thus allowing us to interpret the logic rules that the model has learned.

4) We show how the learned transition matrix can be modified to manipulate the behavior of the agent to reliably perform other desired tasks without further learning. As a result, we can generalize to new tasks and fix the mistakes of unreliable experts without additional expert demonstrations.

II. Related Work

Logic-based Approaches Some recent work uses logical structure to make imitation learning and reinforcement learning (RL) problems easier. [32] uses LTL to define constraints on a Monte Carlo Tree Search. [28] and [18] use the product of an LTL-derived FSA with an MDP to make learning more efficient. In [21] the authors use LTL to design a sub-task extraction procedure as part of a more standard deep reinforcement learning setup. However, these methods assume the LTL specifications are already known, and [32, 33, 21, 18] do not allow for a model that is easy to interpret and manipulate. By contrast, our model only requires the current FSA state and the location of logic propositions in the environment.

Multi-task and Meta Learning We can also look at our method through the lens of multi-task and meta learning, methods which solve classes of tasks assuming a distribution over tasks [7, 2, 12, 14, 15]. LVIN is a model-based approach which can be viewed as sharing the structure of the low-level large MDP across tasks, while the high-level small FSA governs the task parameters. If we separated the learning of the FSA and the low-level MDP, we could learn the MDP across multiple tasks. Importantly, the FSA is human interpretable and manipulable, allowing us to change the task being solved with no new data (zero-shot), in contrast with one-shot methods like MAML [14] or [20] which require more data to adapt to new tasks from the task distribution.

Faulty Experts Other works tackle the problem of unreliable experts in imitation learning. [29] interpolates between imitation and intention learning with an entirely different approach based on inverse reinforcement learning, where transition dynamics are...
known. [17] uses reinforcement learning with expert demonstrations, while our approach only requires easy and direct modification of an interpretable policy.

**Hierarchical Learning** LVIN is an instance of hierarchical learning: We can view the FSA as a high-level description of the tasks the agent must accomplish. The first instance of hierarchical learning was introduced in [31]. Related is the options framework of [39]. The idea is to temporally abstract out the kinds of actions you must take in a sequence, and to use sequences of these actions to specify policies. [2] applies the options framework and uses policy sketches, sequential strings of sub-policies from a sub-policy alphabet, to build an overall policy. Here, only actions have hierarchical structure while our method simplifies the entire MDP by providing a high-level view via the logic FSA.

In more recent work, [27] combines high-level imitation policies with low-level reinforcement learned policies to train more quickly. [23] builds in a hierarchy of planning with models of objects in the world rather than only considering low-level states, similar to the propositions in our model. Both models lack an interpretable transition matrix which can be easily modified to change policy behavior.

### III. Problem Statement

Our goal is for agents to find interpretable and manipulable solutions to 2D gridworld tasks in the imitation learning setting. We define the finite gridworld state space \( S \) to consist of \((r, c)\) pairs \((r, c)\). The action space \( A \) consists of the 8 cardinal movement directions. We assume the agent is provided with expert demonstrations of a task (e.g., packing a lunchbox we can obtain new policies without re-learning by modifying the policy alphabet, to build an overall policy. Here, only actions have hierarchical structure while our method simplifies the entire MDP by providing a high-level view via the logic FSA.

We define the finite gridworld state space with expert demonstrations of a task (e.g., packing a lunchbox we can obtain new policies without re-learning by modifying the policy alphabet, to build an overall policy. Here, only actions have hierarchical structure while our method simplifies the entire MDP by providing a high-level view via the logic FSA.

LVIN uses expert trajectories to learn the approximate MDP and then (2) plans on top of the learned MDP using value iteration, a standard dynamic programming method which solves known MDPs [5]. An MDP is a tuple \((S, A, R, \gamma)\) corresponding to states, actions, a reward function \(R: S \times A \to \mathbb{R}\), a transition function \(T: S \times A \to \text{dist}(S)\) (a distribution over state space), and a discount factor \(\gamma \in (0, 1)\). LVIN learns the MDP parameters \(R, T, \gamma\), and the final policy is a lookup table commonly known as a Q-function [38]: each state \(s\) and action \(a\) are assigned a value \(Q(s, a)\). The policy is \(\pi(s) := \max_a Q(s, a)\).

We assume the state space of the approximate MDP we learn can be factored into two components: \(S\), the gridworld state space, and \(F\), the FSA states (defined by formulae of propositions). We thus learn a tuple \((S \times F, A, T, \gamma)\). The transitions \(T\) can be factored into the components \(P : S \times A \to \text{dist}(S)\) and \(TM : F \times L \to \text{dist}(F)\) assuming that unique propositions define the transitions between states in \(F\), i.e., the FSA is deterministic. The FSA MDP is given by \((F, L, 0, TM, 1)\). Note \(TM\) depends on \(S\) through \(M(S) = L\).

**Value-Iteration Network** The Value Iteration Network (VIN) [41] is a neural network architecture used to find policies for MDPs. The key idea is to learn a neural net policy given value features generated by a value iteration procedure on an MDP learned from expert data that approximates the true MDP of the environment. The main insight is that the standard value iteration algorithm can be expressed in the form of a convolutional neural network. For known MDP \((S, A, T, R, \gamma)\), the value iteration updates are given by:

\[
Q^{t+1}(s, a) \leftarrow R(s, a) + \sum_{s' \in S} \gamma T(s'|s, a) V^t(s')
\]

\[
V^{t+1}(s) \leftarrow \max_a Q^{t+1}(s, a)
\]

We interpret \(\gamma T(s'|s, a)\) as a convolution filter (the grid world dynamics are sparse), and the maximization over actions as a max-pooling step. By iterating these steps \(k\) times, we get a \(k\)-layer neural network. Since this procedure is differentiable, we can simultaneously learn a reward map \(R\), a transition matrix \(\gamma T\), the resulting value function \(V\), and a neural net policy over \(V\) simply by stacking each operation end-to-end and then backpropagating through the policy loss function (an imitation loss if there are experts, or an environment reward in the more general reinforcement learning setting) [41].
LVIN: Augmenting VIN with Logic We generalize VIN by assuming the factorization of the approximate MDP we learn into low-level and high-level components corresponding to $S$ and $\mathcal{F}$, respectively. The constraints on proposition map $\mathcal{M}$ identify propositions with the actions of the FSA MDP (Fig. 3). We think of the resulting factorization as creating a separate VIN for each FSA state (Fig. 4). We learn reward map $\mathcal{R}$ and transitions $\mathcal{T}$ describing an MDP by backpropagating the imitation loss from the Q-value policy through the following network. For input $(s, f, a)$, the output of the $j$th layer of the LVIN network is:

$$LVIN_{j+1}(\gamma \mathcal{F}, \mathcal{R}, V_j) := (\overline{Q}(s, f, a), \overline{V}(s, f), \overline{V}(s, f))_{j+1}$$

(1)

$$\overline{Q}_{j+1}(s, f, a) := \mathcal{R}(s, f, a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s, a) \overline{V}^j(s', f)$$

(2)

$$\overline{V}^j+1(s, f) := \max_a \overline{Q}^j+1(s, f, a)$$

(3)

$$\overline{V}^j+1(s, f) := \mathbb{E}_{f' \sim \mathcal{R}(f, \mathcal{M}(s))} \left[ \overline{V}^j(s', f') \right]$$

(4)

$$LVIN(\gamma \mathcal{F}, \mathcal{R}, \{\overline{V}^j\}_{j=0}^k) := LVINK_{k}(\cdot, \cdot, \cdot, \cdot, \cdot)$$

(5)

Alg. 1 gives the full training algorithm. We augment the training of $\overline{T}$ with an additional predictive task: given the current state, FSA state and an action, predict the next FSA state. This objective is meant to provide supervision to the task of learning the FSA transition matrix factor of the full transition dynamics. There are therefore two training losses: (1) a cross-entropy loss on next FSA state prediction, for learning the unknown FSA transition matrix, and (2) a cross-entropy loss on the action prediction. Cross-entropy between distributions $p$ and $q$ is denoted $H(p, q)$.

**Algorithm 1** LVIN Multi-Trajectory Training

1: procedure LVIN-TRAINING
2: Training Inputs: $\{(s_t, f_t, \mathcal{M}(s_t), a_t)^{(n)}\}_{t=1}^N$
3: To learn:
4: Transition matrix $\mathcal{T} \in \mathbb{R}^{F \times F \times \mathcal{L}}$
5: Low-level action kernels $\mathcal{P}(\cdot|\cdot, a)$
6: Value and Q functions $\overline{V}, \overline{Q}$
7: Build the model $LVIN(\gamma \mathcal{F}, \mathcal{R}, \{\overline{V}^j\}_{j=0}^k)$
8: Normalize $\mathcal{T}M$ so that it is row-stochastic.
9: for all $s \in S$: $\mathcal{M}(s) \neq \emptyset$, all $f \in \mathcal{F}$ do
10: $\overline{V}^j+1(s, f) := \mathbb{E}_{f' \sim \mathcal{R}(f, \mathcal{M}(s))} \left[ \overline{V}^j(s', f') \right]$
11: end for
12: for all $(s_t, f_t, \mathcal{M}(s_t), a_t)^{(n)}$ in data do
13: Gradient update on $\mathcal{T}$:
14: loss $= H(f_{t+1}, \overline{\mathcal{M}}(f_t, \mathcal{M}(s_t)))$
15: Backpropagate the imitation loss through LVIN:
16: loss $= H(a_t, \sigma_{\text{softmax}}(Q_{LVIN}(s_t, f_t)))$
17: end for
18: end procedure

Avoiding System Identification Since we learn the MDP with an imitation loss and a predictive loss, we avoid the sample-inefficient system identification problem (learning the true MDP exactly as in model-based IL): we only care about approximate MDPs which result in good policies after planning (a similar property holds for VIN). Our approach therefore lies inbetween model-based and model-free IL and benefits from properties of both settings. Thus we expect to see some errors in $\mathcal{T}$ and $\mathcal{R}$ that barely affect the LVIN policy, leaving interpretability and manipulability intact.

Fig. 3: The FSA transition matrix (learned by predicting the next FSA state) connects the value maps across FSA states (S0, S1, G, T) (see Eq. (4)). Each proposition (A: first goal, O: obstacle, B: second goal) is associated with a row of the learned TM for each FSA state based on $\mathcal{M}$.

Fig. 4: LVIN forward pass: Each FSA state has a value map. $\mathcal{P}(\cdot|\cdot, a)$ are shared across FSA states (can relax), and applied to produce Q-maps. Max-pooling yields updated value maps for each FSA state. The third layer output is depicted in detail in Fig. 3. The process is looped $k$ times, see Eqs. (1)-(5).

V. Experiments and Results

We test LVIN against 2-3 baselines on 4 domains. As in [41, 3, 8], we consider gridworlds, which can express many complex tasks. (LVIN extends to any discrete environment (including dynamic ones) with more compute.) Each domain illustrates a key claim. The kitchen domain lends itself to in-depth analysis due to its simplicity. In the longterm domain, the agent must collect four keys and pass through four doors in sequence in order to reach the goal. Its complexity shows how the learned TM can be interpreted to understand the rules governing the agent’s behavior even in cases where the actual FSA is difficult to understand.
as discussed in Sec. V-D. We show how to manipula
ate a TM in
the pickworld domain, Sec. V-E. Lastly, the rules of the driving
domain show how the TM can be modified to fix policies learned
from demonstrations by faulty experts, Sec. V-F.

A. Generating Expert Data

Linear Temporal Logic We use linear temporal logic (LTL) to
formally specify tasks [9]. Formulae φ constructed in LTL have
the syntax grammar

\[ φ ::= p \mid ¬φ \mid φ_1 ∨ φ_2 \mid ∗φ \mid φ_1 U φ_2 \]  

(6)

where \( p \) is a proposition (a boolean-valued truth statement that
can correspond to objects or goals in the world), \( ¬ \) is negation,
\( ∨ \) is disjunction, \( ∗ \) is “next”, and \( U \) is “until”. The derived
rules are conjunction \((∧)\), implication \((⇒)\), equivalence \((⇔)\),
“eventually” \((φ \equiv U True φ)\) and “always” \((φ \equiv □φ)\), see [4] for details. Intuitively, \( φ_1 U φ_2 \) means that \( φ_1 \) is true until
\( φ_2 \) is true, \( ∗φ \) means that there is a state where \( φ \) is true and \( □φ \)
means that \( φ \) is always true.

Generating Data We use the software packages SPOT [13]
and Lomap [43] to convert LTL formulae into FSAs. Every FSA that
we consider has a goal state, referred to in figures as G, which is
the desired final state of the agent, as well as a trap state, referred
to in figures as T, which is an undesired terminal state. For each
domain, we generate a set of environments in which obstacles and
other propositions are randomly placed. Given the FSA and an
environment, we run Dijkstra’s shortest path algorithm to create
expert trajectories that we use as data for imitation learning.

B. Baselines

VIN: We compare the performance of LVIN to VIN. VIN cannot
predict the next FSA state, nor can it learn a TM.

Hard-coded LVIN: It is not necessary to learn the TM from data
– if the FSA is known, then the TM corresponding to the FSA can
be used. We compare the performance of LVIN to LVIN with a
hard-coded TM to see if learning the TM degrades performance.

CNN: We formulate a less constrained version of LVIN that uses a
3D CNN instead of a TM to transfer values between FSA states.
A TM is also learned, but it is learned independently of the action
predictions and is not used in planning. The CNN operation
acts on a concatenation of the proposition matrix and the value
function, returning the next iteration of the value function. The
CNN has \(|F|\) input and \(|F|\) output channels. The kernel size is
\((|L| + |F|, 1, 1)\), so the convolution operates on one cell \((r, c)\)
of the domain at a time, linearly combining the propositions and
logic state values.

C. Environments

Kitchen Domain We first examine the kitchen domain (shown in
Fig. 5a), an \(8 \times 8\) gridworld with deterministic actions that enable
movement to adjacent cells. The domain has three propositions:
o for obstacle, a for milk, and b for cereal. The specification is
\( ∗(a ∧ ∗b) ∧ □¬o \) – first fill the bowl with milk (visit a) and
then put in the cereal (visit b) while avoiding randomly placed
obstacles (chairs, tables, and plants).
The first two rows of Fig. 5b show that LVIN, hard-coded LVIN,
and the CNN baseline all learn how to plan. The VIN baseline,
however, has poor action prediction. Examining the VIN’s failure
modes reveals that since it has no record of which goal it has
visited, the VIN model goes to the cereal without going to the
milk almost as frequently as it visits both goals in the correct
order. It also often visits only the milk without ever advancing to
the cereal. VIN’s performance highlights the importance of the
FSA as a form of memory for tasks with sequential steps. We also
note that the CNN baseline performs just as well as LVIN,
indicating that explicitly integrating a TM into the planning step
is not necessary for good performance.

Longterm Domain The longterm domain is a \(12 \times 9\) gridworld
and shows LVIN’s ability to learn complex sequential specifi-
cations. In this environment (Fig. 6a) there are 10 propositions: keys
ka, kb, kc, kd that unlock doors da, db, dc, and dd, respectively;
and g for the goal and o for obstacles. To progress to the goal, the
agent must follow the specification \(g ∧ □¬o ∧ (¬da U ka) ∧
(¬db U kb) ∧ (¬dc U kc) ∧ (¬dd U kd)\) that involves learning
a “longterm” plan – it must first pick up Key A, then go get Key
D, then Key B, then Key C, before it can access the room in
which the goal is located. The results in Table 1a show that while
LVIN and the CNN baseline have good performance, VIN cannot
complete a single rollout successfully, which is unsurprising
given the lengthy sequential nature of this domain.

Pickworld Domain The pickworld domain is an \(18 \times 7\) grid-
world where the agent must first pick up either a sandwich a
or a burger b and put it in a lunchbox d, and then pick up
a banana c and put it in the lunchbox d. The specification is
\( ∗((a ∨ b) ∧ ∗(d ∧ ∗(c ∧ ∗d))) ∧ □¬o \). As shown in Table Ib, VIN

<table>
<thead>
<tr>
<th>Action</th>
<th>Accuracy</th>
<th>Hard-coded LVIN</th>
<th>CNN</th>
<th>VIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVIN</td>
<td>98.85%</td>
<td>99.07%</td>
<td>98.29%</td>
<td>68.05%</td>
</tr>
<tr>
<td>FSA</td>
<td>99.71%</td>
<td>99.73%</td>
<td>99.73%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Performance over 5000 rollouts:

<table>
<thead>
<tr>
<th></th>
<th>Both Goals, Correct Order</th>
<th>Only Milk</th>
<th>Only Cereal</th>
<th>Both Goals, Wrong Order</th>
<th>No Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVIN</td>
<td>99.84%</td>
<td>99.76%</td>
<td>99.20%</td>
<td>99.84%</td>
<td>99.76%</td>
</tr>
<tr>
<td>Hard-coded LVIN</td>
<td>99.76%</td>
<td>99.66%</td>
<td>99.10%</td>
<td>99.76%</td>
<td>99.66%</td>
</tr>
<tr>
<td>CNN</td>
<td>99.20%</td>
<td>99.10%</td>
<td>98.00%</td>
<td>99.20%</td>
<td>99.10%</td>
</tr>
<tr>
<td>VIN</td>
<td>99.20%</td>
<td>99.10%</td>
<td>98.00%</td>
<td>99.20%</td>
<td>99.10%</td>
</tr>
</tbody>
</table>
cannot complete a single successful rollout due to the sequential nature of the domain.

**Driving Domain** The driving domain (Fig. 6c) is a $14 \times 14$ gridworld with the goal of showcasing LVIN’s ability to learn and encode rules, in this case three “rules of the road.” The model must learn three rules of the road – reach the goal ($g$) and encode rules, in this case three “rules of the road.” The gridworld contains parts of the TM of the longterm domain learned by LVIN. Cells of interest are highlighted in yellow; unexpected values are highlighted in red. Propositions in a given state that are never encountered are shaded in gray. This is because there is only one “sequential” goal, which is to reach the goal. Otherwise, the rules of the road can be encoded into the VIN’s reward function. However, since the VIN does not encode the TM, it can only learn the rules implicitly, whereas the LVIN and CNN baselines can learn the rules explicitly. This allows a user to check that safe rules have been learned, as discussed in Sec. V-F.

### Table I

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>da</td>
<td>100.00%</td>
</tr>
<tr>
<td>S1</td>
<td>db</td>
<td>82.80%</td>
</tr>
<tr>
<td>S2</td>
<td>dc</td>
<td>0.00%</td>
</tr>
<tr>
<td>S3</td>
<td>dd</td>
<td>100.00%</td>
</tr>
<tr>
<td>S4</td>
<td>db</td>
<td>82.80%</td>
</tr>
<tr>
<td>G</td>
<td>kc</td>
<td>99.40%</td>
</tr>
<tr>
<td>T</td>
<td>kc</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

(a) In the initial state, Door A is not allowed, and Key A leads to state S1.

(b) In S1, Key D leads to S2.

(c) In S2, Key B leads to S3.

(d) In S3, Key C leads to S4.

(e) S4 leads to goal.

### Table II: The learned transition matrix of the longterm domain

<table>
<thead>
<tr>
<th>State</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The specification is 10 propositions and translates to an FSA with 33 states. With no knowledge of the specification it would be difficult to tell what is going on. However, the structure of the domain forces the agent to pick up the keys in a specific order. Therefore the agent visits only 6 of the 33 states (7 including the trap state). Since LVIN learns the TM of this reduced FSA, after training we can examine the TM to reconstruct the FSA and learn the rules of the system. Table II contains parts of the TM of the longterm domain learned by LVIN.
LVIN. Each table is associated with one FSA state and specifies how each proposition maps to the next FSA state.

In the TM of the initial state $S_0$, Table IIa, we see that all doors and keys map to the trap state, except for Key A, which maps to state $S_1$. In other words, the model has learned that when in $S_0$, the agent is not permitted to travel through doors and that it must pick up Key A before any other key. Partial TMs for the other states show that the model has learned a sequence of keys to pick up, and that it cannot pass through the door associated with its key until it has picked up the key. Unexpected transitions are highlighted in red. In every case, unexpected transitions occur where the model has not actually observed a transition (columns highlighted in grey) but rather had to infer the value. This is understandable because in these cases the model picks a value that helps it reduce its loss, which may not coincide with the true TM.

The learned and true TMs of the other domains are shown in Fig. 7, and they can be analyzed in a similar way to the longterm domain’s TM. Note that the trap and goal states are not shown because they are trivial. For the kitchen and pickworld domains, the states are ordered sequentially – for the kitchen domain (Fig. 7a), the goal in $S_0$ is to reach $a$, and the goal in $S_1$ is to reach $b$. In the pickworld domain (Fig. 7b), the goal in $S_0$ is to reach either $a$ or $b$; the goal in $S_1$ is to reach $c$; the goal in $S_2$ is to reach $d$; and the goal in $S_3$ is to reach $d$ again. For the driveworld domain (Fig. 7c), $S_0$ corresponds to when the car is in the right lane, driving towards goal $g$. $S_1$ corresponds to when the car is in the left-hand lane, and $S_2$ corresponds to when the car is at a red light.

### E. Manipulability

Since we can interpret the TM layer, we can also modify it to change the behavior of the agent in a predictable way. To demonstrate, we manipulate the TM that was learned in the pickworld domain. The learned TM tells the robot to pick up either the sandwich or the burger, put it in the lunchbox, and then pick up the banana and put it in the lunchbox; $\phi_{p1} = \Box((a \lor b) \land (S_1 \land (c \lor d))) \land \Box \neg \neg a$. However, whoever is having their lunch packed may have a preference between sandwich and burger. Let “only sandwich, then banana” be $\phi_{p2}$ and “only burger, then banana” be $\phi_{p3}$. As another example, a user may prefer the banana to be packed before the sandwich/burger. Let “banana, then sandwich/burger” be $\phi_{p4}$.

The modifications to the TM are shown in Fig. 8. To make the agent pick up only the sandwich, we modify the TM’s initial state $S_0$ so that $b$ (the burger) maps back to $S_0$ instead of the next state $S_1$ (Fig. 8a). Similarly, to pick up only the burger, we modify $S_0$ so that $a$ (the sandwich) maps back to $S_0$, as shown in Fig. 8b, and the sandwich is ignored. Lastly, to modify the TM to pick up the banana first (Fig. 8c), we first modify $S_0$ so that $a$ and $b$ are ignored and $c$ (the banana) leads to $S_1$. In $S_1$, the agent’s goal is to drop its payload into the lunchbox. We then modify the next state, $S_2$, so that $a$ and $b$ are the goals of $S_2$ and $c$, the banana, is ignored. These modifications result in the robot picking up the banana in $S_0$ and picking up the sandwich or burger in $S_2$. The results of these changes can be seen in Table III. The first two columns show that LVIN and the CNN baseline trained directly on these new specifications successfully learn them. The second pair of columns shows the performance of LVIN and the CNN baseline when they are initially trained on the original specification, $\phi_{p1}$, and then have their TMs modified to represent the new specifications. Modified LVIN has equivalent performance to the LVIN model trained directly on the new specifications. By contrast, the modified CNN model performs poorly because the TM is not integrated into the planning step for the CNN. Instead, the CNN baseline attempts to perform the old specification. The tests highlight an important shortcoming of the CNN model: Since the TM is not directly incorporated into the planning step, modifying the TM does not change the behavior of the agent.

**Manipulability Experiments on Jaco Arm** To show how LVIN can be applied to the real world, we implemented the algorithm...
on a Jaco arm, shown in Fig. 1. The Jaco arm is a 6-DOF arm with a 3-fingered hand and a mobile base. An Optitrack motion capture system was used to track the hand and the four objects of interest (a plastic banana, sandwich, burger, and lunchbox). The system was implemented using ROS [34]; the Open Motion Planning Library (OMPL) [37] was used for motion planning for the arm. The motion capture system was used to translate the positions of the hand and the lunch items into a 2D grid corresponding to Fig. 6b, and an LVIN model trained on simulated data was used to generate a path satisfying the specification. The LVIN model trained on $\phi_{p1}$ was then “manipulated” as discussed to follow specifications $\phi_{p2}$, $\phi_{p3}$, and $\phi_{p4}$. 20 trials were run for each specification with the Jaco arm, and the results are shown in Table IV. The arm was largely successful in carrying out the planned trajectories – 4 of the 5 failures were caused by the banana falling out of the hand, and only 1 of the 80 trials failed because of LVIN generating a bad path.

### F. Fixing Expert Errors

Our interpretable and manipulable model can also be used to fix the mistakes of faulty experts. Suppose the real-world driving data contains bad behavior from drivers breaking the rules. We model this scenario in Table V, where the Unsafe TM shows a scenario in which the model has learned to run a red light 10% of the time. This result can be observed directly from the TM, since the probability of entering the initial state given that the agent is on a red light is 10%, meaning it will ignore the red light, while the probability of recognizing that it is in a “red light” state is 90%. We correct the TM by setting the initial state entry to 0 and the red light state entry to 1. We perform 1000 rollouts using each of these TMs. The Unsafe TM results in the agent running 9.88% of red lights while the Safe TM prevents the agent from running any red lights.

<table>
<thead>
<tr>
<th>Unsafe TM</th>
<th>Safe TM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial State</strong></td>
<td><strong>red light</strong></td>
</tr>
<tr>
<td><strong>Left Lane</strong></td>
<td>0.0</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>0.0</td>
</tr>
<tr>
<td><strong>Red Light</strong></td>
<td>0.9</td>
</tr>
<tr>
<td><strong>Trap</strong></td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Rollout Performance

<table>
<thead>
<tr>
<th>Unsafe TM</th>
<th>Safe TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.88%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

TABLE V

VI. CONCLUSION

Developing interpretable and manipulable models that learn to plan is an ongoing goal in deep policy learning. By learning an FSA transition matrix in conjunction with a planning module, we were able to build a model that a human can control intuitively. As a result, the model immediately generalizes to new task specifications, can be fixed in the presence of expert errors, and can also solve long-term planning tasks requiring higher-level state transition information than the environment provides. Future work will focus on improving the poor scaling of computation with state and action space size, continuous state space versions of LVIN, and removing the logic oracle assumption during training time.

ACKNOWLEDGMENTS

NSF grant 1723943, ONR grant N000141812830, and the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001 supported this project. K.V. is supported by an NSF GRFP fellowship. Any opinions, findings, conclusions or recommendations in the paper do not necessarily reflect the views of the funding agencies.
REFERENCES


