Discovering Logic-Informed Intrinsic Rewards to Explain Human Policies

Chengzhi Cao [*1] **Yinghao Fu** [*1 2] **Chao Yang** [1] **Shuang Li** [1]

Abstract

In high-stakes fields like healthcare, it is crucial to distill valuable strategic insights from expert clinicians. This paper focuses on extracting these knowledge-based insights from demonstrations provided by experts, where we represent the knowledge as a set of logical rules. Our learning framework is built upon the classic Inverse Reinforcement Learning (IRL). We assume that experts, like clinicians, are rational, and the treatments they choose are the best choices based on their logical understanding of the situation. Our algorithm can automatically extract these logical rules from their demonstrations. We introduce a neural logic tree generator, which is trained to generate logical statements step by step, starting from the goal and working backward. This mirrors the way humans engage in backward reasoning. Similarly, we interpret policy planning as a forward reasoning process, where the optimal policy is determined by finding the best path forward based on the provided rules. The neural logic tree generator and the policy are learned using the IRL until convergence. This process ultimately leads to the discovery of the most effective strategic rules. As a bonus, our algorithm also allows us to recover the reward function. In our experiments, we demonstrate that our method excels at discovering meaningful logical rules, particularly in the context of healthcare.

1. Introduction

Although deep reinforcement learning has been used in *high-stakes* systems such as healthcare to aid clinicians in making medical decisions (Komorowski et al., 2018), the *lack of interpretability* of the learned *black-box* policies hinders their wide applications in real life. Clinicians are not satisfied with knowing which action to take in certain situations but are also interested in understanding why to take such actions.

In this paper, we focus on answering the following question: *how to automatically uncover the intrinsic logical knowledge to guide the reward design and explain the observational state-action trajectories from expert demonstrations?* In clinical decision systems, it is desirable to extract descriptive and high-level explanatory rules based on disease phenotypes and therapies from demonstrations by clinicians. These discovered logical rules can enhance the sharing of clinical insights, improve the interpretability of medical policies, and contribute to the refinement of treatment strategies.

Reward engineering has been a longstanding barrier in many complex decision-making problems. This paper aims to reveal *logic-informed* reward functions and policies by modifying the classic IRL framework (Ng et al., 2000; Finn et al., 2016b; Fu et al., 2018). The policies, reward functions, and their explanatory logical rules will be automatically learned from the data. Notably, our IRL framework automatically mines logic rules from data, which may reduce the cognitive bias introduced by humans and alleviate the human workload. Moreover, the discovered logic rules can be transferred to similar tasks to assist in reward design and policy learning, mitigating the sample efficiency bottleneck and addressing the sparse reward challenge.

Our overall learning framework is built upon the classic *inverse reinforcement learning* (IRL). Our IRL involves two stages: logic tree learning and policy learning. To facilitate efficient and differentiable rule discovery, we introduce a *neural logic tree generator* that generates a composition of logic variables in a sequential manner by mirroring the *backward* reasoning process. It starts with generating the goal and then subsequently generating the symbolic conditions necessary to achieve that goal. Our formulated IRL will drive the logic tree generator to adeptly capture the probabilistic distributions of the most effective strategic rules.

Our policy learning, however, can be regarded as forward reasoning, where the agent is optimized by identifying the most favorable path for forward chaining, given the discovered rules. The proposed IRL algorithm *alternates between neural logic tree learning and policy learning until convergence, resembling a cycle of backward and forward*

[*]Equal contribution [1]School of Data Science, The Chinese University of Hong Kong (Shenzhen) [2]Department of Biostatistics, City University of Hong Kong. Correspondence to: Shuang Li <lishuang@cuhk.edu.cn>.

ICML 2025 Workshop on Programmatic Representations for Agent Learning, Vancouver, Canada.

reasoning. Once converged, the algorithm yields the best set of explanatory logic rules and learned policies. As a by-product, the reward function can be recovered.

To address reward identification challenges in stochastic environments (Ng et al., 2000), we employ the Deep-PQR algorithm (Geng et al., 2020), a recent IRL method. Deep-PQR introduces the concept of "anchor actions" to facilitate the unique recovery of true reward functions. Overall, our IRL algorithm strategically optimizes the logic generator and policy function until convergence, subsequently recovering the Q-function and reward function. Ultimately, the resulting logic tree informs reward engineering and explains expert policies.

Our contributions can be summarized as follows:

(*i*) We present a novel IRL framework that simultaneously learns experts' logical reasoning processes and policies from observational data, enhancing interpretability compared to black-box solutions.

(*ii*) We introduce a neural rule generator capable of incrementally expanding symbolic trees using arithmetic and logical operators to explain experts' demonstrations. This expansion process is learned in a differentiable way.

(*iii*) Our reward learning framework is both tractable and efficient, allowing us to learn intrinsic and unique rewards guided by symbolic logic trees.

(*iv*) We evaluate our method on various datasets, including synthetic and a real-world healthcare scenario. The experimental results demonstrate that our symbolic reward learning framework can outperform most state-of-the-art methods.

2. Preliminaries: Expert Behavior Models

We consider an *agent behavior model* under the assumption of maximizing the *entropy-augmented* long-term expected reward. The Markov decision process (MDP) is represented by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma)$, where $\mathcal{S}$ denotes the state-space, $\mathcal{A}$ denotes the action-space, $\mathcal{P}$ represents the transition probability distribution, $r(\mathbf{s}, \mathbf{a})$ corresponds to the reward function, ρ_0 is the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor. State and action variables at time t are denoted as $\mathbf{s}_t$ and $\mathbf{a}_t$, respectively. $\pi(\mathbf{s}, \mathbf{a})$ represents the stochastic policy of agents, indicating the conditional probability $p(\mathbf{a}_t|\mathbf{s}_t)$. We model the agent's behavior π as follows:

$$\max_{\pi} \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\mathbf{s}_t, \cdot\right)\right) \mid \mathbf{s}_0 = \mathbf{s}\right], \quad (1)$$

where $\mathcal{H}(\pi(\mathbf{s}, \cdot)) := -\int_{\mathcal{A}} \log(\pi(\mathbf{s}, \mathbf{a}))\pi(\mathbf{s}, \mathbf{a})d\mathbf{a}$ is the information entropy. $\alpha\mathcal{H}\left(\pi\left(\cdot \mid \mathbf{S}_t\right)\right)$ with $\alpha > 0$ is introduced to encourage the exploration of the agent behavior $\pi(\mathbf{s}, \mathbf{a})$.

If we assume the stochastic policy takes a Boltzmann distribution, also known as stochastic energy-based policy, i.e., $\pi(\mathbf{s}, \mathbf{a}) = \frac{\exp(-\mathcal{E}(\mathbf{s}, \mathbf{a}))}{\int_{\mathbf{a}' \in \mathcal{A}} \exp(-\mathcal{E}(\mathbf{s}, \mathbf{a}')d\mathbf{a}')}$, where $\mathcal{E} > 0$ is an energy function. One can prove that (Haarnoja et al., 2017) by assuming that agents take energy-based policies, the optimal policy function by solving Eq. (1) will be

$$\pi^*(\mathbf{s}, \mathbf{a}) = \frac{\exp\left(\frac{1}{\alpha} Q(\mathbf{s}, \mathbf{a})\right)}{\int_{\mathbf{a}' \in \mathcal{A}} \exp\left(\frac{1}{\alpha} Q\left(\mathbf{s}, \mathbf{a}'\right)\right) d\mathbf{a}'}, \quad (2)$$

Moreover, the likelihood of the observed demonstrations $\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \cdots, \mathbf{s}_T, \mathbf{a}_T\}$ can be derived (using the chain rule) as

$$p(\tau) = \prod_{t=0}^{T} \pi^*\left(\mathbf{s}_t, \mathbf{a}_t\right) \prod_{t=0}^{T-1} p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right). \quad (3)$$

Eq. (2) shows that the optimal agent behavior will take the actions that yield higher accumulative reward with exponentially higher probabilities. In our paper, we assume that *the experts are adopting the optimal policy of form Eq. (2).*

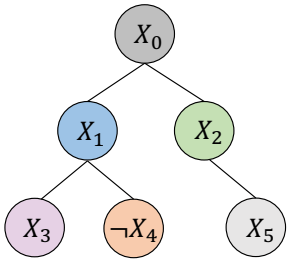

Figure 1. Illustration of a logic tree $\mathcal{R}$. The tree indicates that to reach the goal X_0, the following logic rules: $X_0 \leftarrow X_1 \wedge X_2$, $X_1 \leftarrow X_3 \wedge \neg X_4$, and $X_2 \leftarrow X_5$ must be executed.

3. Logic Tree Generator

Our key idea involves the introduction of a *neural logic tree generator*, from which *a collection of logic rules that need to be executed to reach the goal* will be generated in a top-down fashion, mirroring the *backward* reasoning process of humans. As shown in the left segment of Fig. 2, our generator takes the observed state-action demonstrations as input, and initially encodes them into the *symbolic predicate space*. Subsequently, it utilizes a Transformer-based autoregressive generative model to produce the logic trees. The obtained logic trees serve as symbolic explanations for the expert demonstrations. Moreover, they will be subsequently leveraged in the policy and reward learning phase. We will defer to Section 4 to discuss the IRL algorithm in terms of policy and reward learning, where the parameters of our logic tree generator will also be learned.

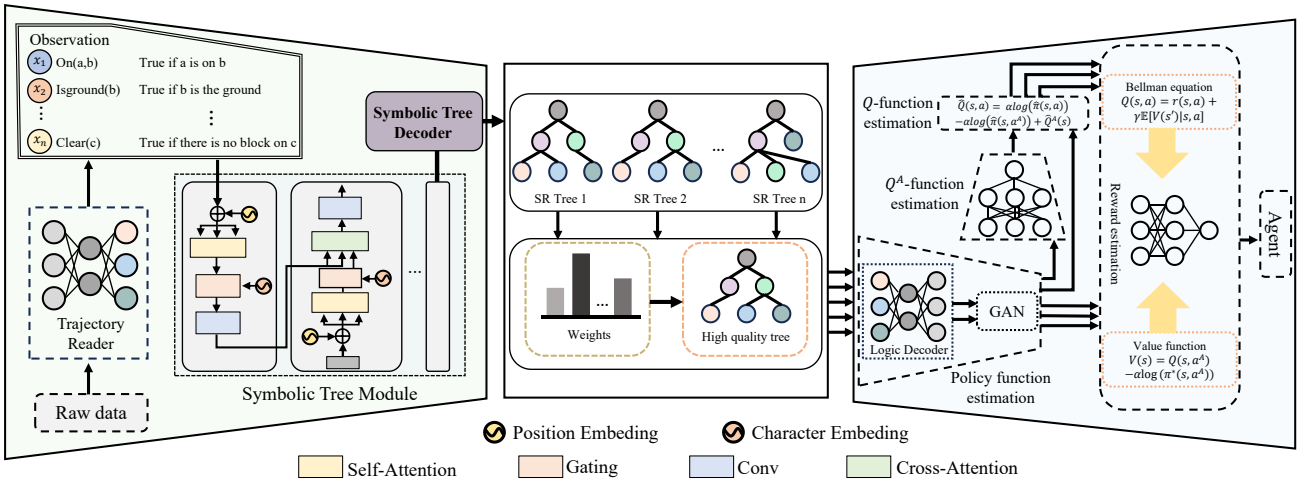

Figure 2. Overview of our proposed framework. Left segment: The diagram depicts the architecture of the neural logic tree. We utilize a Transformer-based autoregressive generator to create the logic trees in a sequential manner. Right segment: The diagram illustrates our comprehensive logic-informed Inverse Reinforcement Learning (IRL) framework. The IRL framework employs a Generative Adversarial Network (GAN)-based cost learning approach, where we formulate a minimax problem to simultaneously train the neural logic generator and the policy. Additionally, the reward function can be uniquely recovered as a by-product of this process.

3.1. Symbolic Logic Tree

Let $\mathcal{X}$ be a predicate set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, where each predicate $X_i \in \{0, 1\}$ is Boolean logic variable. Formally, a symbolic tree $\mathcal{R}$ consists of a set of logic rules:

$$\bigcup_{k\in\mathcal{K}} \left\{ f_k \,\middle|\, f_k := \bigwedge_{u\in I_k^1} X_u \wedge \bigwedge_{v\in I_k^0} \neg X_v \right\}, \tag{4}$$

where each logic rule f_k from the rule index set $\mathcal{K}$ is written as a conjunctive clause, and I_k^1 (resp. I_k^0) is the index set of the variables in $\mathcal{X}$ without (resp. with) a NOT operator. An example of the logic tree $\mathcal{R}$ considered in our paper is shown in Fig. 1, where the tree defines a collection of logic rules, indicating the conditions that need to be satisfied to reach the goal.

3.2. Autoregressive Generative Model

Our (amortized) neural logic tree generator is designed to generate a symbolic logic tree with a distribution $p_\phi(\mathcal{R} \mid \tau)$, conditional on a state-action trajectory τ. Specifically, we generate a symbolic logic tree $\mathcal{R}$ as a *sequence* in an autoregressive manner, according to a *pre-order traversal*, i.e.,

$$p_\phi(\mathcal{R} \mid \tau) = p_\phi(X^g \mid \tau) \prod_{l=2}^{L} p_\phi(X^l \mid \tau, X^{<l}), \tag{5}$$

where the index l refers to the position in this pre-order traversal, and $X^0 = X^g$ is the goal predicate, and $\boldsymbol{X}^{<l} = (X^0, \ldots, X^{l-1})$. The goal X^0 in our model is conceptualized based on the desired outcome of the task, which may indeed be as binary as success versus failure in certain scenarios. It is generally fixed throughout a trajectory to provide a consistent objective. Our logic generator starts from the root node, representing the final goal, and generates subsequent subgoals and conditions.

Our symbolic tree generator contains three main components: (1) a *trajectory reader* encodes the raw trajectory data; (2) an *abstract symbolic tree reader* encodes the predicates and the previously generated partial symbolic tree; (3) a *decoder* integrates the node scheduled for expansion along with inputs from the two previous readers to predict the next predicate.

To construct $p_\phi(\mathcal{R} \mid \tau)$, we begin a trajectory reader, which tokenizes τ into the predicate space, resulting in a sequence of ordering $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, where n denotes the length of the grounded predicate sequence. This tokenization is performed using a set of predefined labeling functions, which are meticulously designed through a blend of domain-specific expertise and insights derived from pre-trained models. Then each grounded predicate $X_{(i)}$ is further divided into characters $c_1^{X_{(i)}}, c_2^{X_{(i)}}, \ldots, c_m^{X_{(i)}}$ by the first block of the abstract symbolic tree reader, where m represents the number of characters in $X_{(i)}$. The character here represents the state/action which is associated with this predicate. All predicates and characters are embedded as real-valued vector $\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \ldots, \mathbf{X}_{(n)}$ and $\mathbf{c}_1^{X_{(i)}}, \mathbf{c}_2^{X_{(i)}}, \ldots, \mathbf{c}_m^{X_{(i)}}$, then we can represent a predicate by character embeddings with a fully-connected layer. The first block of the abstract symbolic tree reader contains three different sub-layers: self-attention, gating mechanism,

and convolution, designed to extract features. The neural structure of our reader follows the Transformer architecture and uses multi-head attention to capture long-dependency information.

While our trees are generated by predicting sequences of predicates, these predicates alone lack a concrete representation of the tree's structure, making them insufficient for predicting the next predicate. Therefore, the rest blocks of the abstract symbolic tree reader were used to incorporate both the predicted predicates and the tree's structural information. Specifically, We encode the rules using an attention mechanism and subsequently employ a tree convolution layer to amalgamate the encoded representation of each node with its ancestors. Our final component is a decoder that integrates information from generated logic rules with the state-action trajectory description and predicts the next predicate. We also employ a pointer network to directly identify a predicate from the input, designating it as a logic tree's terminal node. The decoder treats the non-terminal node to be expanded as a query, represented as a path from the root to the node to be expanded. For detailed input descriptions and neural structures for each module, please refer to Appendix B.

4. Learning Framework

Next, we introduce our innovative logic-informed IRL framework, which aims to learn three key components: the logic tree generator represented as p_ϕ, the expert policy, and the reward function. Our algorithm operates by iteratively updating the expert policy and the rule generator p_ϕ until convergence. Subsequently, we recover the Q-function and reward function in a specific order. We will provide more detailed explanations in the following sections.

4.1. Joint estimation of Expert Policy and logic generator

Logic-Informed Energy Function Our innovation lies in the design of the *energy function*, which encapsulates *high-level strategic* information revealed from the trajectory, utilizing predefined predicates and logic rules. More precisely, we propose to parametrize the energy function using *logic-informed features* given a generated logic tree (see Eq. (4)), as outlined in the following equation:

$$\mathcal{E}_\theta(\tau;\mathcal{R}) = -\sum_{k\in\mathcal{K}(\mathcal{R})}\theta_k\Big(\sum_{u\in I_k^1(\mathcal{R})}X_u(\tau) - \sum_{v\in I_k^0(\mathcal{R})}X_v(\tau) - |I_k^1(\mathcal{R})| + \epsilon\Big). \tag{6}$$

Here, $0<\epsilon<1$, $\theta_k\geq 0$; the index set of logic rules $\mathcal{K}$ and the index set of variables f_k^1, f_k^0 are associated with the logic tree $\mathcal{R}$; $|\cdot|$ denotes the cardinality of a set; and all predicate variables X_u, X_v are grounded by the trajectory τ. The parameter θ_k can be designed as a unique parameter for each rule within the tree, or it can be shared among all the rules in the tree. Sharing the parameters across rules can help reduce the number of parameters that need to be learned.

Each summand in Eq. (6) (i.e., the value within the big parentheses), once exponentiated, served as a soft approximation of the logic rule $f_k(\tau)$ as defined in Eq. (4). To illustrate, when $X_u(\tau)=1$ for every $u\in I_k^1(\mathcal{R})$ and $X_v(\tau)=0$ for every $v\in I_k^0(\mathcal{R})$, we have that $f_k(\tau)=1$ and the exponential of the summand equals $e^{+\epsilon}$. Conversely, when $X_u(\tau)=0$ for every $u\in I_{f_k}^1(\mathcal{R})$ and $X_v(\tau)=1$ for every $v\in I_k^0(\mathcal{R})$, we have that $f_k(\tau)=0$ and the exponential of the summand equals $e^{-|I_k^1(\mathcal{R})\cup I_k^0(\mathcal{R})|+\epsilon}$, which is presumably a very small number that is close to 0. The parameter θ_k controls the scaling of the exponential as well as the weight of each clause in the logic tree. We will revisit the interpretation of the energy function when we discuss the policy learning.

With a slight abuse of notation, we introduce a logic-informed energy function, which can be interpreted as the *product of experts* (Hinton, 2002), where each generated logic tree sample can be regarded as an expert's logical reasoning used in the energy function.

$$\mathcal{E}_{\theta,\phi}(\tau) = \mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\mathcal{E}_\theta(\tau;\mathcal{R})]. \tag{7}$$

Here, each symbolic tree $\mathcal{R}$ independently evaluates the likelihood of a trajectory τ as $\exp(-\mathcal{E}_\theta(\tau;\mathcal{R}))$. The distribution of expert's logic trees is $p_\phi(\cdot\mid\tau)$ and the expression $\exp(-\mathcal{E}_{\theta,\phi}(\tau))$ represents the combined likelihood.

It is crucial to note that $\mathcal{E}_{\theta,\phi}(\tau)$ is dependent on the learnable parameter θ and the distribution p_ϕ of the logic tree $\mathcal{R}$. During the learning process, the objective is to minimize the energy when applying an expert demonstration τ. In other words, the energy function is trained to increase the likelihood of grounding each predicate $\mathcal{X}_i$ within the symbolic tree $\mathcal{R}$ and the goal predicate. In practice, to make the expectation tractable, we will use the top-K logic trees with generated probabilities to approximate the expectation. This selection reflects a snapshot of the most probable trees at a given iteration and allows us to approximate the expectation without the computational expense of considering all possible trees. In our experiments, we use $K=1$, and the performance is quite satisfactory.

GAN Guided Cost Learning Now, given the logic-informed energy function, we consider a discriminator of the form

$$\mathcal{D}_{\theta,\phi,\psi}(\tau) = \frac{\exp(-\mathcal{E}_{\theta,\phi}(\tau))}{\exp(-\mathcal{E}_{\theta,\phi}(\tau)) + p_{\pi_\psi}(\tau)}, \tag{8}$$

where p_{π_ψ} represents the likelihood of a trajectory τ induced from a policy π_ψ, parameterized by ψ; $\exp(-\mathcal{E}_{\theta,\phi}(\tau))$ represents the likelihood of a state-action trajectory τ induced from the logic-informed energy model Eq. (7). The GAN-guided cost learning framework considers a minimax problem:

$$\min_{\psi}\max_{\theta,\phi}\mathbb{E}_{\tau\sim\pi_{\text{expert}}}\big[\log\mathcal{D}_{\theta,\phi,\psi}(\tau)\big]+ \mathbb{E}_{\tau\sim\pi_\psi}\big[\log(1-\mathcal{D}_{\theta,\phi,\psi}(\tau))\big]. \tag{9}$$

The discriminator maximizes the log-likelihood between the encapsulated raw data trajectory and the generated trajectories by the current policy π_ψ. Practically, we replace π_ψ with a mixture between the raw data trajectory and the generated trajectories by the current policy. Given ψ, the optimal discriminator satisfies $\mathcal{E}_{\theta^*,\phi^*}(\tau) = -\log p_\psi(\tau)$, at which the output of the discriminator is $1/2$ for all trajectories.

The GAN objective can be trained using gradient-based algorithms with appropriate parameterization. Before that, we can compute the gradient of $\mathcal{E}_{\theta,\phi}(\tau)$ as $\mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\nabla_\phi \log p_\phi(\mathcal{R} \mid \tau)\cdot\mathcal{E}_\theta(\tau;\mathcal{R})]$. Therefore, we can compute the gradient of the GAN objective with respect to ϕ using the log-derivative trick, which yields $\nabla_\phi\mathbb{E}_{\tau\sim\pi_{\text{expert}}}\big[\log\mathcal{D}_{\theta,\phi,\psi}(\tau)\big] + \mathbb{E}_{\tau\sim\pi_\psi}\big[\log(1-\mathcal{D}_{\theta,\phi,\psi}(\tau))\big]$.

Policy learning can be regarded as the importance sampling used for estimating the *partition function* of an energy-based model (Finn et al., 2016a). In other words, given the current energy function, the policy is trained to generate state-action trajectories that will yield a low-energy function (i.e., a high probability), resulting in the logic trees being more likely to be satisfied. In essence, policy learning is about finding the most effective path to achieve the end goal given the current logic rules.

Therefore, our overall logic-informed IRL can be seen as a cycle of alternating between backward reasoning and forward reasoning until convergence is achieved. Consequently, we can learn the best probabilistic distribution of the logic trees as well as the policy.

4.2. Reward Recovery

After learning the expert policy, we recover the reward function using the deep PQR algorithm (Geng et al., 2020), which is a modification to the well-known AIRL (Fu et al., 2018) to address the reward ambiguity. This is achieved by first identifying the Q-function under an anchor-action assumption and then estimating the reward function.

Two-stage Q-function Estimation Recall from Eq. (2) the relationship between the expert policy and the optimal Q-function, the ratio remains unchanged if $Q^*(\mathbf{s},\mathbf{a})$ is shifted by a function $\Psi(\mathbf{s})$. As a result, Q^* is not identifiable in the above form. To resolve this issue, we adopt the anchor-action assumption (Geng et al., 2020), which postulates the presence of an anchor action $\mathbf{a}^A \in \mathcal{A}$ such that its reward values $r(\cdot,\mathbf{a}^A)$ is fixed beforehand. In our setting, the anchor action is chosen as a "non-action", yielding no reward. Given the anchor estimator $\hat{Q}^A$ and the policy estimator $\pi_{\hat{\psi}}$, the Q-function is estimated by:

$$\hat{Q}(\mathbf{s},\mathbf{a}) = \alpha\log(\pi_{\hat{\psi}}(\mathbf{s},\mathbf{a})) - \alpha\log(\pi_{\hat{\psi}}(\mathbf{s},\mathbf{a}^A)) + \hat{Q}^A(\mathbf{s}). \tag{10}$$

Thereby, we can apply the Fitted-Q-Iteration Identification method, which amounts to solving a simple one-action MDP.

Reward Estimation Given the Q-function estimator $\hat{Q}$ and the policy estimator $\pi_{\hat{\psi}}$, the reward function is estimated in a manner similar to the Bellman equation:

$$\hat{r}(\mathbf{s},\mathbf{a}) = \hat{Q}(\mathbf{s},\mathbf{a}) - \gamma\hat{\mathbb{E}}_{\mathbf{s}'}[-\alpha\log(\hat{\pi}(\mathbf{s}',\mathbf{a}^A)) + \hat{Q}(\mathbf{s}',\mathbf{a}^A) \mid \mathbf{s},\mathbf{a}]. \tag{11}$$

Here, $\hat{\mathbb{E}}_{\mathbf{s}'}$ denotes the estimated expectation over the state variable $\mathbf{s}'$, which represents the one-step look-ahead state variable. The main idea behind the formulation in Eq. (11) is to replace the value function in the Bellman equation with a specification that explicitly incorporates agent policies into the reward functions. This representation avoids the direct calculation of the value function, simplifying the estimation of the expectation. To estimate the expectation, we use a deep neural network given $\hat{\pi}(\mathbf{s},\mathbf{a})$ and $\hat{Q}(\mathbf{s},\mathbf{a})$, transforming the reward estimation problem into a supervised learning task.

With this formulation, we can effectively estimate the reward function in a more direct manner without explicitly computing the value function. By considering the impact of agent policies, we obtain a more refined representation of the reward, enabling efficient supervised learning for the reward estimation process. The use of deep neural networks in this approach provides a flexible and scalable framework for reward estimation, making it applicable to a wide range of applications.

We want to emphasize that even though our reward function is defined at the state-action level, its learning process is implicitly influenced by the symbolic logic trees that we've learned. The learned logic trees, being at a higher level of strategy description, are more generalizable than the reward function.

5. Experiment

In this section, to test the robustness of our proposed framework, we conduct several experiments on synthetic and

Table 1. Comparison between different methods in the blocks world, sorting integers, and finding shortest paths, where m is the number of blocks in the blocks world environment or the size of the arrays/graphs in the sorting/path environment. The performance is evaluated by two metrics and separated by "/": the probability of completing the task during the test, and the average `Moves` used by the agents when they complete the task.

Methods	BlocksWorld		Sorting		Finding Path		Running time
	m=40	m=80	m=40	m=80	m=40	m=80	
NLM	100%/70	83%/247	100%/41	89%/216	100%/4	100%/32	66.9s
MemNN	57%/231	0%/N/A	96%/745	74%/1385	35%/36	0%/N/A	101.3s
MaxEnt-IRL	42%/264	0%/N/A	82%/811	44%/1879	26%/49	0%/N/A	32.6s
Deep PQR	78%/125	43%/362	100%/165	70%/553	77%/31	31%/198	161.0s
AIRL	53%/305	24%/1147	92%/766	71%/1406	71%/42	0%/N/A	70.8s
Ours	100%/59	93%/189	100%/41	97%/170	100%/4	100%/29	57.2s

real-world datasets and show that our method can solve a broad set of decision-making tasks, such as block manipulation and healthcare event prediction, and meanwhile provide strategic explanations.

5.1. Synthetic Experiments

We test our model's capability of decision-making in the classic blocks world domain by slightly extending the model to fit the formulation of the Markov Decision Process (MDP) in reinforcement learning. Additionally, we perform experiments specifically designed to validate the accuracy and quality of our estimated reward function.

Experimental Setting. We further show our model's ability to tackle algorithmic tasks, such as `BlocksWorld`, `Sorting`, and `Path`. We view an algorithm as a sequence of primitive actions and cast it as a reinforcement learning problem. Each synthetic dataset contains 1, 000 event streams partitioned into three sets: training (70%), validation (15%), and test (15%).

Baselines For IRL, we consider the classic MaxEnt-IRL proposed in (Ziebart et al., 2008), which estimates the Q-function and treats it as the reward function. Further, we include the AIRL method (Fu et al., 2018), which attempts to distinguish the reward function from the Q-function by a disentangling procedure. Moreover, we consider three baselines as representatives of the connectionist and symbolicist: Memory Networks (MemNN) (Sukhbaatar et al., 2015) and Neural Logic Machine (NLM) (Dong et al., 2019). We also make comparisons with other models such as Deep PQR (Geng et al., 2020).

Results. The results have been shown in Table 1. Notably, the performance gap between our approach and the Neural Logic Machines (NLM) (Dong et al., 2019) is evident. While both methods achieve close to the best possible result on each task, there is little room for improvement. But ours obtain fewer moves when finishing all tasks. The MemNN (Sukhbaatar et al., 2015) method is ineffective when the size of the arrays/graphs in the sorting/path environment becomes larger. Deep PQR (Geng et al., 2020) can presumably learn disentangled rewards, but we find that the formulation does not perform well even in learning rewards in the original task, let alone transferring to a new domain. It learns successfully in the training domain but does not acquire a representation that is suitable for transfer to test domains, with a 78% success rate of completing the `BlocksWorld` task (m=40). In contrast, our method not only excels in learning disentangled rewards but also adapts effectively to significant domain shifts, even in high-dimensional environments where exact reward extraction is challenging. The accuracy of our reward function estimates, as evidenced by the results of reward recovery, further validates the alignment of our approach with expert decisions, despite the constraints of our dataset.

Moreover, we also show a most likely logic tree generated from our algorithm in Fig. 3 to demonstrate its effectiveness. Given an initial configuration of blocks world, our goal is to transform it into a target configuration by taking a sequence of Move operations. The learning system should recover a set of rules and generalize to a blocks world that contains more blocks than those encountered during training. In our tree architecture, the leaf nodes define the initial states, and the inner nodes are determined by gating functions encoding the probability of taking the rightmost branch at each leaf node.

Rule Evaluation In the real-world dataset, there is no ground truth of learned logic rules, so we follow (Li et al., 2022) to verify our model's rule discovery ability on synthetic datasets with a known set of ground-truth rules and weights. Note that it was originally utilized for the temporal point process, so we modify it by adding spatial variables (such as "left, right, front, and behind") to fit in our settings. The weight learning results on 4 synthetic datasets

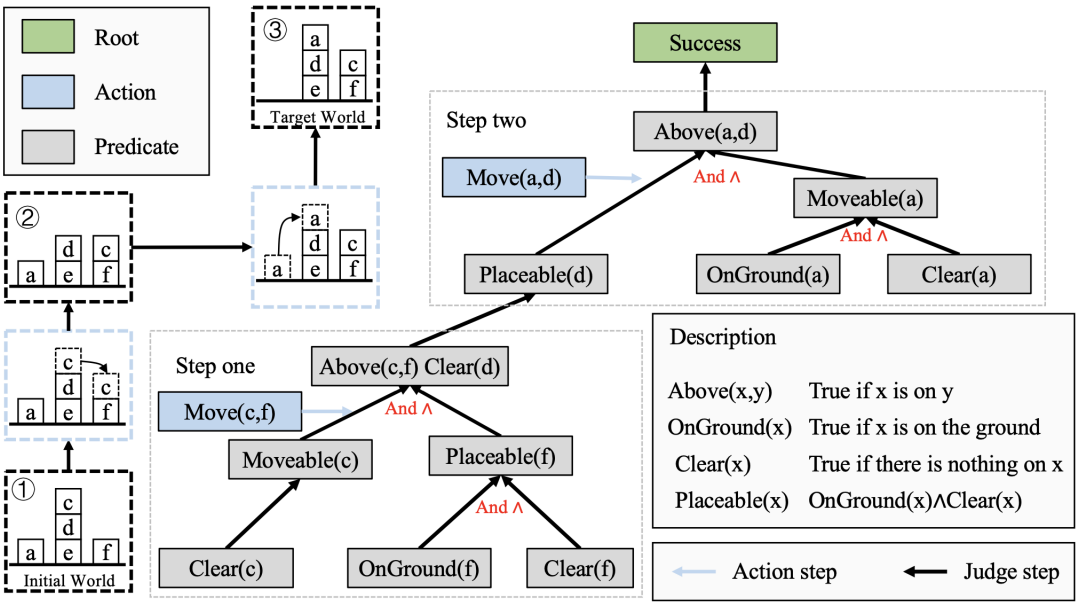

Figure 3. A logic tree induced by our algorithm in BlocksWorld task. In the first step, we move the block c onto the block f. In the second step, we move the block a onto the block d to the target configuration.

are shown in Table 3.

Running time We show the runtime for all methods averaged on BlocksWorld tasks. Note that all methods are tested on an RTX 2080Ti GPU. The Table 1 shows the time of 100 evaluation episodes. Our method is simpler to train at the same time achieving even better performance.

5.2. Real-world Experiments

Baselines. To compare our method with state-of-the-art models, we select the following methods as baselines: 1) RNN/Attention-based model: Chet (Lu et al., 2021a), HiTANet (Luo et al., 2020), ConCare (Ma et al., 2019), CGL (Lu et al., 2021b), Deep PQR (Geng et al., 2020) and NLRL (Jiang & Luo, 2019).

Results. We report the mean and standard deviations of evaluation metrics by running each model 5 times with different parameter initializations. Table 2 shows the results of diagnosis prediction using w-F_1 (%) and R@k (%). It shows that our method outperforms baselines on both datasets. Note that our method is better than CGL (Lu et al., 2021b) even without medical ontology graphs used in these two models. It further validates the significance of learning logic rules in health-care prediction. The attention-based method (Ma et al., 2019) and graph-based method (Lu et al., 2021b) can force models to only focus on the visits that contain risk factors and ignore the rest visits. Aggregating all visits together may further induce noise and hurt final performance. Our approach underscores the importance of explainable AI, balancing a slight uptick in performance with a marked improvement in transparency.

6. Related Work

Inverse reinforcement learning In the Machine Learning literature, inverse reinforcement learning (IRL) is a form of imitation learning and learning from demonstration. Imitation learning aims at learning policies from expert demonstrations, and IRL methods accomplish this by first inferring the expert's reward function (Jiang & Luo, 2019). The second goal is to run RL in a different environment with the estimated reward, where rewards are likely to transfer more readily across environments compared to policies. Previous IRL approaches have included maximum margin approaches (Abbeel & Ng, 2004; Jain et al., 2006) and probabilistic approaches such as (Levine et al., 2011; Ziebart et al., 2008). In this work, we work under the maximum IRL framework of (Geng et al., 2020). Some advantages of this framework are that it removes ambiguity between demonstrations and the expert policy, and allows us to cast the reward learning problem as a maximum likelihood problem. Generative adversarial imitation learning (Ho & Ermon, 2016) differs from our work in that it is not an IRL algorithm that seeks to recover reward functions. It aims only to recover the expert's policy, which is a less portable representation for transfer. (Uchibe, 2018) does not interleave policy optimization with reward learning within an adversarial framework. Improving a policy within an adversarial framework corresponds

Table 2. Diagnosis prediction results on MIMIC-III and MIMIC-IV using w-F_1 (%) and R@k (%).

Methods	MIMIC-III			MIMIC-IV		
	w-F_1	R@10	R@20	w-F_1	R@10	R@20
Chet	22.63(0.08)	28.64(0.13)	37.87(0.09)	26.35(0.13)	30.28(0.09)	38.69(0.15)
HiTANet	21.15(0.19)	26.02(0.25)	35.97(0.18)	24.92(0.27)	27.45(0.33)	36.37(0.24)
ConCare	20.94(0.06)	24.04(0.16)	34.11(0.12)	23.59(0.18)	26.52(0.13)	35.23(0.07)
Deep PQR	20.86(0.14)	24.61(0.08)	34.23(0.11)	23.13(0.13)	26.39(0.06)	35.45(0.19)
CGL	21.92(0.12)	26.64(0.30)	36.72(0.15)	25.41(0.08)	28.52(0.42)	37.15(0.29)
NLRL	20.23(0.06)	24.59(0.09)	34.03(0.07)	23.75(0.19)	26.80(0.14)	35.70(0.15)
Ours	22.78(0.09)	29.01(0.11)	38.10(0.08)	26.47(0.10)	30.35(0.07)	38.65(0.11)

Table 3. Rule discovery and weight learning results (GT weights/learned weights) on 4 synthetic datasets.

Weights	w_0	w_1	w_2
Dataset-1	1.00/0.94	1.00/0.91	2.00/1.85
Dataset-2	1.00/0.82	0.50/0.34	0.50/0.44
Dataset-3	1.00/0.89	1.50/1.33	1.50/1.39
Dataset-4	2.00/1.82	1.00/0.87	1.00/0.90

to training an amortized sampler for an energy-based model. Our work instead focuses on how to achieve generalization within the standard IRL formulation.

Intrinsic Rewards Learning Rewards learning aims at using heuristically designed intrinsic rewards in RL settings leading to interesting formulations. Liu et al. (Liu et al., 2014) propose a multi-agent architecture in which each agent, in addition to performing the standard role of designing appropriate policies, learns good reward functions from experience using a gradient-based approach. Kulkarni et al. (Kulkarni et al., 2016) present to integrate hierarchical action-value functions, operating at different temporal scales with goal-driven intrinsically motivated deep reinforcement learning. Dilokthanakul et al. (Dilokthanakul et al., 2017) design the discrete sets of sub-goals and their related intrinsic rewards that the meta-controller can select from. (Zheng et al., 2018) build on the optimal rewards framework to define the optimal intrinsic reward function as one that when used by an RL agent achieves behavior that optimizes the task-specifying or extrinsic reward function. Our work differs from these works in that the reward functions discovered are low-dimensional symbolic trees instead of high-dimensional neural networks.

7. Limitations and Discussion

We can enhance our logic-informed IRL framework in several ways: (*i*) Currently, the accuracy of the generated logic trees needs validation by human experts. In future work, we can develop a human-in-the-loop algorithm that allows flexible incorporation of human expert opinions in rule generation. (*ii*) In the current version, users must specify the predicate set in advance. However, can we enable our algorithm to automatically discover new concepts or predicates? (*iii*) At present, we employ a neural logic rule generator. In the future, we may replace it with a different approach like LLMs, and we're interested in evaluating the performance of our logic-informed IRL with this new knowledge generator.

8. Conclusion

In this paper, we present a logic-informed IRL framework. A neural logic generator is trained to produce logical rules to explain expert demonstrations and their underlying logical reasonings. Using the proposed IRL framework, both the neural logic generator and the policy are learned through a minimax optimization until the algorithm converges. Our overall logic-informed IRL can be seen as a cycle of alternating between backward reasoning and forward reasoning until convergence is achieved. As a bonus, our approach can also uniquely recover the reward functions. We empirically evaluated our methods on synthetic and real healthcare datasets, which demonstrated promising results.

Impact Statement

This paper introduces a novel framework aimed at enhancing the interpretability of decision-making systems through Inverse Reinforcement Learning, with a focus on logical reasoning processes. For application, particularly in healthcare, the proposed method can be used to learn the reward function of doctors managing the mechanical ventilation of patients and their sedation while being ventilated. It can develop an RL agent that can balance the risks of long-term mechanical ventilation with removing the ventilation too quickly from a patient. A reward function that encodes this information is difficult to handcraft and therefore should be learned from expert demonstrations. It suggests significant potential for societal benefits by improving decision-making accuracy and transparency.

References

Abbeel, P. and Ng, A. Apprenticeship learning via inverse reinforcement learning. *Proceedings of the twenty-first international conference on Machine learning*, 2004. URL https://api.semanticscholar.org/CorpusID:207155342.

Dilokthanakul, N., Kaplanis, C., Pawlowski, N., and Shanahan, M. Feature control as intrinsic motivation for hierarchical reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30:3409–3418, 2017. URL https://api.semanticscholar.org/CorpusID:38464943.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. Neural logic machines, 2019.

Finn, C., Christiano, P., Abbeel, P., and Levine, S. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016a.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pp. 49–58. PMLR, 2016b.

Fu, J., Luo, K., and Levine, S. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Geng, S., Nassif, H., Manzanares, C., Reppen, M., and Sircar, R. Deep pqr: Solving inverse reinforcement learning using anchor actions. In *International Conference on Machine Learning*, pp. 3431–3441. PMLR, 2020.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Neural Information Processing Systems*, 2016. URL https://api.semanticscholar.org/CorpusID:16153365.

Jain, A., Hu, M., Ratliff, N. D., Bagnell, D., and Zinkevich, M. A. Maximum margin planning. *Proceedings of the 23rd international conference on Machine learning*, 2006. URL https://api.semanticscholar.org/CorpusID:263868651.

Jiang, Z. and Luo, S. Neural logic reinforcement learning. In *International conference on machine learning*, pp. 3110–3119. PMLR, 2019.

Johnson, A. E., Bulgarelli, L., Shen, L., Gayles, A., Shammout, A., Horng, S., Pollard, T. J., Hao, S., Moody, B., Gow, B., et al. Mimic-iv, a freely accessible electronic health record dataset. *Scientific data*, 10(1):1, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Komorowski, M., Celi, L. A., Badawi, O., Gordon, A. C., and Faisal, A. A. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11):1716–1720, 2018.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *ArXiv*, abs/1604.06057, 2016. URL https://api.semanticscholar.org/CorpusID:4669377.

Levine, S., Popovic, Z., and Koltun, V. Nonlinear inverse reinforcement learning with gaussian processes. In *Neural Information Processing Systems*, 2011. URL https://api.semanticscholar.org/CorpusID:12063228.

Li, S., Feng, M., Wang, L., Essofi, A., Cao, Y., Yan, J., and Song, L. Explaining point processes by learning interpretable temporal logic rules. In *International Conference on Learning Representations*, 2022. URL https://api.semanticscholar.org/CorpusID:247613805.

Liu, B. D., Singh, S., Lewis, R. L., and Qin, S. Optimal rewards for cooperative agents. *IEEE Transactions on Autonomous Mental Development*, 6:286–297, 2014. URL https://api.semanticscholar.org/CorpusID:6013954.

Lu, C., Han, T., and Ning, Y. Context-aware health event prediction via transition functions on dynamic disease graphs. *ArXiv*, abs/2112.05195, 2021a. URL https://api.semanticscholar.org/CorpusID:245117528.

Lu, C., Reddy, C. K., Chakraborty, P., Kleinberg, S., and Ning, Y. Collaborative graph learning with auxiliary text for temporal event prediction in healthcare. *arXiv preprint arXiv:2105.07542*, 2021b.

Luo, J., Ye, M., Xiao, C., and Ma, F. Hitanet: Hierarchical time-aware attention networks for risk prediction on electronic health records. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 647–656, 2020.

Ma, L., Zhang, C., Wang, Y., Ruan, W., Wang, J., Tang, W., Ma, X., Gao, X., and Gao, J. Concare: Personalized clinical feature embedding via capturing the healthcare context. In *AAAI Conference on Artificial Intelligence*, 2019. URL https://api.semanticscholar.org/CorpusID:208310294.

Ng, A. Y., Russell, S., et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 1, pp. 2, 2000.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Uchibe, E. Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters*, 47:891–905, 2018. URL https://api.semanticscholar.org/CorpusID:23855231.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zheng, Z., Oh, J., and Singh, S. On learning intrinsic rewards for policy gradient methods. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:4933781.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

A. The Maximum Entropy Policy

In this appendix, we present proofs for the theorems that demonstrate how a policy function can be monotonically optimized with respect to the maximum entropy objective. Recall the objective function of Max-Entropy RL:

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{\infty} \gamma^t \mathbf{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\mathbf{s}_t, \cdot\right)\right) \mid \mathbf{s}_0 = \mathbf{s} \right]. \tag{12}$$

This objective corresponds to maximizing the discounted expected reward and entropy for future states originating from every state-action tuple $(\mathbf{s}_t, \mathbf{a}t)$ weighted by its probability ρ_π under the current policy. We begin by defining the Q-function for any given policy π. This Q-value represents the expected total reward, taking into account both rewards and entropy, under the policy π:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r_0 + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t \left(r_t + \alpha \mathcal{H}\left(\pi\left(\mathbf{s}_t, \cdot\right)\right)\right) \right]. \tag{13}$$

The discounted maximum entropy policy objective can now be defined as:

$$J(\pi) = \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[Q^\pi(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}\left(\pi\left(\mathbf{s}_t, \cdot\right)\right)\right]. \tag{14}$$

If we greedily maximizes the sum of entropy and value with one-step look-ahead, then we obtain $\hat{\pi}$ from π:

$$\mathbb{E}_{\mathbf{a} \sim \pi}\left[Q^\pi(\mathbf{s}, \mathbf{a})\right] + \alpha \mathcal{H}\left(\pi\left(\mathbf{s}, \cdot\right)\right) \leq \mathbb{E}_{\mathbf{a} \sim \hat{\pi}}\left[Q^\pi(\mathbf{s}, \mathbf{a})\right] + \alpha \mathcal{H}\left(\hat{\pi}\left(\mathbf{s}, \cdot\right)\right). \tag{15}$$

When we assume that the entropy parameter $\alpha = 1$, it is worth noting that:

$$\mathcal{H}\left(\pi\left(\mathbf{s}, \cdot\right)\right) + \mathbb{E}_{\mathbf{a} \sim \pi}\left[Q^\pi(\mathbf{s}, \mathbf{a})\right] = -D_{KL}\left[\pi\left(\mathbf{s}, \cdot\right) \| \hat{\pi}\left(\mathbf{s}, \cdot\right)\right] + \log \int \exp\left(Q^\pi\left(\mathbf{s}, \mathbf{a}\right)\right) d\mathbf{a}. \tag{16}$$

Then we can show that Q is bounded for any $\mathbf{s}$:

$$\begin{aligned}
Q^\pi(\mathbf{s}, \mathbf{a}) &= \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}\left(\pi\left(\mathbf{s}_1, \cdot\right)\right) + \mathbb{E}_{\mathbf{a}_1 \sim \pi}\left[Q^\pi(\mathbf{s}_1, \mathbf{a}_1)\right]\right] \\
&\leq \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_1, \cdot\right)\right) + \mathbb{E}_{\mathbf{a}_1 \sim \hat{\pi}}\left[Q^\pi(\mathbf{s}_1, \mathbf{a}_1)\right]\right] \\
&= \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma\left(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_1, \cdot\right)\right) + r_1\right)\right] + \gamma^2 \mathbb{E}_{\mathbf{s}_2}\left[\mathcal{H}\left(\pi\left(\mathbf{s}_2, \cdot\right)\right) + \mathbb{E}_{\mathbf{a}_2 \sim \pi}\left[Q^\pi\left(\mathbf{s}_2, \mathbf{a}_2\right)\right]\right] \\
&\leq \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma\left(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_1, \cdot\right)\right) + r_1\right)\right] + \gamma^2 \mathbb{E}_{\mathbf{s}_2}\left[\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_2, \cdot\right)\right) + \mathbb{E}_{\mathbf{a}_2 \sim \hat{\pi}}\left[Q^\pi\left(\mathbf{s}_2, \mathbf{a}_2\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{s}_1, \mathbf{s}_2, \mathbf{a}_2 \sim \hat{\pi}}\left[\left[r_0 + \gamma\left(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_1, \cdot\right)\right) + r_1\right) + \gamma^2\left(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_2, \cdot\right)\right) + r_2\right)\right]\right. \\
&\quad \left. + \gamma^3 \mathbb{E}_{\mathbf{s}_3}\left[\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_3, \cdot\right)\right) + \mathbb{E}_{\mathbf{a}_3 \sim \hat{\pi}}\left[Q^\pi\left(\mathbf{s}_3, \mathbf{a}_3\right)\right]\right]\right] \\
&\quad \vdots \\
&\leq \mathbb{E}_{\tau \sim \hat{\pi}}\left[r_0 + \sum_{t=1}^{\infty} \gamma^t \left(\mathcal{H}\left(\hat{\pi}\left(\mathbf{s}_t, \cdot\right)\right) + r_t\right)\right] \\
&= Q^{\hat{\pi}}\left(\mathbf{s}, \mathbf{a}\right).
\end{aligned} \tag{17}$$

So when we start with an arbitrary policy π_0 and define the *policy iteration* as:

$$\pi_i(\mathbf{s}, \mathbf{a}) = \frac{\exp\left(\frac{1}{\alpha} Q^{\pi_i}(\mathbf{s}, \mathbf{a})\right)}{\int_{\mathbf{a}' \in \mathcal{A}} \exp\left(\frac{1}{\alpha} Q^{\pi_i}\left(\mathbf{s}, \mathbf{a}'\right)\right) d\mathbf{a}'}. \tag{18}$$

Then $Q^{\pi_i}(\mathbf{s}, \mathbf{a})$ optimized monotonically, so π_i will converge to π_∞, and the optimal policy must satisfy this energy-based Boltzmann distribution.

Note that $Q(\mathbf{s}, \mathbf{a})$ can be calculated as

$$Q(\mathbf{s}, \mathbf{a}) := r(\mathbf{s}, \mathbf{a}) + \max_{\pi} \mathbb{E}\left\{ \sum_{t=1}^{\infty} \gamma^t \left[r\left(\mathbf{S}_t, \mathbf{A}_t\right) + \alpha \mathcal{H}\left(\pi\left(\mathbf{S}_t, \cdot\right)\right)\right] \mid \mathbf{s}, \mathbf{a} \right\}. \tag{19}$$

For GAN guided cost learning, we can compute the gradient of the GAN objective with respect to ϕ using the log-derivative trick, which yields:

$$\begin{aligned}
&\nabla_\phi \mathbb{E}_{\tau\sim\pi_{\text{expert}}}\big[\log \mathcal{D}_{\theta,\phi,\psi}(\tau)\big] + \mathbb{E}_{\tau\sim\pi_\psi}\big[\log(1-\mathcal{D}_{\theta,\phi,\psi}(\tau))\big] \\
&= -\mathbb{E}_{\tau\sim\pi_{\text{expert}}}\big[\nabla_\phi \mathcal{E}_{\theta,\phi}(\tau)\big] + \mathbb{E}_{\tau\sim\pi_{\text{expert}}}\left[\frac{\exp(-\mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\mathcal{E}_\theta(\tau;\mathcal{R})])}{\exp(-\mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\mathcal{E}_\theta(\tau;\mathcal{R})]) + p_{\pi_\psi}(\tau)}\cdot\nabla_\phi\mathcal{E}_{\theta,\phi}(\tau)\right] \\
&\quad + \mathbb{E}_{\tau\sim\pi_\psi}\left[\frac{\exp(-\mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\mathcal{E}_\theta(\tau;\mathcal{R})])}{\exp(-\mathbb{E}_{\mathcal{R}\sim p_\phi(\cdot|\tau)}[\mathcal{E}_\theta(\tau;\mathcal{R})]) + p_{\pi_\psi}(\tau)}\cdot\nabla_\phi\mathcal{E}_{\theta,\phi}(\tau)\right].
\end{aligned} \tag{20}$$

B. Transformer-based Symbolic Tree Generator

We generate the symbolic tree by predicting the predicates of the rules. Given the state-action trajectory and the currently generated partial abstract symbolic tree, our model can calculate the probabilities of the predicate to expand this node, as shown in Eq. (6) in the main paper.

B.1. Abstract symbolic tree reader

We design an abstract symbolic tree reader to model the structure of the generated partial symbolic tree. While our trees are generated by predicting sequences of predicates, these predicates alone lack a concrete representation of the tree's structure, making them insufficient for predicting the next predicate. Therefore, we apply the abstract symbolic tree reader to incorporate both the predicted predicates and the tree's structural information. It contains a stack of blocks, with the first block containing three distinct sub-layers previously introduced: self-attention, the gating mechanism, and the convolution layer. A residual connection is employed between each pair of consecutive sub-layers, following the approach outlined in (He et al., 2016), and is subsequently followed by layer normalization.

Self-Attention Within our Transformer block, multi-head attention is utilized to effectively capture long-range dependencies and facilitate the learning of non-linear features. In the case of a sequence of mapping predicates denoted as $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, their embeddings are obtained through a lookup table. Additionally, positional encoding is employed to encode positional information, which is computed as follows:

$$p_{j,i}[2k] = \sin\left(\frac{i+j}{10000^{2k/j}}\right), \tag{21}$$

$$p_{j,i}[2k+1] = \sin\left(\frac{i+j}{10000^{2k/j}}\right), \tag{22}$$

Here, $p_{i,j}[\cdot]$ refers to a specific dimension within the vector $\mathbf{p}_{i,j}$. In this context, j represents the jth block and k represents the embedding size. In the initial reader block, the input consists of the sum of the table-lookup embedding and the position embedding. In subsequent blocks, the input is the vector sum of the lower Transformer block's output and the position embedding specific to that block. The self-attention mechanism employed here follows the same architecture as described in the original Transformer (Vaswani et al., 2017). We denote the output of the self-attention as $\mathbf{X}^{\text{self}}_{(1)}, \mathbf{X}^{\text{self}}_{(2)}, \ldots, \mathbf{X}^{\text{self}}_{(n)}$.

Gating Mechanism Character embeddings are incorporated after self-attention, and the softmax weight $\mathbf{k}^{(c)}_{(i)}$ for character embeddings is obtained through a transformation from character embedding $\mathbf{X}_{(i)}$. The softmax weight $\mathbf{k}^{(y)}_{(i)}$ for the Transformer's output is derived from a linear transformation of $\mathbf{X}^{\text{self}}_{(i)}$. Additionally, the control vector $\mathbf{q}_{(i)}$ is obtained from $\mathbf{X}^{\text{self}}_{(i)}$ through a linear transformation. The gate can be computed as follows:

$$[\alpha^{(y)}_{(i),t}, \alpha^{(c)}_{(i),t}] = \text{softmax}\{\mathbf{q}^{\mathsf{T}}_{(i)}\mathbf{k}^{(y)}_{(i)}, \mathbf{q}^{\mathsf{T}}_{(i)}\mathbf{k}^{(c)}_{(i)}\}, \tag{23}$$

$\alpha^{(y)}_{(i),t}$ and $\alpha^{(c)}_{(i),t}$ are used to weigh the features of the Transformer's layer $\mathbf{c}^{(y)}_{(i)}$ and the features of character embeddings $\mathbf{c}^{(c)}_{i}$, which are transformed from $\mathbf{X}^{\text{self}}_{(i)}$ and $\mathbf{X}^{(c)}_{(i)}$, respectively. So the output of gating can computed as follows:

$$\mathbf{g}_{(i),t} = [\alpha^{(y)}_{(i),t}\mathbf{c}^{(y)}_{(i)}, \ldots, \alpha^{(c)}_{(i),t}\mathbf{c}^{(c)}_{(i)}] \tag{24}$$

$$\mathbf{g}_{[(i),\cdot]} = [\mathbf{X}^{\text{gate}}_{(1)}, \cdots, \mathbf{X}^{\text{gate}}_{(n)}], \tag{25}$$

Convolution Following the gating process, two convolutional layers are utilized to capture local features surrounding each predicate and produce the following output:

$$\mathbf{X}^{(\text{conv},l)} = W^{(\text{conv},l)}[\mathbf{X}^{(\text{conv},l-1)}_{(i-(w-1)/2)}; \cdots ; \mathbf{X}^{(\text{conv},l-1)}_{(i+(w-1)/2)}]. \tag{26}$$

Here, l represents the convolutional layer, and w denotes the window size. $\mathbf{X}^{(\text{conv},l)}$ corresponds to the output of the l-th convolutional layer. It is important to note that the input to the first layer is the output of the gating process, denoted as $\mathbf{X}^{\text{gate}}_{(i)}$.

To encompass various aspects of information, we represent the tree as a sequence of predicates. We then encode the rules using an attention mechanism and subsequently employ a tree convolution layer to amalgamate the encoded representation of each node with its ancestors. Suppose we have a sequence of predicates $X_{(1)}, X_{(2)}, \ldots, X_{(P)}$, where P denotes the sequence's length. Within the Abstract Symbolic Tree Reader, we generate four types of embeddings:

Predicate sequence embedding. To encode the information of predicates, we use table-lookup embeddings to present these P predicates as real-valued vectors $\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \ldots, \mathbf{X}_{(P)}$.

Predicate definition embedding. The former embedding represents the predicates as an atomic token and loses the information of the predicates' content, so we introduce predicate definition embedding here. For a symbolic predicate $i : \alpha \to \beta_1, \ldots, \beta_K$, where α is the parent node and $\beta_1, \ldots, \beta_K$ are child nodes (which can be terminal or non-terminal symbols), the index i is the predicate's ID. We encode the predicate content as a vector $\mathbf{X}_c$ using a fully connected layer with inputs being the table-lookup embeddings $\alpha, \beta_1, \cdots, \beta_K$ of the respective symbols, and the sequence is padded to a maximum length. The predicate definition features $\mathbf{X}^{(p)}_{(1)}, \mathbf{X}^{(p)}_{(2)}, \ldots, \mathbf{X}^{(p)}_{(P)}$ are then computed by another fully-connected layer as follows:

$$\mathbf{X}^{(p)}_{(i)} = W^{(p)}[\mathbf{X}_{(i)}; \mathbf{X}_c; \alpha]. \tag{27}$$

Here, $\mathbf{X}_{(i)}$ represents the table-lookup embedding of the predicate $X_{(i)}$ in the symbolic tree, while $\mathbf{X}_c$ represents the content-encoded predicate representation.

Position embeddings. Position embeddings are computed as in Eq. (21), representing the position of each predicate within the sequence $X_{(1)}, X_{(2)}, \ldots, X_{(P)}$.

Depth embeddings. As position embeddings may not capture the position of a predicate within the symbolic tree, we introduce depth embedding. Similar to predicate definition embedding, we represent the depth of the predicate based on its parent node without the content embedding.

These embeddings are input into the reader, and after passing through four distinct sub-layers, they are transformed into $\mathbf{X}^{(ast)}_{(1)}, \mathbf{X}^{(ast)}_{(2)}, \ldots, \mathbf{X}^{(ast)}_{(P)}$. In contrast to the first block, we incorporate a cross-attention sub-layer and transform the convolution layer into a tree convolution layer. The cross-attention sub-layer is informed of the input trajectory, facilitated by multi-head attention. The tree-convolution layer is used to amalgamate information about a node and its ancestors. Traditional Transformer architectures struggle to maintain the relationship between two nodes that are far apart in the rule but close in structure. Further details are shown below.

Cross-Attention Incorporating information from the input trajectory is essential. Therefore, we involve the output of the trajectory reader here. This is achieved through a multi-head attention mechanism, following the same approach as the attention mechanism in the Transformer decoder's attention to its encoder.

Tree Convolution Utilizing a traditional convolutional layer to effectively amalgamate information from a node with its ancestors poses challenges. To address this issue, we treat the symbolic tree as a graph and employ an adjacency matrix denoted as M to represent the directed relationships within the graph. When one predicate $X_{(i)}$ serves as the parent of $X_{(j)}$, it is represented by $M_{(ji)} = 1$. Assuming the outputs of the preceding layer are $\mathbf{X}_{(1)}, \ldots, \mathbf{X}_{(P)}$, we can ascertain the parents of these nodes through matrix multiplication with M:

$$[\mathbf{X}^{(\text{parent})}_{(1)}, \cdots, \mathbf{X}^{(\text{parent})}_{(P)}] = [\mathbf{X}_{(1)}, \cdots, \mathbf{X}_{(P)}]M. \tag{28}$$

Here, $\mathbf{X}^{(\text{parent})}_{(i)}$ represents the parent of the ith node. It's important to note that the father of the root node is the padded root node. The tree-based convolution window applied to the current sub-tree is given by:

$$\mathbf{X}^{(\text{tconv},l)} = f(W^{(\text{tconv},l)}\left[\mathbf{X}^{(\text{tconv},l-1)}; \mathbf{X}^{(\text{tconv},l-1)}; \ldots; \mathbf{X}^{(\text{tconv},l-1)}M^{w-1}\right]). \tag{29}$$

where $W^{(\text{tconv},l)}$ is the wighrs of the convolutional layer. and w is the window size. l is the layer of these convolutional layers. Similar to the convolution layer in the trajectory reader, the input of the first tree convolution layer is the output of the attention layer.

B.2. Decoder of Symbolic Tree Generator

Our final component is a decoder that integrates information from generated logic rules with the state-action trajectory description and predicts the next predicate. It consists of a stack of blocks, each containing several sub-layers. Each sub-layer is surrounded by a residual connection followed by layer normalization. The decoder treats the non-terminal node to be expanded as a query, represented as a path from the root to the node to be expanded. These nodes in the path are represented as real-valued vectors, then a fully connected layer is applied to these vectors and outputs a path of the symbolic tree. Then two attention layers were applied to integrate the outputs of the first block $\mathbf{X}_{(1)}^{(sat)}, \mathbf{X}_{(2)}^{(sat)}, \ldots, \mathbf{X}_{(n)}^{(sat)}$ and the tree convolutional block $\mathbf{X}_{(1)}^{(ast)}, \mathbf{X}_{(2)}^{(ast)}, \ldots, \mathbf{X}_{(P)}^{(ast)}$. Finally, two fully connected layers were used to extract features for prediction.

C. Q-Function with respect to the Anchor Action

To begin, let's isolate the case where $t = 0$ from the summation in the value function and derive the following expression:

$$V(\mathbf{s}) = \mathbb{E}\left[r(\mathbf{s}, \mathbf{A}_0) + \alpha\mathcal{H}\left(\pi^*\left(\mathbf{s}, \cdot\right)\right)\right] + \sum_{t=1}^{\infty}\gamma^t\mathbb{E}\left[r\left(\mathbf{S}_t, \mathbf{A}_t\right) + \alpha\mathcal{H}\left(\pi^*\left(\mathbf{S}_t, \cdot\right)\right)\right]. \tag{30}$$

By the definition of Q-function in the Eq. (13), we can get:

$$V(\mathbf{s}) = \mathbb{E}\left[Q\left(\mathbf{s}, \mathbf{A}\right)\right] + \alpha\mathcal{H}\left(\pi^*\left(\mathbf{s}, \cdot\right)\right). \tag{31}$$

where the expectation is over the action following the optimal policy Eq. (2) in the main paper. Next, by the definition of expectation and information entropy, we can derive

$$\begin{aligned}
V(\mathbf{s}) &= \int_{\mathbf{a}\in\mathcal{A}} Q\left(\mathbf{s}, \mathbf{a}\right)\pi^*\left(\mathbf{s}, \mathbf{a}\right)d\mathbf{a} - \alpha\int_{\mathbf{a}\in\mathcal{A}}\log\left(\pi^*\left(\mathbf{s}, \mathbf{a}\right)\right)\pi^*\left(\mathbf{s}, \mathbf{a}\right)d\mathbf{a} \\
&= \int_{\mathbf{a}\in\mathcal{A}} Q\left(\mathbf{s}, \mathbf{a}\right)\pi^*\left(\mathbf{s}, \mathbf{a}\right)d\mathbf{a} - \alpha\int_{\mathbf{a}\in\mathcal{A}}\frac{Q\left(\mathbf{s}, \mathbf{a}\right)}{\alpha}\pi^*\left(\mathbf{s}, \mathbf{a}\right)d\mathbf{a} \\
&+ \alpha\int_{\mathbf{a}\in\mathcal{A}}\log\left[\int_{\mathbf{a}'\in\mathcal{A}}\exp\left(\frac{Q\left(\mathbf{s}, \mathbf{a}'\right)}{\alpha}\right)d\mathbf{a}'\right]\pi^*\left(\mathbf{s}, \mathbf{a}\right)d\mathbf{a} \\
&= \alpha\log\left[\int_{\mathbf{a}'\in\mathcal{A}}\exp\left(\frac{Q\left(\mathbf{s}, \mathbf{a}'\right)}{\alpha}\right)d\mathbf{a}'\right] \\
&= \alpha\log\left[\int_{\mathbf{a}\in\mathcal{A}}\exp\left(\frac{Q\left(\mathbf{s}, \mathbf{a}\right)}{\alpha}\right)d\mathbf{a}\right].
\end{aligned} \tag{32}$$

Then, we consider a anchor action $\mathbf{a}^A$, and extract $\alpha\log\left[\exp\left(\frac{Q(\mathbf{s},\mathbf{a}^A)}{\alpha}\right)\right]$ from Eq. (32):

$$\begin{aligned}
V(\mathbf{s}) &= \alpha\log\left[\frac{\int_{\mathbf{a}\in\mathcal{A}}\exp\left(\frac{Q(\mathbf{s},\mathbf{a})}{\alpha}\right)d\mathbf{a}}{\exp\left(\frac{Q(\mathbf{s},\mathbf{a}^A)}{\alpha}\right)}\right] + \alpha\log\left[\exp\left(\frac{Q\left(\mathbf{s}, \mathbf{a}^A\right)}{\alpha}\right)\right] \\
&= \alpha\log\left(\frac{1}{\pi^*\left(\mathbf{s}, \mathbf{a}^A\right)}\right) + Q\left(\mathbf{s}, \mathbf{a}^A\right) \\
&= -\alpha\log\left(\pi^*\left(\mathbf{s}, \mathbf{a}^A\right)\right) + Q\left(\mathbf{s}, \mathbf{a}^A\right).
\end{aligned} \tag{33}$$

Then according to the Theorem 2. in (Haarnoja et al., 2017), we have

$$Q\left(\mathbf{s}, \mathbf{a}\right) = r\left(\mathbf{s}, \mathbf{a}\right) + \gamma\mathbb{E}\left[V(\mathbf{s}')\right]. \tag{34}$$

Finally, by taking Eq. (33) into Eq. (34), we get the connection:

$$Q(\mathbf{s}, \mathbf{a}) = r\left(\mathbf{s}, \mathbf{a}\right) + \gamma\mathbb{E}_{\mathbf{s}'}\left[-\alpha\log(\pi^*(\mathbf{s}', \mathbf{a}^A)) + Q(\mathbf{s}', \mathbf{a}^A) \mid \mathbf{s}, \mathbf{a}\right]. \tag{35}$$

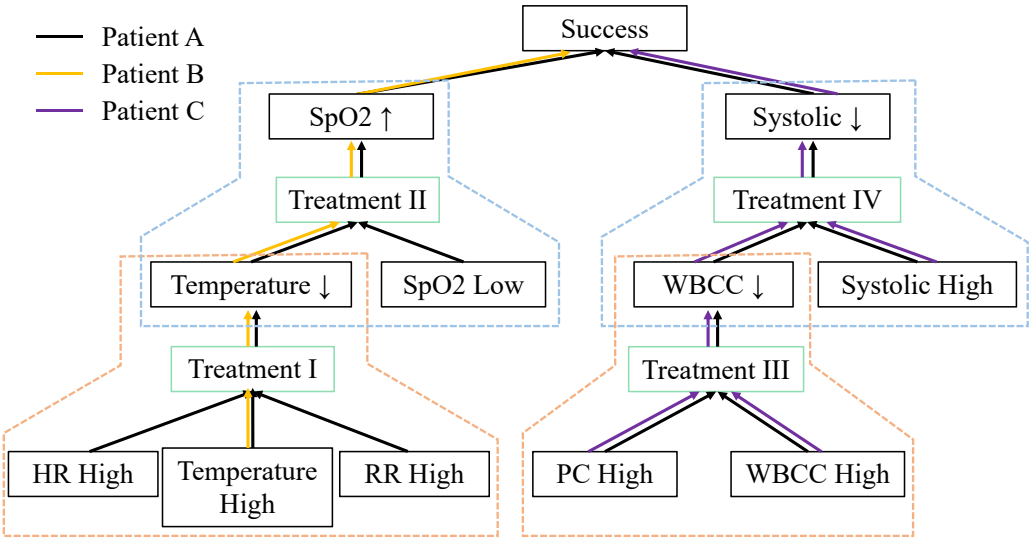

Figure 4. Multidimensional trees of logic strategy generated in MIMIC III dataset. WBCC: White blood cell count; PC: platelets count; RR: Respiratory rate; HR: Heart rate. The leaf nodes present some symptoms of the patient, and each dashed box means that by taking some treatments, the patient may lessen the symptoms. After taking four treatments, the patient fully recovered from all illnesses.

D. Implemental Details

For all the deep learning-based models, we implement them in PyTorch and train them on Ubuntu 16.04 with 3090 GPU. The batch size is set to 50 for all the methods. The dimension of the final hidden state for prediction is set to 256, i.e., $l = 256$. The layer of RNN or Transformer is set to 1 for all the methods unless there is a hierarchical structure. Dropout methods are used for all the models in the final prediction layer unless there is a default setting. The dropout rate is set to 0.5. Adam (Kingma & Ba, 2014) optimizer is used for all the methods. For the learning rate, we use the grid search approach to select the best one for each method according to the validation set.

E. Dataset Description

BlocksWorld. In this environment, the agent will learn how to stack the blocks into certain styles, that are widely used as a benchmark problem in the relational reinforcement learning research. The blocks world environment contains two worlds: the initial world and the target world, each containing the ground and m blocks. The task is to take actions in the operating world and make its configuration the same as the target world. The agent receives positive rewards only when it accomplishes the task and the sparse reward setting brings significant hardness.

MIMIC Dataset. We consider the Medical Information Mart for Intensive Care (MIMIC-III) database and MIMIC-IV (Johnson et al., 2023) database to predict prescription based on 8 observations – temperature, white blood cell count, heart rate, hematocrit, hemoglobin, blood pressure, creatinine, and potassium. MIMIC-III contains 7,493 patients with multiple visits from 2001 to 2012, while there are 85,155 patients in MIMIC-IV with multiple visits from 2008 to 2019. Since there is an overlapped time range between MIMIC-III and MIMIC-IV, we randomly sampled 10,000 patients from MIMIC-IV from 2013 to 2019. By the nature of real-world clinical practice, observation history must be considered by the acting policies – making our decision-making environments partially observable.

Sorting. This task trends to iterative swap elements to sort the array in ascending order. Given a length-m array a of integers, We treat each slot in the array as an object and input their index relations and numeral relations to each model.

Finding Path. Given an undirected graph represented by its adjacency matrix as relations, the algorithm needs to find a path from a start node to the target node. We formulate the shortest path task as a decision-making task. The agent iteratively chooses the next node along the path. In the next step, the starting node will become the next node.

F. Visualization of MIMIC dataset

To Explain patient trajectories, our real-world example studies decision-making within the MIMIC-III dataset in Fig. 4. Let **Patient A** initially have the sepsis symptoms including above-average body temperature, high respiratory rate, high white blood cell count, and so on, progressing towards rehabilitation, progressing towards rehabilitation. Let **Patient B** is diagnosed with high body temperature at the first visit without other symptoms, and **Patient C** is diagnosed with high white blood cell count and high platelets count. Take Patient A as an example, our generated strategy gives a certain diagnosis of sepsis by dividing his symptoms into four parts and therapy them separately. Firstly, considering the high temperature, high respiratory rate, and high heart rate, this strategy suggests Treatment I, such as taking the medicine to decrease the body temperature down to the normal level, following treating the low SpO2 by Treatment II at the following time step. Also, the other symptoms (including high platelets count, high white blood cell count and high systolic) are solved by Treatment III and IV hierarchically. Our policy correctly learns that treatments are only needed until the diagnosis is confirmed. Moreover, the symptoms of Patient B and Patient C are the sub-tree of Patient A, so their treatment strategies are also included in this sepsis treatment system, and other complications can also be treated by following this strategy.

Table 4. Ablation study of removing different portions (denoted as p) of the input predicate. Note that p=10% means that we remove 10% predicates from the framework. We show diagnosis prediction results on MIMIC-III using w-F1 (%) and R@k (%).

Metrics	R@10	R@20
p=0%	29.01(0.11)	38.10(0.08)
p=10%	25.32(0.16)	33.45(0.14)
p=20%	23.21(0.18)	32.81(0.07)
p=30%	19.93(0.08)	31.64(0.11)
p=40%	18.24(0.09)	30.83(0.06)

G. Additional Results

Our generator takes the observed state-action demonstrations as input, and initially encodes them into the symbolic predicate space. So the input predicate has a great influence on the final results. Moreover, we removed different portions (denoted as p) of the input predicate, and the results are shown in the Table 4. we can see that when we remove some input predicates, the performance of R@10 and R@20 drops significantly.

Our findings, detailed in Table 5, demonstrate that our method outperforms others in both performance and reward recovery. Notably, the performance gap between our approach and the Neural Logic Machines (NLM) (Dong et al., 2019) is evident.

For generated rules, we add some explanation about the logic rule and corresponding actions from Blocks World environment. For example, a block should be moved if (1) it is moveable; and (2) there is at least one block below it that does not match the target configuration. Call the desired predicate "ShouldMove(x)". Note that this is only a part of logic rules needed to complete the Blocks World challenge. The learner also needs to figure out where should the block be moved onto. And we can also add some complex predicates to suit in some specific conditions. We list our defined predicates in Table 6 the learned rules relate to the Blocks World here in Table 7.

Table 5. Mean Squared Error (MSE) for reward recovery with a different number of blocks, where m is the number of blocks in the blocks world environment.

Block Num	m=10	m=20	m=30	m=40	m=50
NLM	0.225	0.219	0.138	0.439	0.21
Ours	0.093	0.139	0.124	0.351	0.155

Table 6. Defined predicates for Blocks World. These predicates describe the state of each block and the interaction between different blocks.

Predicate	Explanation
Above(a,b)	The block a is above the block b
Clear(a)	There is no blocks on the block a
Moveable(a)	The block a is movable
SameY(a,b)	the block a and the block b have the same Y-axis
SameX(a,b)	the block a and the block b have the same X-axis
Placeable(a)	the block a is placeable
InitialWorld(a)	There is the initial settings
ShouldMove(a,b)	the block a should be moved onto the block b, and increase the world ID
Onground(a)	The block a is on the ground
SameWorldID(a,b)	The block a and the block b is in the same world
SmallerWorldID(a,b)	The world ID of block a is smaller than the block b

Table 7. Part of learned rules in Blocks World generated by our framework.

Rule 1:	SameXAbove(x, y) $\leftarrow$ SameWorldID(x, y) $\wedge$ SameX(x, y) $\wedge$ Above(x, y)
Rule 2:	Clear(x) $\leftarrow \forall$ y $\neg$ SameXAbove(y,x)
Rule 3:	Moveable(x) $\leftarrow$ Clear(x) $\wedge \neg$ OnGround(x)
Rule 4:	InitialWorld(x) $\leftarrow \forall$ y $\neg$ SmallerWorldID(y, x)
Rule 5:	ShouldMove(x) $\leftarrow$ ($\forall$ y $\neg$ SmallerWorldID(y, x)) $\wedge$ ($\forall$ y $\neg$ SameWorldID(y, x) $\wedge$ SameX(y, x) $\wedge$ Above(y, x)) $\wedge$ ($\exists$ y SameWorldID(y,x) $\wedge$ SameX(x,y)$\wedge$ Above(x,y)) $\wedge \neg$($\exists$ z $\neg$ SameWorldID(y, z) $\wedge$ SameID(y, z) $\wedge$ SameX(y, z) $\wedge$ SameY(y, z))