Learning Two-Step Hybrid Policy for Graph-Based Interpretable Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

We present a two-step hybrid reinforcement learning (RL) policy that is designed to generate interpretable and robust hierarchical policies on the RL problem with graph-based input. Unlike prior deep reinforcement learning policies parameterized by an end-to-end black-box graph neural network, our approach disentangles the decision-making process into two steps. The first step is a simplified classification problem that maps the graph input to an action group where all actions share a similar semantic meaning. The second step implements a sophisticated rule-miner that conducts explicit one-hop reasoning over the graph and identifies decisive edges in the graph input without the necessity of heavy domain knowledge. This two-step hybrid policy presents human-friendly interpretations and achieves better performance in terms of generalization and robustness. Extensive experimental studies on four levels of complex text-based games have demonstrated the superiority of the proposed method compared to the state-of-the-art.

1 Introduction

Recent years have witnessed the rapid developments of deep reinforcement learning across various domains such as mastering board games Silver et al. (2016); Schrittwieser et al. (2020) playing video games Mnih et al. (2015), and chip design Mirhoseini et al. (2021), etc. The larger and complicated architecture of deep reinforcement learning models empowered the capabilities of resolving challenging tasks while brought in significant challenges of interpreting the decision making process of those complex policies Puiutta & Veith (2020). This trade-off between performance and interpretability becomes an inevitable issue when DRL is applied to high stakes applications such as health care Rudin (2019). In this work, we focus on graph-based interpretable reinforcement learning Zambaldi et al. (2018); Waradpande et al. (2020) as the graph representation is expressive in various domains including drug discovery Patel et al. (2020), visual question answering Hildebrandt et al. (2020), and embodied AI Chaplot et al. (2020); Huang et al. (2019), etc. Especially in robotics and embodied AI, the scene graph Zhou et al. (2019) of the environment is a general representation, which is widely used not only in building the environment Puig et al. (2020); Kolve et al. (2017), but also very effective when building the planning module Jia et al. (2022); Min et al. (2021). Another benefit of studying graph-based RL is that the graph structure can provide natural explanations of the decision-making process without the necessity of introducing new programs Verma et al. (2018) or heavy domain knowledge Bastani et al. (2018) for interpretation. Prior works in interpretable reinforcement learning Verma et al. (2018); Madumal et al. (2020); Liu et al. (2018) either works on restricted policy class Liu et al. (2018) that leads to downgraded performance, or the interpretablility Shu et al. (2017); Zambaldi et al. (2018) is limited. Another common issue of interpretable RL is that the provided explanation is generally difficult to comprehend for non-experts Du et al. (2019).

In this work, we seek to obtain a human-understandable interpretation for the decision making process in graph-based tasks. To better illustrate the desired interpretability in graph-based tasks, we borrow the ideas from the visual recognition community, where people interpret how CNN works by analyzing which region gets the most attention, like Zhou et al. (2016); Zhang et al. (2018). Similarly, in graph-based tasks, we want

to **figure out the most crucial sub-graphs (or edges) used in decision making**. [1] We refer to this kind of interpretation as "interpretability" in graph-based RL, not only because this kind of "interpretability" is similar to the one in the visual recognition community, but also because it is easily understood by humans. A crucial sub-graph also has several desirable characteristics of explanation defined by Molnar (2020), e.g., it is contrastive and selected.

To obtain the desired interpreability and resolve the challenges mentioned above, we propose a novel two-step hybrid decision-making process for general deep reinforcement learning methods with graph input. Our approach is inspired by the observation of human decision-making. When confront complicated tasks that involve expertise from multiple domains, we typically identify which domain of expert we would like to consult first and then search for specific knowledge or solutions to solve the problem. Recognizing the scope of the problem significantly reduces the search space of downstream tasks, which leads to a more simplified problem compared to find the exact solution in all domains directly. As an analogy of this procedure, we disentangle a complicated deep reinforcement learning policy into a classification the problem for problem type selection and rule-miner. The classification establishes a mapping from complex graph input into an action type, which handles high-order logical interactions among node and edge representations with graph neural network. The rule miner conducts explicit one-hop reasoning over the graph and provides user-friendly selective explanations Du et al. (2019) by mining several decisive edges. This two-step decision making is essential not only for providing interpretability, but also for generalization and robustness. It is intuitive to see that the simplified classification is easier to achieve better generalization and robustness than the original complicated RL policy. Furthermore, the rule-miner identifying key edges in the graph is much more robust to the noisy perturbations on the irrelevant graph components.

Our contributions can be summarized as below:

- We propose a general framework for two-step hybrid decision making, which is able to combine neural network and rule-based method together to make a decision while yielding human-friendly interpretations.
- We instantiate our proposed framework in the setting of text-based games with graph input by proposing a novel rule miner for knowledge graphs.
- Extensive experiments on several text-based games Côté et al. (2018) demonstrated that the proposed approach achieves a new state-of-the-art performance for both generalization and robustness.

2 Related Works

2.1 Interpretable RL

Instead of using deep RL as a black box, researchers also worked on making deep RL more interpretable. Mott et al. (2019); Zambaldi et al. (2018) introduce some sorts of attention mechanism into policy networks and explain decision making process by analyzing attention weights. Verma et al. (2018) combines program synthesis and PIDs in classic control to solve continuous control problems, while keep the decision making interpretable by explicitly writing the program. Combining symbolic planning and deep RL has similar effects, like Lyu et al. (2019). Tree-based policies Bastani et al. (2018) are favorable in interpretable RL, since they are more human-readable and easy to verify. Bonaert et al. (2020) introduces goal-based interpretability, where the agent produces goals which show the reason for its current actions. However, many existing works of interpretable RL sacrifices the performance for the interpretability. In contrast, our method achieves even better generalization and robustness while providing interpretability.

2.2 Hierarchical RL

Hierarchical RL focuses on decomposing a long-horizon tasks into several sub-tasks, and applying a policy with hierarchical structure to solve the task. Vezhnevets et al. (2017); Frans et al. (2018) try to solve generic

[1] The idea of the crucial sub-graph is also similar to the crucial point set used in PointNetQi et al. (2017), which is essentially a graph neural network.

long-horizon tasks with a two-level policy, where a high-level policy will generate a latent sub-goal for low-level policies, or directly select a low-level policy, and the low-level policy is responsible for generating the final action. In some robot locomotion tasks, people usually manually define the choices of sub-goals to boost the performance and lower the learning difficulty, like Nachum et al. (2018); Levy et al. (2018). In Multi-agent RL, hierarchical structure is a commonly adopted policy structure to provide certain interpertability Gupta et al. (2021); Mahajan et al. (2019); Wang et al. (2020). However, those approaches can not explain how specific decision is associated with the state space as their low-level policy is still a black box policy. Our two-step hybrid policy is reminiscent of the two-level policy architecture in hierarchical RL. Unlike the high-level policy in hierarchical RL is mainly used to generate a sub-goal, the action pruner in our hybrid policy is also used to reduce the number of action candidates. And the motivations of our method and hierarchical RL are also different: hierarchical RL targets at decomposing a long-horizon tasks into several pieces, but our method focuses on interpertability, generalizability and robustness.

2.3 Refactoring Policy

Given a trained neural network-based policies, one may want to refactor the policy into another architecture, so as to improve the generalizibility or interpretablity. And this refactoring process is usually done by imitation learning. The desired architectures of the new policy can be decision tree Bastani et al. (2018), symbolic policy Landajuela et al. (2021), a mixture of program and neural networks Sun et al. (2018), a graph neural networks Mu et al. (2020), or a hierarchical policy Bonaert et al. (2020). In our method, we also use similar techniques to refactor a reinforcement learning policy into a two-step hybrid policy.

2.4 Combine Neural Networks with Rule-Based Models

Neural networks and rule-based models excel at different aspects, and many works have explored how to bring them together, like Chiu et al. (1997); Goodman et al. (1990); Ray & Chakrabarti (2020); Okajima & Sadamasa (2019); Greenspan et al. (1992). There are various ways to combine neural networks with rule-based models. For example, Wang (2019) combines them in a "horizontal" way, i.e., some of the data are assigned to rule-based models while the others are handled by neural networks. In contrast, our work combines neural networks and rule-based models in a "vertical" way, i.e., the two models collaborate to make decisions in a two-step manner.

3 A General Framework for Two-Step Hybrid Decision Making

In this section, we first describe our problem setting including key assumptions and a general framework that formulates the decision-making process in a two-step manner.

We consider a discrete time Markov Decision Process (MDP) with a discrete state space $\mathcal{S}$ and the action space $\mathcal{A}$ is finite. The environment dynamics is denoted as $\mathbf{P} = \{p(s'|s,a), \forall s, s' \in \mathcal{S}, a \in \mathcal{A}\}$. Given an action set $\mathcal{A}$ and the current state $s_t \in \mathcal{S}$, our goal is to learn a policy (π) that select an action $a_t \in \mathcal{A}$ to maximize the long-term reward $\mathbf{E}_\pi[\sum_{i=1}^{T} r(s_i, a_i)]$. Assume we are able to group the actions into several mutual exclusive action types (A_k) according to its semantic meanings. More concretely, the k-th action type $A_k = \{a_k^1, ..., a_k^n\}$ denotes a subset of actions in original action space $A_k \subseteq A$. Then we have $A_1, A_2, .., A_K \subseteq \mathcal{A}, A_i \cap A_j = \emptyset (i \neq j), \cup_{i=1}^{K} A_i = \mathcal{A}$. It is worth noting that the number of action type K is usually much smaller than original actions ($K \ll |\mathcal{A}|$).

Let the policy $\pi = \langle f_p, f_s \rangle$ represented by a hybrid model that consists of action pruner f_p and action selector f_s. The action pruner is used to prune all available action candidates to a single action type, i.e., $f_p(s_i) = k$, where k is the index of chosen action type and $A_k \in \{A_1, A_2, .., A_K\}$. Then the action selector is used to select a specific action given the action type chosen by the action pruner, i.e., $f_s(s_i, A_k) = a_i$, where $a_i \in A_k$ and $k = f_p(s_i)$.

Intuitively, this design intends to disentangle different phases in decision-making process to two different modules. On the one hand, determining the action type typically involves high-order relationships and needs a model with strong expressive power, then neural network is a good candidate in this regard. On the other

hand, selecting an action within a specific action type can resort to rule-based approaches, which is essential in providing strong intrepretability, generalizability and robustness.

Figure 1 shows the overall pipeline of our two-step hybrid decision making. The agent will receive a state s_i at each time step i. At time step i, we will first call the action pruner $f_p(s_i)$ to select the action type A_{k_i}. Then the rule-based action selector $f_s(s_i, A_{k_i})$ will take as inputs the current state s_i and the action type A_{k_i} given by action pruner to select the specific action to be executed in this step.

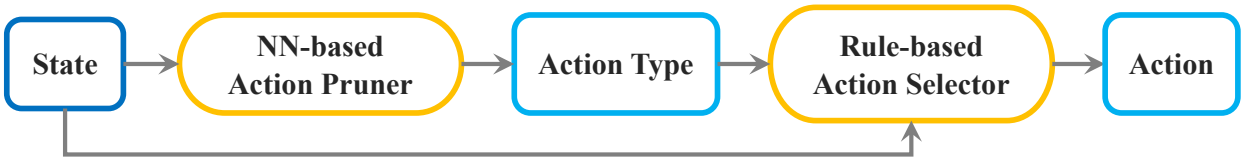

Figure 1: Overall pipeline of the two-step hybrid decision making.

4 Two-Step Hybrid Policy for Text-Based Games with Graph Input

4.1 Overview

In this section, we first discuss our considerations for investigating the interpretability of graph-based tasks and the reasons for using text-based games as our testbed. Firstly, graph is a powerful data structure, and it has been widely adopted in many real-world applications such as drug discovery Patel et al. (2020), visual question answering Hildebrandt et al. (2020), and embodied AI Chaplot et al. (2020); Huang et al. (2019), etc. Meanwhile, the discrete nature and the explicit semantic meaning of graphs (e.g., the relationship of related nodes) provide an easily understandable format for elaborating the decision-making process even for a non-expert. However, the interpretability of graph-based RL tasks is still rarely explored. On the other hand, it is challenging to provide interpretability without any assumption on the action space, even for graph-based RL. We focus on tasks with hierarchical/compositional action spaces as a starting point, which is a common setting in embodied AI. Considering the text-based scene graph has been a general representation of embodied AI tasks Shridhar et al. (2020), we instantiate our proposed framework in the setting of text-based games Côté et al. (2018) with graph input. In text-based games, the agent receives a knowledge graph (as shown in Figure 3) that describes the current state of the environment and the task to be completed, e.g., the task can be represented by several edges like ("potato", "needs", "diced"). Our goal is to learn a policy that maps the input knowledge graph to an action from the provided action set $\mathcal{A}$. Each action $a_j \in \mathcal{A}$ is a short sentence, e.g., "take apple from fridge".

In this setting, we use graph neural networks (GNNs) as the action pruner f_p, and a rule-based model as the action selector f_s. We will elaborate the details of the action pruner and the action selector in Sec 4.4 and Sec 4.5, respectively. A brief description of GNNs can be found in Sec 4.2.

Training the GNN-based action pruner and the rule-based action selector by reinforcement learning is nontrivial since the whole pipeline is not end-to-end differentiable. Therefore, we propose to learn both models separately from a demonstration dataset, and the demonstration dataset can be obtained by a trained reinforcement learning agent. We will elaborate the details in Sec 4.3.1. This process is inspired by existing works like Bastani et al. (2018); Landajuela et al. (2021); Sun et al. (2018); Mu et al. (2020), where policies trained by reinforcement learning can be refactorized into other forms.

4.2 Backgrounds on Graph Neural Networks

We briefly describe a commonly used class of GNNs that encompasses many state-of-art networks, including GCN Kipf & Welling (2016), GAT Veličković et al. (2017), GIN Xu et al. (2018), etc. The class of GNNs generalizes the message-passing framework Gilmer et al. (2017) to handle edge attributes. Each layer of it

can be written as

$$h_v^{l+1} = f_\theta^l(h_v^l, \{x_e : e \in \mathcal{N}_E(v)\}, \{h_{v'}^l : v' \in \mathcal{N}_V(v)\})$$

, where h_v^l is the node feature vector of node v at layer l. $\mathcal{N}_V(v)$ and $\mathcal{N}_E(v)$ denote the sets of nodes and edges that are directly connected to node v (i.e. its 1-hop neighborhood). f_θ^l is a parameterized function, which is usually composed of several MLP modules and aggregation functions (e.g. sum/max) in practice.

4.3 Preparing Datasets for Learning Two-Step Hybrid Policy

4.3.1 Demonstration Acquisition

Given a set of interactive training environments, the goal of this step is to obtain a demonstration dataset, which contains state-action pairs from a policy achieving high reward in the training environment. Specifically, we aim at generating a demonstration dataset $\mathcal{D} = \{(s_i, \pi(s_i))\}_{i=1}^N$, where s_i is the input state (a knowledge graph in our cases) from the training environments, and $\pi(s_i)$ is the output of demonstration policy on the state s_i. The representation of $\pi(s_i)$ is flexible: it can be an action, the logits, or any latent representation, which indicates the demonstration action distribution. This dataset will be used for learning our two-step hybrid policy.

We refer the policy used to generate demonstration as *teacher policy*. It can be obtained in any way, as long as it provides reasonable and good supervision on how to solve the task in the training environments, and it is not necessary to have strong interpretablity, generaliziabilty or robustness. Therefore, deep reinforcement learning is a great tool to learn such a policy.

After obtaining the teacher policy, we use it to interact with the training environments to collect states and label them with the output of the teacher policy. During the interaction, we add action noise to perturb the state distribution in the demonstration dataset. A more diversified state distribution is beneficial to the performance of our two-step hybrid approach.

4.3.2 Action Grouping and Demonstration Splits

As mentioned above, the action pruner is responsible for selecting the correct type of actions for the current state, so that the action candidates will be pruned to a smaller set. Then, a rule-based model will work on this specific action type to select the action to be executed. Therefore, we need to label each state with the action type. Since the action for each state is already in the dataset, we just need to convert each action into an action type. In other words, we need to group all possible actions into several action types.

In text-based games, each action is a short sentence with semantic meaning, e.g., “take apple from fridge". Intuitively, we can group the actions by their semantic meaning. For example, if the actions are “take apple from fridge", “slice potato with knife", “dice cheese with knife", “cook carrot with oven", and “cook onion with stove", we can group them into three types: “take" actions, “cut" actions, and “cook" actions. And we make sure all the actions with the same action type follow the same template, e.g., the template of “take" action can be written as “take <u>object</u> from <u>receptacle</u>", where <u>object</u> and <u>receptacle</u> can be instantiated by appropriate words.

Grouping actions by semantic meanings also aligns with our motivation: 1) After we obtained K group of actions, we substantially alleviate the workload of neural networks from learning a policy that maps states to an action in a varying size and high-dimensional action space to a simplified classification with a much smaller number of classification categories. This simplification could greatly reduces the sample-complexity. 2) Furthermore, this step also creates a much smaller search space to enable the rule-miner to search within a feasible candidates set.

Formally, given a dataset $\mathcal{D} = \{(s_i, \pi(s_i))\}_{i=1}^N$, we will generate an action type for each state, policy output $(s_i, \pi(s_i))$ pair. The action type k_i is determined by $k_i = h(\pi(s_i))$, where $h(\pi(s_i))$ is a domain-specific function which takes the semantic meaning of the action into consideration. As a result of the clustering process, we will get a new demonstration dataset with action types $\mathcal{D} = \{(s_i, \pi(s_i), k_i)\}_{i=1}^N$. In addition, we can also split the demonstration dataset based on the action types to get K subset of the original demonstration dataset, $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K\}$, where $\mathcal{D}_k = \{s_i, \pi(s_i) | h(\pi(s_i)) = k\}$.

4.4 GNN-based Action Pruner

The action pruner needs to output an action type based on the input knowledge graph, so it is essentially a classifier. Given the demonstration dataset $\mathcal{D} = \{(s_i, k_i)\}_{i=1}^{N}$ obtained in the last step, we want to train a classifier $f_p(s;\theta) = k$, where $k \in \{1, 2, ..., K\}$ is an action type. This is a conventional classification problem which can be solved by minimizing cross entropy loss:

$$\theta = \arg\min_{\theta} -\sum_{i}\sum_{j=1}^{K} k_i^j \log(f_\theta^j(s_i)),$$

where $f_\theta(s_i)$ outputs a probability distribution over the K action types, $f_\theta^j(s_i)$ denotes the j-th action type's probability. $k_i^j \in \{0, 1\}$ denotes denotes whether the action type j was chosen in the demonstration dataset at state i.

4.5 Rule-Based Action Selector

4.5.1 Abstract Supporting Edge Sets

When the input is a knowledge graph, the action is naturally strongly correlated with some critical edges. For example, (“potato", “needs", “diced") and (“potato", “in", “player") can lead to the action “dice potato". We refer those decisive edges correlating to an action as the *supporting edge set* of this action. Since we have grouped actions by their semantic meanings, actions within each action type are actually supported by similar edges. For example, “dice potato" is supported by (“potato", “in", “player"), and “dice apple" is supported by (“apple", “in", “player"). As mentioned in Sec 4.3.2, each action type comes with an action template like “dice object". Based on the action template, we can perform some sorts of abstraction. For example, given an input knowledge graph labeled with action “dice apple", we can replace all the “apple" appearing in the graph edges and the action with an abstract name “object". Then we can say the action “dice object" is essentially supported by (“object", “in", “player"), where the two “object" should be instantiated by the same word. Under this kind of abstraction, different actions within the same action type can share a same *abstract supporting edge set* which contains edges with abstract names.

The *abstract supporting edge set* indicates the decisive edges for an action type, and it can be instantiated for each specific action. For example, to check whether the action “dice apple" should be executed, the abstract edge (“object", “in", “player") will be instantiated to edge (“apple", “in", “player"). Then, the existence of (“apple", “in", “player") in input knowledge graph becomes an evidence for selecting the action “dice apple". We aim at finding an abstract supporting edge set for each action type, and it will be used during inference.

4.5.2 Mine Abstract Supporting Edge Sets from Demonstrations

Finding the abstract supporting edge set for each action type is actually a rule mining process. There are several off-the-shelf rule miners like FP-Growth Han et al. (2004), Apriori Agrawal et al. (1994), Eclat Zaki (2000), etc. , but they are not designed for knowledge graphs. Thus, we propose a simple yet effective rule miner for our setting to discover the supporting edge sets.

To find the **A**bstract **S**upporting **E**dge set ASE(A_k) for each action type A_k, we designed a numerical statistic that is intended to reflect the importance of an edge when taking an action $a \in A$, and this numerical statistic is inspired by tf-idf Rajaraman & Ullman (2011). Formally, for action type A_k, we have a corresponding subset of demonstration dataset $\mathcal{D}_k = \{s_i\}$ (ignoring π_i here) where all the actions are within the action type A_k. Under the abstraction mentioned above, we can count the edge frequency for every (abstract) edge e in $\mathcal{D}_k$:

$$freq_k(e) = \frac{|\{s_i | s_i \in \mathcal{D}_k, e \in s_i\}|}{|\mathcal{D}_k|}.$$

Similarly, we can also count the edge frequency for the entire demonstration dataset:

$$freq(e) = \frac{|\{s_i | s_i \in \mathcal{D}, e \in s_i\}|}{|\mathcal{D}|}.$$

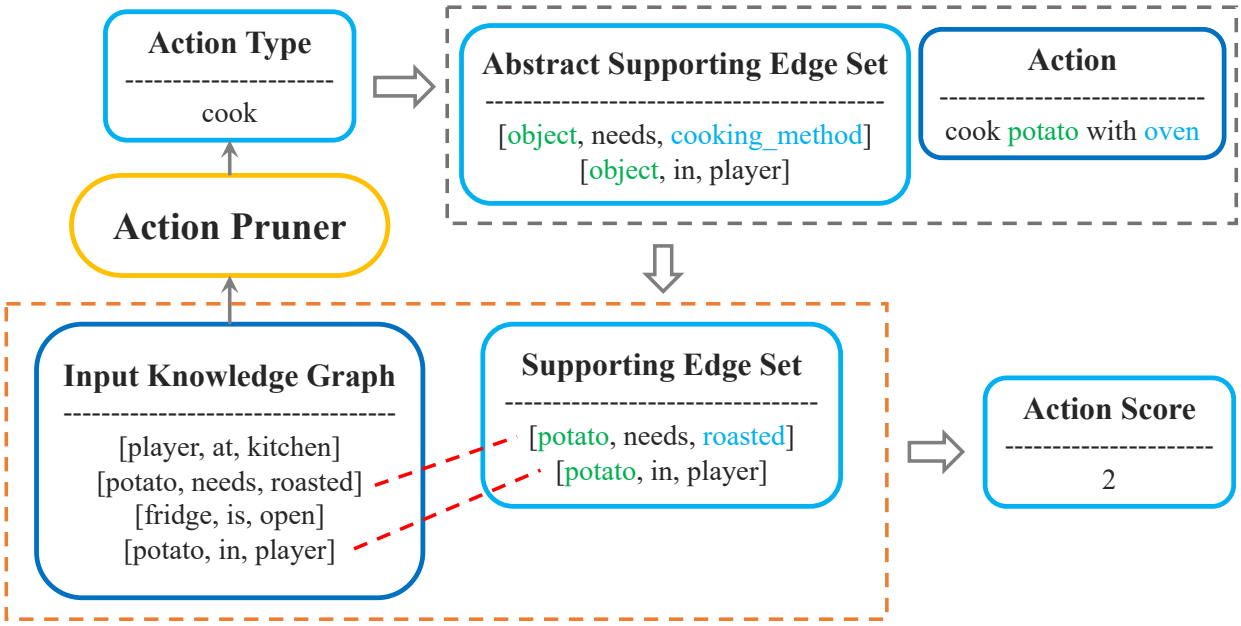

Figure 2: An example of using supporting edge set to score an action. Based on the knowledge graph, the action pruner first predict the action type (e.g., cook) that needs to be taken at current state, then instantiate the abstract supporting edge set to a concrete supporting edge set. Comparing the input knowledge graph with the supporting edge set, we can compute action score for each action and select the action with highest score accordingly.

And we can define an importance score of an edge w.r.t to the action type A_k:

$$I_k(e) = freq_k(e) \cdot \log(\frac{1}{freq(e)}),$$

where the term $freq_k(e)$ is similar to the term-frequency (tf), and the term $log(\frac{1}{freq(e)})$ is similar to the inverse document frequency (idf).

Then we can get the ASE(A_k) by selecting the edges with the importance higher than a threshold, i.e., ASE$(A_k) = \{e|I_a(e) > \tau\}$, where τ is a hyperparameter shared across all action types.

4.5.3 Inference Based on Supporting Edges

During inference, we use the supporting edge sets to score each action within the action type provided by action pruner, and select the action with the highest score. Figure 2 shows a concrete example of using supporting edge set to score an action.

Firstly, the action pruner outputs an action type based on the input knowledge graph, and we can retrieve the abstract supporting edge set of this action type. Secondly, given an action within the action type, we can instantiate the abstract supporting edge set to a specific supporting edge set by replacing the abstract names by the concrete words according to the action, e.g, ("object", "in", "player") will be instantiated to ("potato", "in", "player") if the action is "cook potato with oven". Finally, we can compare the input knowledge graph with the supporting edge set of each action to figure out which supporting edge set is best covered by the input knowledge graph. The number of overlapped edges between the supporting edge set and the the input knowledge graph will be regarded as the score of the action.

Formally, the inference process of our rule-based action selector can be described as follows

$$f_s(s, A) = \arg\max_{a \in A} |s \cap \text{SE}(a)|,$$

where SE(a) is the supporting edge set associated with action a, which is obtained by instantiating the abstract supporting edge set of action type A.

5 Experiments

5.1 Dataset Setup

Difficulty Level	Recipe Size	Number of Locations	Need Cut	Need Cook	Number of Action Candidates	Number of Objects	Number of Sub-tasks
1	1	1	Yes	No	11.5	17.1	4
2	1	1	Yes	Yes	11.8	17.5	5
3	1	9	No	No	7.2	34.1	3
4	3	6	Yes	Yes	28.4	33.4	11

Table 1: TextWorld games statistics (averaged across all games within a difficulty level). The games and statistics are generated by Adhikari et al. (2020).

We evaluate our method on TextWorld, which is a framework for designing text-based interactive games. More specifically, we use the TextWorld games generated by GATA Adhikari et al. (2020). In these games, the agent is asked to cook a meal according to given recipes. It requires the agent to navigate among different rooms to locate and collect food ingredients specified in the recipe, process the food ingredients appropriately, and finally cook and eat a meal.

The state received by the agent is a knowledge graph describing all the necessary information about the game. All the nodes and edges in the knowledge graph are represented in text. Figure 3 shows a partial example of input knowledge graph. The actions are also represented in text. Note that the number of available actions vary from state to state, so most of of existing network architecture used deep reinforcement learning cannot be directly used here.

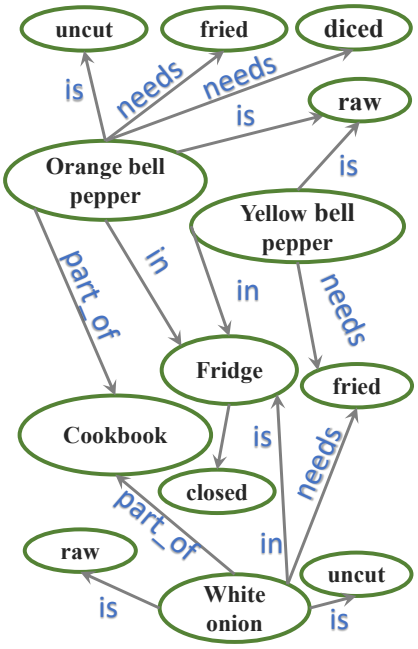

Figure 3: Visualization of a part of an input knowledge graph in TextWorld.

The games have four different difficulty levels, and each difficulty level contains 20 training, 20 validation, and 20 test environments, which are sampled from a distribution based on the difficulty level. The higher the difficulty levels are, the more complicated recipe will be and the more rooms food ingredients will be distributed among. For easier games, the recipe might only require a single ingredient, and the world is limited to a single location, whereas harder games might require an agent to navigate a map of six locations to collect and appropriately process up to three ingredients. Statistics of the games are shown in Table 1. For evaluating model generalizability, we select the top-performing agent on validation sets and report its test scores; all validation and test games are unseen in the training set.

5.2 Implementation Details

5.2.1 Process Knowledge Graphs by GNNs

Graph neural networks (GNNs) serve as backbone networks in both our RL teacher policy and the action pruner in our two-step hybrid policy. More specifically, we use the Relational-GCN to take edge attributes into consideration, and the Relational-GCN is also the backbone architecture used in GATA Adhikari et al. (2020), which is the baseline method in the TextWorld games we used. Actually, the choice of the network

architecture is not very important in our method. This is because both the RL teacher policy and the action pruner are only used to fit the training environments or the demonstration dataset. We did not really expect them to have strong interpretability, because this is the job of the rule-based action selector. Therefore, most graph-based networks with sufficient capacity should be good in our cases. We will also perform an ablation study about network architecture in Sec. 5.4.

The relations and nodes of the input knowledge graphs are initially represented in text (words or short sentences). We use fastText Mikolov et al. (2017), pre-trained on Common Crawl, to convert each token to an embedding, and then average all the token embeddings to obtain the mean fastText embedding of the entire text string. These mean fastText embeddings are pre-computed and fixed during training. For each relation and node, we append their corresponding mean fastText embedding with a trainable embedding vector. The concatenated embeddings are then used as the numerical representation of each relation and node. Finally, a graph with numerical node/relation features is passed through GNNs to get the latent embedding of the entire input knowledge graph.

5.2.2 Demonstration Acquisition

To collect demonstration dataset, we first train a teacher policy by DQN Mnih et al. (2015) in the training environments, which can converge to a near-optimal solution. To adopt variable number of available actions, we let the policy network take the state and an available action as input and output a score for this action. And we can select the action with maximum score as the final output.

The trained teacher policy is used to collect 300K samples through the interaction with the environment, and label them with the taken actions, as illustrated in Sec 4.3.1. When collecting the demonstration dataset, we use ϵ-greedy exploration strategy to increase the diversity of states.

5.3 Results

5.3.1 Interpretability

	Action Type: Cut				Example Action		
Action	verb object with knife				slice potato with knife		
	Head	Relation	Tail	Importance	Head	Relation	Tail
Discovered Supporting Edges	object	in	player	1.7004	potato	in	player
	object	needs	verb_passive	1.6659	potato	needs	sliced
	object	part_of	cookbook	1.0058	potato	part_of	cookbook
	object	is	uncut	0.9837	potato	is	uncut

Table 2: Discovered abstract supporting edges of the “cut" action type in difficulty level 1 of TextWorld environments. The importance of each edge is computed by the metric mentioned in Sec 4.5.2. The right column gives an example action and the corresponding supporting edge set, which is obtained by instantiating the abstract supporting edge set. The underlined words are abstract names for some certain words. For example, “verb" in action can be instantiated to “slice", “dice" or “chop", and “verb_passive" can be instantiated to “sliced", “diced" or “chopped" accordingly.

The interpretablity of our two-step policy is two-fold: 1) the transparent two-step decision making process; 2) the rule-based models make decisions in a way which is easy to interpret by human.

Table 2 shows some representative rules discovered by our rule miners. We observed that all of the four discovered abstract supporting edges are indeed prerequisites of the “cut" actions. In particular, the abstract supporting edge (“object", “needs", “verb_passive") is a crucial prerequisite for the agent to select the correct verb in the “cut" actions. For example, if the input knowledge graph contains the edge (“potato", “needs", “sliced"), then the action “slice potato with knife" will get one more score than others because this edge is in the supporting edge set of action “slice potato with knife".

Since we clearly see the rule-based model makes decisions based on these rules, it is not hard to check whether this model works in a correct way. In this way, the agent can certainly select the correct "cut" action even in unseen test environments.

5.3.2 Generalization

	Training				Test			
Difficulty	1	2	3	4	1	2	3	4
GATA-GTF	98.6	58.4	95.6	36.1	83.8	53	33.3	23.6
Vanilla RL	100	100	98.3	100	83.8	68	50	30.9
Ours	100	100	100	65.5	**100**	**100**	**51.7**	**49.7**

Table 3: Evaluation results on both training environments and test environment in TextWorld. The numbers show the agent's normalized scores.

We compare with two baselines: GATA-GTF and vanilla RL. GATA-GTF is a variant from GATA Adhikari et al. (2020), and it uses the same ground-truth graph input as us. GATA-GTF processes the input graphs by relational graph convolutional networks (R-GCNs) Schlichtkrull et al. (2018), and the whole pipeline is trained by DQN Mnih et al. (2015). Vanilla RL is essentially the same method with GATA-GTF, but implemented by ourselves. With better implementation and careful hyperparameter tuning, it performs better than the implementation released by the GATA authors. And vanilla RL also serves the teacher policy which used to generate demonstrations for our method.

Table 3 shows the normalized scores of different methods on both training environments and test environment in TextWorld. The result of GATA-GTF is obtained from its paper Adhikari et al. (2020). In all the environments, our agent achieves better generalization performance the vanilla RL (which is our RL teacher) and GATA-GTF baselines. Our agent can generalize pretty well to the unseen test environments in all difficulty levels, while the performance of GATA-GTF and vanilla RL performs poorly in unseen test environments.

5.3.3 Robustness

As mentioned above, our method also aims at robustness. To evaluate the robustness of our models, we add different levels of noises to input knowledge graphs and regard the performance of the agents under noisy inputs. In this paper, we define the noise on a knowledge graph as adding additional edges to the graph or dropping existing edges in the graph. Formally, we add noise at (k, p)-level to input knowledge graph in the following way:

1. Add edges: $\lceil k * |E| \rceil$ additional edges will be randomly generated and added to the graph, where E is the edge set of the graph. For each edge h_i, t_i, r_i, head node h_i and tail node t_i are sampled from V_{all} and relation r_i is sampled from R_{all}, where V_{all} means the set of all possible nodes and R_{all} is the set of all possible relations.

2. Drop edges: $\lceil p * |E| \rceil$ edges will be dropped from the graph. Note that only original edges will be dropped, and the randomly added edges will not be dropped.

Table 4 shows the performance of vanilla RL and our method under noisy input graphs generated in the above mentioned way. Note that we test robustness in training environments instead of test environments, because the performance of both agents in test environments are not good enough and examining robustness of poor-performed agents does not make too much sense.

We observed that in difficulty level 1, both agents are very robust under the noisy inputs. However, in difficulty level 2 & 3 & 4, the performance of the RL agents are hurts a lot by the input noises, especially in the difficulty level 4. In contrast, the agents obtained by our method are still performs pretty good and the performance drops significantly less than the RL agents.

Noise		Difficulty							
		1		2		3		4	
Add	Drop	RL	Ours	RL	Ours	RL	Ours	RL	Ours
0.2	0	**100(0%)**	**100(0%)**	**100(0%)**	**100(0%)**	96(-1%)	**98(-1%)**	38(-61%)	**64(-1%)**
0.2	0.03	**100(0%)**	96(-3%)	81(-19%)	**98(-1%)**	80(-18%)	**98(-1%)**	41(-58%)	**51(-21%)**
0.2	0.06	**100(0%)**	92(-7%)	**93(-6%)**	82(-18%)	86(-11%)	**93(-6%)**	**44(-55%)**	41(-36%)
0.4	0	**100(0%)**	**100(0%)**	85(-15%)	**100(0%)**	65(-33%)	**91(-8%)**	19(-80%)	**58(-11%)**
0.4	0.03	**100(0%)**	**100(0%)**	77(-23%)	**90(-9%)**	71(-27%)	**91(-8%)**	20(-79%)	**53(-18%)**
0.4	0.06	**100(0%)**	96(-3%)	78(-22%)	**83(-16%)**	65(-33%)	**96(-3%)**	19(-80%)	**49(-24%)**
0.6	0	**100(0%)**	**100(0%)**	85(-15%)	**100(0%)**	76(-22%)	**93(-6%)**	8(-91%)	**59(-9%)**
0.6	0.03	**100(0%)**	97(-2%)	75(-25%)	**83(-16%)**	61(-37%)	**86(-13%)**	9(-90%)	**39(-39%)**
0.6	0.06	**100(0%)**	91(-8%)	76(-24%)	**80(-20%)**	60(-38%)	**96(-3%)**	10(-89%)	**46(-29%)**

Table 4: Robustness analysis of our method and vanilla RL baseline. We evaluate the performance of agents on **training environments** under different noise levels, e.g., "add 0.2 drop 0.03" means we randomly add 20% additional edges while randomly dropping 3% existing edges in graph. The numbers out of parentheses are normalized scores and the numbers in parentheses are relative performance change comparing to the performance without input noise. The bold numbers indicate which method performs better in each setting.

5.4 Ablation Study

In this section, we study the contributions of different modules in our method. Our proposed method mainly consists of three modules: 1) an RL teacher policy, 2) a GNN-based action pruner, 3) a rule-based action selector. Therefore, we replace each module with other alternatives and see how the generalization performance is influenced. All the experiments below are performed on the TextWorld games with difficulty level 1.

5.4.1 RL teacher policy

We first study how different kinds of teacher policies affects the performance. We consider the following three alternatives to our RL teacher policy: 1) an RL teacher policy with different backbone network: Graph Transformer Network (GTN) Yun et al. (2019); 2) a manually written ground truth policy which can perfectly solves the tasks; 3) a random policy. Note that we choose ground truth policy and random policy as teachers because they represent two extreme cases: a very good teacher and a very bad teacher.

Method Variant	Training	Test
Ours (backbone is GCN)	100	100
Use GTN as backbone of teacher	100	100
Use ground truth policy as teacher	100	100
Use random policy as teacher	8.8	15.0

Table 5: Evaluation results on both training environments and test environment in TextWorld difficulty level 1. The numbers show the agent's normalized scores.

Table 5 shows the results of this ablation study. From this table, we can observe the following things:

- It does not make much difference by switching the backbone of the RL teacher policy from GCN to GTN. This verifies our expectation: any graph-based network should be good as long as it has sufficient capacity for the training environment / data, because the job of the RL teacher is only to provide a demonstration dataset.

- Using ground truth policy also does not really affect the performance of our method. Since our RL teacher policy actually achieves almost perfect performance on the training environments, the demonstrations generated by our RL teacher and the ground truth policy are almost equally good.
- Using a random policy does not work at all. This also meets our expectation that a good demonstration dataset is very important for learning the action pruner and the rule-based action selector.

5.4.2 GNN-based action pruner

In this section, we mainly assess the influences of the network architectures used in the GNN-based action pruner. Specfically, we replace the backbone of our action pruner with Graph Transformer Network (GTN) Yun et al. (2019). And we also study the case where the RL teacher also uses GTN as the backbone.

Method Variant	Training	Test
Ours (GCN teacher, GCN action pruner)	100	100
GCN teacher, GTN action pruner	100	100
GTN teacher, GTN action pruner	95	100

Table 6: Evaluation results on both training environments and test environment in TextWorld difficulty level 1. The numbers show the agent's normalized scores.

From the Table 6, we can see that the architecture changes have minimal influence on the performance. This verifies our expectation again: any graph-based network should be good as long as it has sufficient capacity for the training environment / data. Though the "GTN teacher, GTN action pruner" variant does not get full scores on the training environment, the performance gap is very small.

5.4.3 Rule-based action selector

Finally, we study whether a rule-based action selector is crucial by replacing it with a network-based action selector. The resulting policy is somehow like a mixture of experts, where the gating function and the experts are all networks.

Method Variant	Training	Test
Ours (rule-based action selector)	100	100
GCN-based action selector	100	70

Table 7: Evaluation results on both training environments and test environment in TextWorld difficulty level 1. The numbers show the agent's normalized scores.

From the Table 7, we can see that the network-based action selector does a good job in the training environments but performs poorly on the test environments. This is because the network-based action selector overfits the training data instead of really learning the underlying rules of decision making. In contrast, our rule-based action selector learns the real underlying rules that generalize well. Therefore, the rule-based action selector is a crucial component of our method.

6 Conclusion

In this work, we propose a two-step hybrid policy for graph-based reinforcement learning. The two-step hybrid policy disentangles complicated black-box deep reinforcement learning policy into an action pruner and an action selector. The joint effort of two modules can not only generate human-friendly explanations of each decision, but also provide significantly better performance in terms of generalization and robustness. We conduct comprehensive experiments on the representative text-based environments Côté et al. (2018) to demonstrate the effectiveness of our approach.

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33, 2020.

Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pp. 487–499. Citeseer, 1994.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 2499–2509, 2018.

Gregory Bonaert, Youri Coppens, Denis Steckelmacher, and Ann Nowe. Explainable reinforcement learning through goal-based interpretability. 2020.

Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12875–12884, 2020.

Chih-Chou Chiu, Ling-Jing Kao, and Deborah F Cook. Combining a neural network with a rule-based expert system approach for short-term power load forecasting in taiwan. *Expert Systems with Applications*, 13(4): 299–305, 1997.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pp. 41–75. Springer, 2018.

Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Rodney M Goodman, Chuck Higgins, John Miller, and Padhraic Smyth. A rule-based approach to neural network classifiers. In *International Neural Network Conference*, pp. 886–889. Springer, 1990.

Hayit K Greenspan, Rodney Goodman, and Rama Chellappa. Combined neural network and rule-based framework for probabilistic pattern recognition and discovery. 1992.

Tarun Gupta, Anuj Mahajan, Bei Peng, Wendelin Böhmer, and Shimon Whiteson. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 3930–3941. PMLR, 2021.

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

Marcel Hildebrandt, Hang Li, Rajat Koner, Volker Tresp, and Stephan Günnemann. Scene graph reasoning for visual question answering. *arXiv preprint arXiv:2007.01072*, 2020.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8565–8574, 2019.

Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav Sukhatme. Learning to act with affordance-aware multimodal neural slam. *arXiv preprint arXiv:2201.09862*, 2022.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, pp. 5979–5989. PMLR, 2021.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2018.

Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 414–429. Springer, 2018.

Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 2493–2500, 2020.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 32, 2019.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*, 2021.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems*, pp. 12329–12338, 2019.

Tongzhou Mu, Jiayuan Gu, Zhiwei Jia, Hao Tang, and Hao Su. Refactoring policy for compositional generalizability using self-supervised object proposals. *Advances in Neural Information Processing Systems*, 33, 2020.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 31:3303–3313, 2018.

Yuzuru Okajima and Kunihiko Sadamasa. Deep neural networks constrained by decision rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2496–2505, 2019.

Lauv Patel, Tripti Shukla, Xiuzhen Huang, David W Ussery, and Shanzhi Wang. Machine learning methods in drug discovery. *Molecules*, 25(22):5277, 2020.

Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B Tenenbaum, Sanja Fidler, and Antonio Torralba. Watch-and-help: A challenge for social perception and human-ai collaboration. *arXiv preprint arXiv:2010.09890*, 2020.

Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pp. 77–95. Springer, 2020.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

Anand Rajaraman and Jeffrey David Ullman. Mining of massive datasets: Data mining (ch01). *Min. Massive Datasets*, 18:114–142, 2011.

Paramita Ray and Amlan Chakrabarti. A mixed approach of deep learning method and rule-based method to improve aspect level sentiment analysis. *Applied Computing and Informatics*, 2020.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning*, pp. 4790–4799. PMLR, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 3540–3549. PMLR, 2017.

Tong Wang. Gaining free or low-cost interpretability with interpretable partial substitute. In *International Conference on Machine Learning*, pp. 6505–6514. PMLR, 2019.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv:2010.01523*, 2020.

Vikram Waradpande, Daniel Kudenko, and Megha Khosla. Graph-based state representation for deep reinforcement learning. *arXiv preprint arXiv:2004.13965*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390, 2000.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Deep reinforcement learning with relational inductive biases. 2018.

Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8827–8836, 2018.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

Yang Zhou, Zachary While, and Evangelos Kalogerakis. Scenegraphnet: Neural message passing for 3d indoor scene augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7384–7392, 2019.