LLM-AUGMENTED SYMBOLIC RL WITH LANDMARK-BASED TASK DECOMPOSITION

Alireza Kheirandish, Duo Xu, Faramarz Fekri

School of Electrical and Computer Engineering, Georgia Institute of Technology

ABSTRACT

One of the fundamental challenges in reinforcement learning RL is to take a complex task and be able to decompose it to subtasks that are simpler for the RL agent to learn. In this paper, we report on our work that would identify subtasks by using some given positive and negative trajectories for solving the complex task. We assume that the states are represented by first-order predicate logic using which we devise a novel algorithm to identify the subtasks. Then we employ a Large Language Model (LLM) to generate first-order logic rule templates for achieving each subtask. Such rules were then further fined tuned to a rule-based policy via an Inductive Logic Programming (ILP)-based RL agent. Through experiments, we verify the accuracy of our algorithm in detecting subtasks which successfully detect all of the subtasks correctly. We also investigated the quality of the common-sense rules produced by the language model to achieve the subtasks. Our experiments show that our LLM-guided rule template generation can produce rules that are necessary for solving a subtask, which leads to solving complex tasks with fewer assumptions about predefined first-order logic predicates of the environment.

Index Terms— Reinforcement Learning, Large Language Model, Inductive Logic Programming, Contrastive Learning

1. INTRODUCTION

In the realm of Reinforcement Learning (RL), strategically using landmarks and subtasks is a key technique for managing complex tasks [1]. This method systematically breaks down daunting challenges into smaller, achievable goals and clear pathways, making intricate tasks more manageable [2]. To complete a complex task, we must visit certain specific states—referred to as landmarks—that contain essential information for successfully accomplishing the task. Landmarks act as critical milestones that facilitate effective decision-making and enhance structured, efficient problem-solving strategies [3]. These landmarks constitute essential milestones about the task, crucial for achieving the goal. For example, a landmark could be possessing specific combinations of objects, arriving at a particular location, or visiting certain places in a specific order [4]. We define each of these landmarks that are necessary to complete a task as a subtask. Subtasks can consist of either the entire state or a subset of the state. Subtasks are particularly valuable in complex environments where a straightforward trajectory to the goal is not readily apparent or where the policy required to solve intricate tasks is complex, making straightforward solutions challenging.

While other works have addressed identifying landmarks through reward-centric algorithms [5, 6], our algorithm uses state trajectories labeled only with a single indicator of whether the trajectory was successful in completing the task. This approach is crucial in environments with sparse and non-interpretable rewards. For this purpose, we have used contrastive learning [7] with the logic-predicate representation of the states as its input.

Recently, there has been significant interest in symbolic RL in general [8, 9]. Symbolic RL has the advantage of being human interpretable and also more generalizable to new environments. In particular, as a special type of symbolic RL, inductive logic programming (ILP)-based RL agents [10, 11, 12] have utilized differentiable rule learners known as ∂ILP [13, 14] to form logic-based policies.

Recently, an RL method denoted as NUDGE [15] was proposed using ILP to generate interpretable policy as a set of weighted rules. We will be using NUDGE framework as the ILP engine for further fine tuning our rules generated by LLM for the subtasks.

When processing an input state, the NUDGE system identifies entities and their interactions, transforming raw states into logical representations. In the realm of first-order logic, a predicate functions as a Boolean operation on terms, which are defined as objects or variables. We establish our subtasks using distinct combinations of predicates, thereby facilitating the creation of interpretable subtasks. Our empirical findings indicate that creating subtasks does not require detailed predicates from the environment.

The advent and evolution of Large Language Models (LLMs) have sparked significant interest due to their ability to utilize common sense knowledge and process information in natural language, mirroring real-world understanding [16]. There are recent research works that elaborate LLMs as either auxiliary supports or principal agents within RL frameworks [17, 18]. These innovative approaches utilize the descriptive and inferential strengths of LLMs to more effectively navigate and solve complex environmental challenges.

By synergizing LLMs' linguistic capabilities, they push the boundaries of what intelligent systems can achieve in tackling complex tasks [19].

The generation of related rules represents the initial step for an ILP-based RL agent to establish rule-based policies. Generating a comprehensive rule space from scratch in symbolic RL presents significant challenges due to the vastness of the potential rule space [10]. Previous works have addressed this problem by using algorithms based on human expert rule templates [15]. However, our approach leverages the common sense knowledge embedded within LLMs to efficiently generate the necessary rules. We replace human-generated rule templates with LLM-generated rule templates, empirically demonstrating that our approach is as efficient as other rule generation algorithms.[1]

In section 2, we introduce our algorithm for identifying necessary subtasks. Section 3 presents the LLM rule generation technique. In this section, we explore how we can utilize the interpretable subtasks generated from the previous section to develop further interpretable rules. These rules are then used as rule templates for an ILP-based RL agent, formulating a rule-based policy.

2. LANDMARK IDENTIFICATION FROM TRAJECTORIES

Reinforcement learning (RL) tackles decision-making problems in environments defined by a state space S, an action space A, and transition dynamics $P(s' \mid s, a)$, where the goal is to maximize rewards over time. In this context, a policy $\pi(a \mid s)$ maps states to actions, guiding the agent towards maximizing the expected discounted sum of rewards $E_\pi \left[\sum_t \gamma^t r(s_t, a_t, s_{t+1})\right]$, where γ is a discount factor to prioritize immediate rewards. This sets the stage for designing RL algorithms that can learn optimal actions in complex decision spaces.

Incorporating a rule-based policy within this RL framework can provide a structured and interpretable way to guide decision-making. Leveraging concepts from First-Order Logic (FOL), we represent policies as rules. In FOL, predicates describe relationships between terms (constants, variables, or function-based expressions), $p(t_1, \ldots, t_n)$, and rules consist of a head (the action to be taken) and a body (a set of predicates describing the current state). Rules are often written in the form $A : -B_1, \ldots, B_n$, where A is the head (action) and $B_1, \ldots, B_n$ are the body predicates.

In our approach, we employ an ILP-based RL agent, as described in the NUDGE [15], with states represented by grounded FOL predicates. To identify landmarks, we first apply a contrastive learning algorithm to detect potential landmark states, followed by a graph search algorithm [20] to identify the necessary grounded predicates for each subtask. We leverage both positive and negative trajectories from a Neural Network (NN) RL agent, collecting 50 positive and 500 negative trajectories during the early stage of training. The advantage of using an NN agent is that it does not require prior information about the environment. Positive trajectories are those that successfully achieve the task's goal, while negative ones do not.

Each state trajectory is defined as τ_i, where $\tau_i = (s_0, s_1, \ldots, s_T)$. τ_i^p is the i'th positive trajectory and τ_i^n refers to i'th negative trajectory. We used a two-layer NN to assign a number between zero and one to every state. We propose that landmarks should consistently appear in all positive trajectories but may occasionally appear in some negative ones. To achieve this, we train the NN to output 1 for landmark states and 0 for non-landmark states. For this aim, we should maximize this function:

$$\sum_{(\tau_i^p, \tau_j^n)} \log\left(\frac{\exp\left(\sum_{s_k} f_\theta(\tau_i^p(s_k))\right)}{\exp\left(\sum_{s_k} f_\theta(\tau_i^p(s_k))\right) + \exp\left(\sum_{s_k} f_\theta(\tau_j^n(s_k))\right)}\right)$$

where $\tau_i^p(s_k)$ denotes the k'th state of i'th trajectory of positive samples. The sum is over the pairs of randomly chosen trajectories from positive and negative samples. The results of the algorithm are detailed in the experimental section of this paper.

Next, we develop a method for identifying subtasks from our landmark candidates. The necessity of subtasks in every positive trajectory is a characteristic that stems from the definition of a subtask. A subtask is defined as a necessary state or subset of a state that must be visited to complete a task.

The algorithm takes as its input the set of all candidates' landmark states resulting from the contrastive learning algorithm. Then it proceeds to evaluate all combinations of grounded predicates to identify all subtasks. As shown in Fig. 1, we associate all of the predicates to $Node_{00}$ at the root of the tree graph. A subtask is defined by its consistent presence in every positive trajectory and its absence in negative trajectories, which we verify by examining random negative samples. If no subtask is detected at the current node, we extend the tree graph by adding leaves. Each leaf is created by removing a predicate from the current set assigned to the node, move to a deeper level, and add the newly formed nodes to the frontier.

To determine the next node to explore from the frontier, a softmax function is applied on $f(Node)$, which is based on two factors: the number of unique predicate combinations in the node and its level in the search hierarchy. Our goal is to find the largest set of predicates that define a subtask. Once a node is validated, it is explored further by increasing its level and removing it from the frontier. Details are provided in Algorithm 1.

Our graph search algorithm identifies the largest set of predicates that reliably activate landmarks, treated as subtasks for the next stage. Fig. 2 highlights how the graph search enhances the algorithm's precision and efficiency.

[1] https://github.com/KheirAli/LLM_Landmark.git

Algorithm 1 Graph Search Algorithm

```
Landmarks ← ∅, $g(C) \leftarrow 0$
$Node_{0,0}$ ← All predicates used in the embedding input
Frontier Nodes (FN) ← $Node_{0,0}$
Frontier Nodes States(FNS) ← All unique detected states with a value of 1 in the contrastive learning algorithm
Negative Test ($NT$) ← Random 10 negative sample
while $g(c) < 1$ do
    $f(Node_{j,i}) = -\frac{Number\ of\ States\ in\ Node_{j,i}}{Number\ of\ States\ in\ Node_{0,0}} - i$
    Chosen Node ($CN_{j,i}$) ← Choose a node from softmax distribution over all $f(Node_{j,i})$ on FN
    Node States(NS) ← Unique states with CN predicates
    for state in NS do
        if (state $\in \tau_{i,p}, \forall i$) & ($\exists i \in NT$, state $\notin \tau_{i,n}$) then
            Landmarks ← state, $g(C) \leftarrow 1$
        end if
    end for
    Frontier Nodes (FN) ← $FN/CN_{j,i}$
    New Nodes($NN_{j:k+j,i+1}$) ← $CN_{j,i}/p_k$
    Frontier Nodes (FN) ← FN+ $NN_{j:k+j,i+1}$
end while
```

3. RULE GENERATION FOR ATTAINING LANDMARKS USING LLM

By employing subtask decomposition, we simplified the challenge of learning RL policy rules for a complex task by breaking it down into smaller, manageable subtasks. In this context, we employed few shot learning with the LLAMA 3.1 [21] model to generate rules for each identified subtask. The experimental results are discussed in the following section, with details of the prompts shown in Fig. 4. The input to the LLM consists of a constant part, which includes definitions of predicates used to represent the states and general information about the environment. To create base rules, we combined the subtask with a base prompt and two rule examples from other environments, helping the model follow the rule template and grasp the logic behind the rules.

To evaluate the effectiveness of a rule, we tested the RL agent using generated template rules. If the rules fail to achieve the subtask, we refine the template rules. We record the state corresponding to the lowest reward as the failed state. Since the LLM did not generate a complete set of rules for us, we refined them by utilizing additional prompts. These prompts ask the LLM to interpret the rule and modify it by removing some predicates to increase generality or by adding predicates to enhance detail. Depending on the complexity of the subtask, we can generate rules that are either more general or more detailed.

4. EXPERIMENT

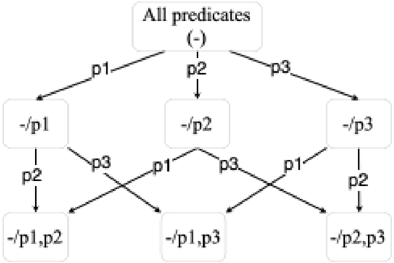


Fig. 1. The schematic shows the graph generated by the algorithm for environments with three predicates. The root represents states containing all three predicates, and each subsequent level illustrates states formed by removing one predicate. Each edge indicates which predicate was removed at that node. The final leaves contain only one predicate.

Score	4 subtasks	3 subtasks	2 subtasks
GetOut*	22.86 ± 2.46	23.06 ± 2.37	23.29 ± 2.34
GetOut	22.84 ± 2.49	23.02 ± 2.33	23.31 ± 2.38
Loot*			5.31 ± 0.65
Loot			5.45 ± 0.51

Table 1. Comparison of our algorithm on tasks with and without predicate knowledge, where GetOut* and Loot* exclude predicates like have-object and pickup-object, while GetOut and loot include them. Score is the agregated rewrds.

The environment, adapted from the GetOut and Loot environment in [15]. GetOut has been modified to include distinct landmarks and new objects, such as two coins, a flag, and a red key. The four subtasks we refer to are: collecting two coins, collecting a flag, collecting a blue key, and then proceeding to the door.. An example state of the modified GetOut environment is shown in Fig. 6.

In Table 1, we compare the results of the algorithm in two environments: one with additional predicates and knowledge, and another with fewer predicates. We evaluate it on tasks with varying numbers of subtasks. Since we did not have labels for the landmark states in Fig. 2, we manually labeled them to evaluate the accuracy of subtask detection. Table 2 highlights the necessity of subtasks, showing results after rule generation and policy learning. Fig. 3 compares our algorithm to human generated rules, demonstrating similar success and showing that missing subtask results in task failure. Fig. 5 illustrates the comparison between the rule policy from the Nudge and a template generated rule and policy for the coin subtask.

5. CONCLUSION

The paper introduces a novel method for detecting landmarks to decompose complex tasks into subtasks. FOL state representation and leveraging LLM led us to create rule-based policies through an ILP-based RL agent. Experiments demonstrate that the algorithm is both accurate and efficient in subtask detection and that LLM-guided rule generation This method reduces reliance on predefined logic predicates, offering a more flexible and scalable solution. Future work aims to extend the approach to real-world tasks and enhance rule fine-tuning for broader generalization.

Score	4 subtasks	3 subtasks	2 subtasks
GetOut*/4	22.86 ± 2.46	-10.24 ± 2.05	-14.47 ± 2.54
GetOut*/3		23.02 ± 2.33	-10.41 ± 2.64

Table 2. Comparison of subtask necessity, with the x-axis showing the number of learned subtasks in our algorithm and the score representing the average return for tasks with 3 and 4 subtasks.

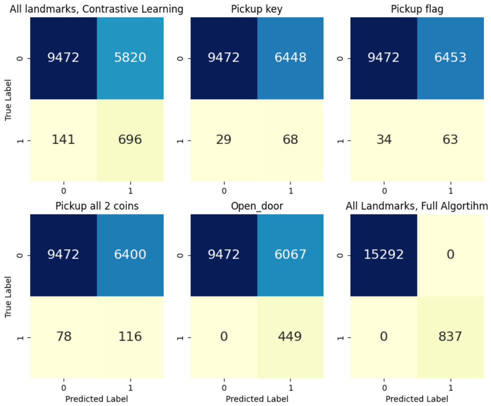


Fig. 2. Performance of landmark identification: The top-left plot shows contrastive learning results for all landmarks, and the bottom-right plot displays improvements after applying a tree graph search. Other plots focus on specific landmarks before the graph search. Recall improved from 83% to 100%, and precision increased from 10% to 100% with the search algorithm.

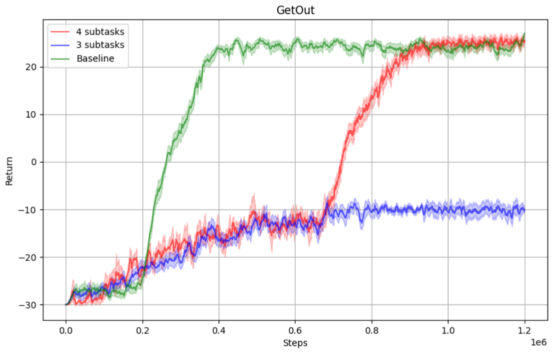
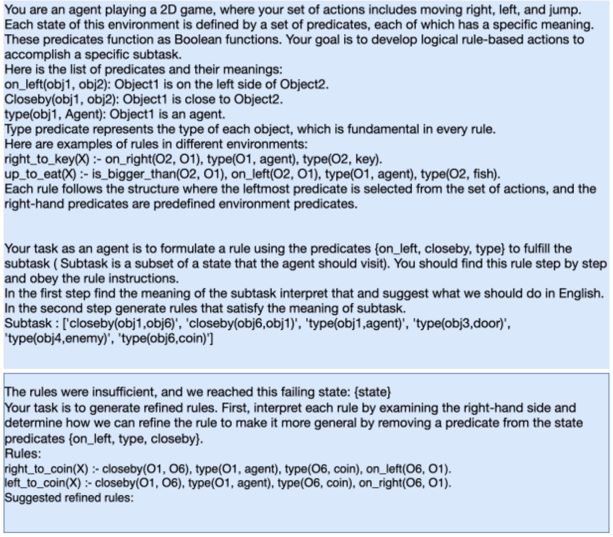


Fig. 3. Comparison of algorithm convergence: The red plot shows performance on 4 subtasks, the blue plot on 3 subtasks, and the green plot represents the ILP-RL agent using a human expert's rule template.

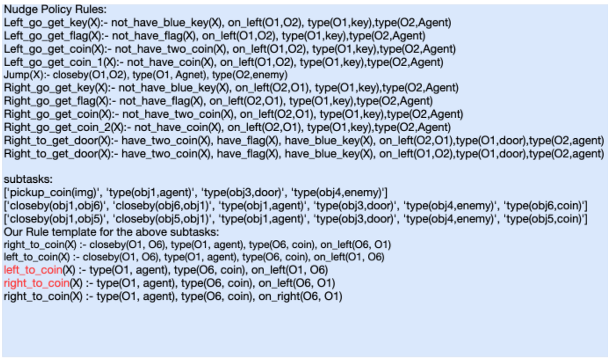


You are an agent playing a 2D game, where your set of actions includes moving right, left, and jump.
Each state of this environment is defined by a set of predicates, each of which has a specific meaning.
These predicates function as Boolean functions. Your goal is to develop logical rule-based actions to accomplish a specific subtask.
Here is the list of predicates and their meanings:
on_left(obj1, obj2): Object1 is on the left side of Object2.
Closeby(obj1, obj2): Object1 is close to Object2.
type(obj1, Agent): Object1 is an agent.
Type predicate represents the type of each object, which is fundamental in every rule.
Here are examples of rules in different environments:
right_to_key(X) :- on_right(O2, O1), type(O1, agent), type(O2, key).
up_to_eat(X) :- is_bigger_than(O2, O1), on_left(O2, O1), type(O1, agent), type(O2, fish).
Each rule follows the structure where the leftmost predicate is selected from the set of actions, and the right-hand predicates are predefined environment predicates.

Your task as an agent is to formulate a rule using the predicates {on_left, closeby, type} to fulfill the subtask (Subtask is a subset of a state that the agent should visit). You should find this rule step by step and obey the rule instructions.
In the first step find the meaning of the subtask interpret that and suggest what we should do in English.
In the second step generate rules that satisfy the meaning of subtask.
Subtask : ['closeby(obj1,obj6)', 'closeby(obj6,obj1)', 'type(obj1,agent)', 'type(obj3,door)', 'type(obj4,enemy)', 'type(obj6,coin)']

The rules were insufficient, and we reached this failing state: {state}
Your task is to generate refined rules. First, interpret each rule by examining the right-hand side and determine how we can refine the rule to make it more general by removing a predicate from the state predicates {on_left, type, closeby}.
Rules:
right_to_coin(X) :- closeby(O1, O6), type(O1, agent), type(O6, coin), on_left(O6, O1).
left_to_coin(X) :- closeby(O1, O6), type(O1, agent), type(O6, coin), on_right(O6, O1).
Suggested refined rules:

Fig. 4. Top image: Prompt for generating the base template rule, including a constant section with few shot examples from various environments and the specific coin subtask. Bottom image: Few shot learning applied to refine the template rule by generating more general rules.



Nudge Policy Rules:
Left_go_get_key(X):- not_have_blue_key(X), on_left(O1,O2), type(O1,key),type(O2,Agent)
Left_go_get_flag(X):- not_have_flag(X), on_left(O1,O2), type(O1,key),type(O2,Agent)
Left_go_get_coin(X):- not_have_two_coin(X), on_left(O1,O2), type(O1,key),type(O2,Agent)
Left_go_get_coin_1(X):- not_have_coin(X), on_left(O1,O2), type(O1,key),type(O2,Agent)
Jump(X):- closeby(O1,O2), type(O1, Agnet), type(O2,enemy)
Right_go_get_key(X):- not_have_blue_key(X), on_left(O2,O1), type(O1,key),type(O2,Agent)
Right_go_get_flag(X):- not_have_flag(X), on_left(O2,O1), type(O1,key),type(O2,Agent)
Right_go_get_coin(X):- not_have_two_coin(X), on_left(O2,O1), type(O1,key),type(O2,Agent)
Right_go_get_coin_2(X):- not_have_coin(X), on_left(O2,O1), type(O1,key),type(O2,Agent)
Right_to_get_door(X):- have_two_coin(X), have_flag(X), have_blue_key(X), on_left(O2,O1),type(O1,door),type(O2,agent)
Right_to_get_door(X):- have_two_coin(X), have_flag(X), have_blue_key(X), on_left(O1,O2),type(O1,door),type(O2,agent)

subtasks:
['pickup_coin(img)', 'type(obj1,agent)', 'type(obj3,door)', 'type(obj4,enemy)']
['closeby(obj1,obj6)', 'closeby(obj6,obj1)', 'type(obj1,agent)', 'type(obj3,door)', 'type(obj4,enemy)', 'type(obj6,coin)']
['closeby(obj1,obj5)', 'closeby(obj5,obj1)', 'type(obj1,agent)', 'type(obj3,door)', 'type(obj4,enemy)', 'type(obj5,coin)']
Our Rule template for the above subtasks:
right_to_coin(X) :- closeby(O1, O6), type(O1, agent), type(O6, coin), on_left(O6, O1)
left_to_coin(X) :- closeby(O1, O6), type(O1, agent), type(O6, coin), on_left(O1, O6)
left_to_coin(X) :- type(O1, agent), type(O6, coin), on_left(O1, O6)
right_to_coin(X) :- type(O1, agent), type(O6, coin), on_left(O6, O1)
right_to_coin(X) :- type(O1, agent), type(O6, coin), on_right(O6, O1)

Fig. 5. Comparison of the human expert's rule policy with LLM-generated rules for coin subtask. The final policy chosen by the ILP-RL agent is marked in red, demonstrating the effectiveness of subtasks in guiding smaller policy rules with less predicate or environmental information.

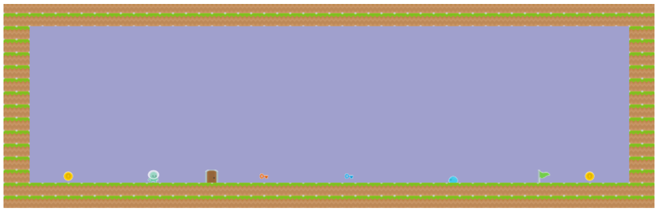


Fig. 6. GetOut environment: The humanoid agent and other objects are in a defined state. The agent can move right, left, or jump.

6. REFERENCES

[1] Andrew G Barto and Sridhar Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, pp. 341–379, 2003.

[2] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021.

[3] Julie Porteous, Laura Sebastia, and Jörg Hoffmann, "On the extraction, ordering, and usage of landmarks in planning," in *Sixth European Conference on Planning*, 2014.

[4] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo, "Improving hierarchical planning performance by the use of landmarks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012, vol. 26, pp. 1763–1769.

[5] Kishor Jothimurugan, Steve Hsu, Osbert Bastani, and Rajeev Alur, "Robust subtask learning for compositional generalization," in *International Conference on Machine Learning*. PMLR, 2023, pp. 15371–15387.

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

[7] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton, "Contrastive representation learning: A framework and review," *Ieee Access*, vol. 8, pp. 193907–193934, 2020.

[8] 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*. PMLR, 2021, pp. 5979–5989.

[9] Kinjal Basu, Keerthiram Murugesan, Subhajit Chaudhury, Murray Campbell, Kartik Talamadupula, and Tim Klinger, "Explorer: Exploration-guided reasoning for textual reinforcement learning," *arXiv preprint arXiv:2403.10692*, 2024.

[10] Zhengyao Jiang and Shan Luo, "Neural logic reinforcement learning," in *International conference on machine learning*. PMLR, 2019, pp. 3110–3119.

[11] Ali Payani and Faramarz Fekri, "Incorporating relational background knowledge into reinforcement learning via differentiable inductive logic programming," *arXiv preprint arXiv:2003.10386*, 2020.

[12] Duo Xu and Faramarz Fekri, "Integrating symbolic planning and reinforcement learning for following temporal logic specifications," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 01–08.

[13] Richard Evans and Edward Grefenstette, "Learning explanatory rules from noisy data," *Journal of Artificial Intelligence Research*, vol. 61, pp. 1–64, 2018.

[14] Ali Payani and Faramarz Fekri, "Inductive logic programming via differentiable deep neural logic networks," *arXiv preprint arXiv:1906.03523*, 2019.

[15] Quentin Delfosse, Hikaru Shindo, Devendra Dhami, and Kristian Kersting, "Interpretable and explainable logical policies via neurally guided symbolic abstraction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei, "Llm as a mastermind: A survey of strategic reasoning with large language models," *arXiv preprint arXiv:2404.01230*, 2024.

[17] Alex Place, "Adaptive reinforcement learning with llm-augmented reward functions," *Authorea Preprints*, 2023.

[18] Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An, "True knowledge comes from practice: Aligning llms with embodied environments via reinforcement learning," *arXiv preprint arXiv:2401.14151*, 2024.

[19] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al., "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[20] Stephen Muggleton, "Inductive logic programming," *New generation computing*, vol. 8, pp. 295–318, 1991.

[21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al., "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.