2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Unsupervised Learning of Neuro-symbolic Rules for Generalizable Context-aware Planning in Object Arrangement Tasks

Siddhant Sharma[1], Shreshth Tuli[1], and Rohan Paul[2]
[1]Work primarily done while at IIT Delhi. [2]Affiliated with Computer Science and Engg. (CSE) and Yardi School of AI (ScAI), IIT Delhi.

***Abstract*— As robots tackle complex object arrangement tasks, it becomes imperative for them to be able to generalize to complex worlds and scale with number of objects. This work postulates that extracting action primitives, such as push operations, their pre-conditions and effects would enable strong generalization to unseen worlds. Hence, we factorize policy learning as inference of such generic rules, which act as strong priors for predicting actions given the world state. Learnt rules act as propositional knowledge and enable robots to reach goals in a zero-shot method by applying the rules independently and incrementally. However, obtaining hand-engineered rules, such as PDDL descriptions is hard, especially for unseen worlds. This work aims to learn generic, sparse, and context-aware rules that govern action primitives in robotic worlds through human demonstrations in simple domains. We demonstrate that our approach, namely** RLAP**, is able to extract rules without explicit supervision of rule labels and generate goal-reaching plans in complex Sokoban styled domains that scale with number of objects.** RLAP **furnishes significantly higher goal reaching rate and shorter planning times compared to the state-of-the-art techniques. The code, dataset, and videos are hosted at https://rule-learning-rlap.github.io/.**

I. Introduction

Advances in autonomy are enabling robots to enter real world domains such as homes, factories, hospitals, etc. As robots assist humans in increasingly complex tasks, they need to achieve intended goals in previously unmodeled domains. To generalize to such unseen domains, akin to humans, robots can leverage the rules governing primitive interactions, such as pushing an apple changes its position [1]. Humans inherently recognize such propositional or *if-else* style rules, such as those involving "pushing apples", that are applicable to most light objects as it depends on the weight characteristics and not the shape/size/texture of the object. Such rules allow humans to build strong priors on the physical working principles of such primitive interactions, enabling them to few-shot generalize to unseen domains or settings. Motivated from humans, we develop models to *learn* high-level action rules for object manipulators in robotic domains. We rely on low-level motion planners to synthesise each individual action. Unlike models that directly aim to predict actions given a world state [2], [3], we aim to decouple the problem into learning "general-purpose" interaction rules and subsequently using these as strong priors to infer actions. Planning to reach goals by independently identifying rules and incrementally applying them would allow robots to generalize to unseen settings.

However, learning such rules is hard. In typical use-cases, we need to hand-encode action behaviours and rules as pre and post-conditions such as in PDDL domain descriptions. However, PDDLs do not handle noise in action transitions and require the knowledge of non-colloquial syntax. Recent methods extract PDDLs from natural-language (NL) descriptions [4], but still face the limitation of requiring explicit NL rule annotations for new domains and not being domain agnostic [5]. For typical human partners, it is infeasible to explicitly provide generic rules. Thus, we consider a paradigm where humans can demonstrate actions and the transition observations can be used to *learn* rules. However, this too is challenging. First, learning the applicability of the rules requires additional context, but is important as it allows for offline plan search. Akin to a digital-twin, we can emulate the preconditions and post-conditions. Second, it is important to ensure that we learn rules in complex domains, when disentangling rules is hard. This could be when, for instance, based on the context, the same action can have different effects or vice versa. Example, "move right" and "move left" actions can keep the object stationary if there is a wall on the right and the left, respectively.

This paper builds upon Neural Production Systems [1] to learn a set of sparse, modular and context-aware interaction rules from demonstrations, along with a feasibility checker for their application in any state. The rules and checker provide strong priors to generalize action inference and reach goal states in unseen domains. We present a Rule Learner and Action Predictor (RLAP) that not only learns the applicability conditions and effects of primitive actions from human-demonstrations, but also leverages these rules to infer goal-reaching plans. Even with demonstrations of simple interactions with limited complexity, RLAP is able to extract rules and apply them to reach goals in much harder domains. Our core contribution is in learning sparse rule abstractions to facilitate planning in unseen domains. We demonstrate in Sokoban inspired object rearrangement tasks that RLAP is able to accurately identify rules and have up to 65% and 20% higher goal reaching rate while having up to 95% lower planning times compared to the state-of-the-art action prediction and rule-based models.

II. Related Works

Traditional approaches to robot planning assume the presence of a symbolic model of actions and the world state for synthesizing goal-directed plans [6]–[8]. Encoding such symbolic representations requires expert supervision and are

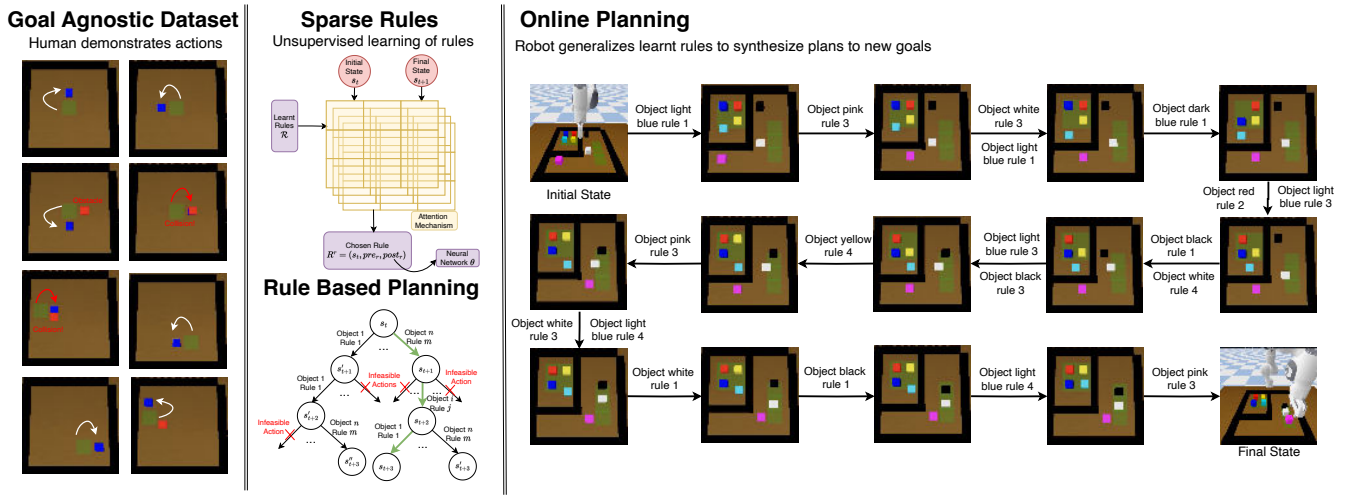


Fig. 1: **Overview.** The human provides multiple demonstrations of "high-level" actions possible in the domain for the purposes of synthesising an arrangement plan. Our approach distills the demonstrations into action "rules" in an unsupervised manner. Rules are neuro-symbolic are nature and capture local object interaction (via slot attention) and predict post effects and collisions. The representation of actions are rule are akin to PDDL-style representations that generalise strongly to object instances that satisfy the context. The learned rules enable generalization to longer horizon plans for unseen goals.

often brittle in practice limiting their use. As an alternate, the *learning-to-plan* approaches rely on data provided by humans or collected by the agent to predict actions for a stated goal. Much of the work in this class relies on imitation learning [9] or reinforcement learning [10]–[12] and has demonstrated significant successes in learning short-horizon motor skills such as pouring, pushing, insertion, etc. Efforts to enhance the generalization to longer horizon plans rely on introducing hierarchy/abstraction in the action space resulting in more compact policies. Here, Brunskill et al. [13] create *macro-actions* to scale planning in partially-known domains. Srivastava et al. [14] observe past planning sequences to construct state/action abstractions to aid long-range planning. The work of Andreas et al. [15] capture the compositional nature of language instructions via modular and composable abstractions that can be grounded in the environment. Similar to these approaches, this work aims to learn abstract *rules* to scale to multi-step tasks that may require rich object interactions. Along with learning the action effects, we learn collision-awareness based on local geometric context, allowing the robot to assess feasibility of action outcomes during the planning process itself.

A number of action or *skill* representations have been proposed in literature [16]. Konidaris et al. [17] model actions, the applicability and effects of actions as classifiers over the state space and reason over compositions of actions by establishing constraints over the effects of one action and the pre-condition of another. Kaebling et al. [18] consider planar pushing tasks by focusing on learning a representation for post conditions of an action. Planning in this case is carried out by back-chaining through action models. Efforts such as [19] learn pre/post conditions as *affordance maps* directly from image data demonstrating learning for interactions such as multi-object pushing, blowing, hitting etc. Similarly, the work of [20] constructs grounded preconditions of actions directly in 3D point clouds acquired from depth sensors by a robot. Wherein the aforementioned works supervise the action models definition and learn the grounding via data, our approach learns factored and composable action representations *unsupervised* from demonstration data.

Our work is closely related to efforts that learn sparse transition rules [21] using GNN with an explicit object-centric scene representations [22]. Following [1], our approach adopts the object-centric scene representation but injects a strong sparsity prior that suggests that actions operate on objects based on a local context. This paradigm enables stronger generalization compared to GNN-based transition models as we demonstrate through experiments.

III. Problem Formulation

World Model. The robot is an autonomous agent manipulating objects in a confined world, denoted as $s \in \mathcal{S}$. Here, the world state is a collection of objects $s = (o_i, p_i, \eta_i)_1^n$, where objects are encapsulated as their geometry/extent (o_i), coordinates (p_i), and attributes (η_i) such as $IsMovable(o_i) \in \{true, false\}$ or $Type(o_i) \in \{object, obstacle, wall\}$. We assume the presence of a perception system that populates the world state s_t at the time-instance t and that all objects are always fully-observable.

Robot Action Model. Consider a robot that is capable of low-level behaviours such as changing the pose of its end-effector, opening or closing the gripper and transport an object to a target configuration. We represent the action at the t-th timestep by $a_t \in \mathcal{A}$. We assume that the robot is confined to a table-top domain where it can move an object or a set of objects to a candidate goal region. The grasping, pushing or motion of the arm is confined to a 2.5D space. We also assume the presence of immovable walls in the

environment. There might multiple movable objects and the robot is expected to place an object of interest within the goal region. There may be other objects in this region that the agent may have to remove to clear the region and achieve the goal. We consider a non-deterministic transition model where $s_{t+1} \sim T(\cdot \mid s_t, a_t)$. Further, we represent the goal-objective as a reward model $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^+$, which is a positive value when we reach a goal state and a negative value when there is an object collision.

Human Demonstrations. We assume the presence of human partner(s) who provides demonstrations of primitive interactions and attributes a linguistic label to it. We collect a dataset $D = \{(s_t, s_{t+1}, \lambda_t)\}_{t=1}^{d}$, where λ_t is the NL label for the corresponding action, such as "move left" (see "Goal Agnostic Dataset" section of figure 1 for sample demonstrations). We assume that humans possess the knowledge of latent rules corresponding to each action, using which they plan, but demonstrate only actions. Language labels induce modularity and tie the action abstractions across the dataset.

Learning Rules. The objective is to reach a goal region, represented as g. The underpinning assumption of this work is that there exist finite latent learnable rules, denoted as $\tilde{\mathcal{R}} = (\tilde{R}^0, \ldots, \tilde{R}^m)$, where each rule takes the form $\tilde{R}_i^j = (s_t, pre_i, post_i)$ for object o_i. Here, pre_i and $post_i$ are pre-conditions on o_i and effects of the rule on the object. For instance for a "move left" action, pre_i would include condition on s_t such that $\nexists\ o_k$ such that o_k is at the immediate left of o_i. $post_i$ would include the effect where o_i is moved to the left. These rules are not explicitly labeled by humans, and are latent to the model. However, we want to ensure that the learnt rules $\mathcal{R} = (R^0, \ldots, R^m)$ are close to the latent rules of the domain. As we do not have direct supervision of these, we leverage the reward signal over the similarities between the predicted and ground truth final state to learn generic rules. Once the rules are learnt, we use a model-based planner to plan on novel environments.

IV. Technical Approach

Following the formulation in Section III, we factor plan generation as (i) learning goal agnostic rules that govern action primitives and (ii) leveraging these rules as priors for searching a goal-reaching plan.

Rule Representations. To learn rules and their correspondence with the observed state in the demonstration, we use the slot attention mechanism [23], [24]. Slot attention is apt for rule learning because of the ability to represent rules as generic *slot* vectors that may be combined with any given state or goal [1]. Slot attention is also a form of query-key attention, where the key is rule and the query is the slot as the state. We use slot vectors as $V_i^j = (p_i^0, p_i^1, \lambda_j)$ where p_i^0, p_i^1 are the initial and final coordinates of o_i and λ^j is the NL encoding of action corresponding to rule R^j. The values of the slots are static for a given state s_t and we have $n \cdot m$ key-query pairs for the n objects and m rules for predicting V_p and R_r, and n pairs for predicting V_s as the query is fixed using V_p. Further, we represent rules as $R^j = (\vec{R^j}, PC^j)$, where $\vec{R^j}$ is a learnable embedding vector for the rule and PC^j is the post-condition predictor as a neural model. The m rules are learnable and are agnostic to the given state.

Neural Rule Learner. We use neural networks to learn rules, summarized in figure 2. We use multi-layered perceptron (MLP) attention functions $W^{q_1}(\cdot)$ and $W^{k_1}(\cdot)$ as query and key generators, and query-key attention [23] with Gumbel Softmax [25] for selecting the primary slot, *i.e.*, the object on which the rule is applied and the rule R^r itself. Once the primary slot and the rule has been selected, we use another set of MLP models, $W^{q_2}(\cdot)$ and $W^{k_2}(\cdot)$ to generate queries and keys for the secondary slot. We use the previously selected primary slot V_p for the query vector. Using query-key attention we generate the secondary slot V_s that allows us to check context, such as walls or other objects, that is required to predict final state or feasibility of rule application. The initial coordinates of the primary p_p^0 and secondary slot object p_s^0 are inputs to the post-condition predictor $PC^r(\cdot)$ associated with the chosen rule to predict the final state. We used an MLP for $PC^j(\cdot)\ \forall j \in [1, m]$ with a 32-size hidden layer, and MLPs for attention functions $W^{q_1}(\cdot), W^{k_1}(\cdot), W^{q_2}(\cdot), W^{k_2}(\cdot)$ with a 16-size hidden layer, and the 16-size output layer, which is the key or query vector.

Random Forest Feasibility Checker. The above mentioned primary and secondary slot predictors and post-condition predictors PC^j are required to choose rules and generate action effects. However, for rule applicability, we use feasibility checkers (denoted as $FC^j(\cdot)$), which are binary classifiers corresponding to each $R^j\ \forall j \in [1, m]$, that check whether the predicted rule is applicable on the primary slot object. We implement these feasibility checkers as random forest classifiers with 100 trees each. The chosen rule R^r and the initial coordinates of the primary and secondary slot objects p_p^0 and p_s^0 are used by the corresponding feasibility checker $FC^r(\cdot)$ to predict if the rule is applicable.

Loss Function. We use a singular state-loss function $\mathcal{L}$ to learn all parameters of the neural rule learner, *viz*,

$$\begin{aligned} \mathcal{L} &= \alpha \cdot \mathcal{L}_{MSE} + \beta \cdot \mathcal{L}_{KL} \\ \mathcal{L}_{KL} &= \sum_{i=1}^{n} \sum_{j=i+1}^{m} D_{KL}(PC^i(\cdot), PC^j(\cdot)) \end{aligned} \tag{1}$$

where $\mathcal{L}_{MSE} = MSE(\hat{p}_i^1, p_i^1)$ is the mean-squared error between the predicted $(\hat{p}_i^1)$ and the ground-truth final coordinates (p_i^1) of o_i in s_{t+1} state of the demonstration. We also use the Kullback-Leibler divergence [26] $D_{KL}(\hat{y}||y) = \sum_{c=1}^{M} \hat{y}_c \log \frac{\hat{y}_c}{y_c}$ among the MLP final state predictors to ensure *disentangling* of the rules and encourages the model to distribute the rule vector encodings and ensure disparate rules are learnt. We also use convex combination weights α, β to appropriately weight the two loss functions. To build the random forest, we use the Gini impurity function [27].

Curriculum and Training Details. The above mentioned learning problem is hard to scale with the size of the object set as the neural model must be able to identify generic rules, primary and contextual objects in large-scale settings. To tackle this, we devised a graded curriculum for model training. First, we learned rule post-condition predictors

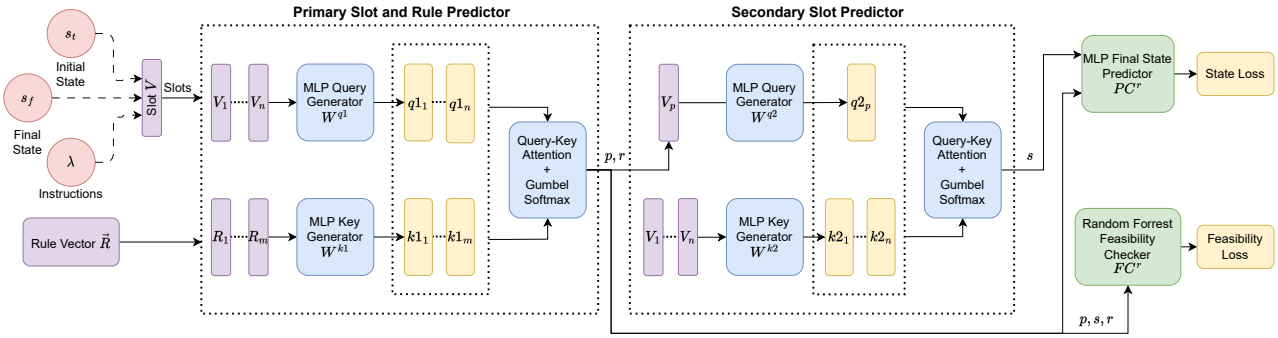


Fig. 2: **Rule Learner and Feasibility Checker.** The inputs include rule vectors and object slots. MLP predictors and feasibility checkers (in green) are learnt using the output of predicted final state and feasibility prediction and used during planning. The attention mechanism is able to predict rules without explicit supervision on the rule taken.

$PC^j(\cdot)$ using demonstrations with a single object as these are agnostic to the number of objects. Second, we used the pre-trained $PC^j(\cdot)$ predictors in a 2-object domain to relearn the rule learning attention model. Finally, we learnt feasibility checkers $FC^r(\cdot)$ using the rule predictor.

Rule-Based MCTS Planning. With the $PC^j(\cdot)$ predictors as post-conditions and feasibility checkers $FC^j(\cdot)$ as pre-conditions, we can plan to reach goals using any model-based planning technique. One such technique is Monte Carlo Tree Search (MCTS) [28] where we denote the current state as the root node of a tree and expand it using actions as edges and subsequent states as child nodes. We can roll-out the tree with action sequences and the tree-path that leads to a goal state (leaf node) becomes a goal-reaching plan. We extend MCTS with branching restriction and a check-and-repair protocol, and call it Rule-Based MCTS (RB-MCTS). First, we use the feasibility checkers $FC^j(\cdot)$ to check if a rule is applicable to prune away tree branches which correspond to infeasible actions (line 6 in Alg. 1). Second, we explicitly check if a generated plan is feasible, by iteratively checking for collisions offline. If an infeasible state is detected, say s_i, we prune off that subtree and continue our search for a new plan (line 28 in Alg. 1). This saves us from having to make explicit checks in a simulator for infeasible states while expanding the tree, only needing us to run the simulator for generated plan candidates. Due to the huge branching factor $(n \cdot m)$ of the MCTS tree for planning in these domains, we save exponential time by avoiding explicit checks.

We use a stochastic policy to explore to avoid our plan search getting stuck in a local minima. We also use a distance heuristic to speed up the search process with upper-confidence-bound (UCB) based search [29]. The branching restriction helps lower the planning time. Despite recomputation by the check-and-repair protocol, we empirically demonstrate that the planning time of RB-MCTS is lower than vanilla MCTS, while also reporting near-optimal plans (see Section V). We present two insights on how RB-MCTS advances a rule-oblivious search strategy. (i) Assuming the recall of $FC^j(\cdot)$ is high, the number of false negatives is low. Thus, the frequency of pruning a feasible state node

Algorithm 1 Rule Based Monte Carlo Tree Search

1: **procedure** RB-MCTS($s_0, G, PC(\cdot), FC(\cdot)$)
2: **while** $i \leq$ iteration limit **do**
3: $node \leftarrow s_0$
4: **while** $node$ is not terminal **do**
5: $node \leftarrow$ select $node.children$ using UCB
6: $actions \leftarrow$ all feasible actions **from** $\boldsymbol{FC(\cdot)}$
7: $node.children \leftarrow a$ on $node$ $\forall a \in actions$
8: $\pi'' \leftarrow$ random policy from $node$ **using** $\boldsymbol{PC(\cdot)}$
9: $node \leftarrow$ leaf node of $rollout$
10: $r \leftarrow \sum_{i=0}^{|\pi''|} r_i$ where $r_i =$ reward of step i
11: **while** $node$ is not root **do**
12: $visits$ of $node \leftarrow visits$ of $node + 1$
13: $reward$ of $node \leftarrow reward$ of $node + r$
14: $node \leftarrow parent$ of $node$
15: $r \leftarrow 0.95 \cdot r$
16: $finalNodes, node \leftarrow [], s_0$
17: **while** $node \notin G$ **do**
18: $node \leftarrow$ select $node.children$ using UCB
19: $finalNodes \leftarrow finalNodes + [node]$
20: **return** $finalNodes, node$
21: **procedure** RLAP($s_0, G, PC(\cdot), FC(\cdot)$)
22: $s_0, G \leftarrow$ initial states, goal states
23: $PC(\cdot), FC(\cdot) \leftarrow$ learnt rules, feasibility checkers
24: $\pi, node \leftarrow [], s_0$
25: **while** $node \notin G$ **do**
26: $\pi', node \leftarrow$ RB-MCTS$(node, G, PC(\cdot), FC(\cdot))$
27: **if** π' is not feasible **then**
28: $node \leftarrow$ last feasible node in π'
29: $\pi \leftarrow \pi + [\pi'$ till $node]$
30: **return** π

(and the corresponding sub-tree) is low. Due to uncertainty in the transition function, in the limit of infinite actions, the probability of not finding a goal-reaching plan tends to 0 in the asymptotic limit as trajectory length scales. (ii) Assuming high precision of $FC^j(\cdot)$, the false positive rate is low. Thus, the number of times we assume an infeasible state to be feasible is low leading to infrequent calls to the check-and-

repair protocol. Having infrequent calls to simulator will significantly reduce the planning time compared to MCTS.

V. Evaluation and Results

Experimental Setup. Our setup is a square-unit table top domain (with origin at the center) with a Franca Emika robot arm that is used to move objects in simulated PyBullet domain [30]. We consider four actions of moving objects left, right, up and down, corresponding to the Cartesian tabular plane. Action transitions in this domain are non-deterministic, as we induce 5% Gaussian noise to the final position. Changing the noise level causes no significant difference to trends we observe and for other noise ranges, we observe the similar improvements.

Dataset. As described previously, we collect a dataset of human demonstration where each datapoint is of the form $(s_t, s_{t+1}, \lambda_t)$. The initial configuration of objects on the table top, *i.e.*, s_t was sampled uniformly at random, discarding states with object collisions. The configuration of objects in the final state, *i.e.*, s_{t+1} was generated by manipulating an object o, chosen uniformly at random from the set of objects on the table, using the randomly selected action a_t. The language encoding λ_t was generated as a one-hot encoding of action a_t. We tried using GloVe embedding [31] of actions, but that did not perform well in our case; however, it may be relevant in other domains with semantically meaningful semantic actions labels. All demonstrations that led to the object going out of bounds were discarded for this case and there were no collisions as we only had one object on the table. For demonstrations with two objects, actions that led to infeasible states weren't dropped, and instead reported separately to learn context aware rules and feasibility checker.

Baselines. For rule learning, we consider a GNN based baseline [21] that aims to learn rules as a graph. For planning, we consider two state-of-the-art reinforcement learning models: PPO [3] and SAC [2]. Proximal Policy Optimization (PPO) is a model-free gradient-based policy optimization model that directly predicts action for a given input state. Soft-Actor Critic (SAC) is a similar approach, but uses entropy maximization to ensure exploration within the domain. We train the RL based baselines using a dataset of rollouts from the human demonstrations. Note that, by design, we reveal action labels for the RL datasets, which makes the problem easier for these competing approaches. We also consider a rule learning model, *i.e.*, NPS [1] with an MCTS based planner as a baseline.

Learning Sparse Context-Aware Rules. Table I shows the confusion matrix of rules for the test set for the single and dual object domains. We secure a 100% accuracy in the 1-object domain while trying to learn which rule was applied given an unsupervised demonstration. We use the rule post-conditions from the single-object case and relearn rule learners and feasibility checkers in the two-object domain. Even with added complexity of a new object, we see from Table I that our accuracy is 97.64% due to the graded curriculum, much higher than GNN (89.32%). Without the curriculum, our accuracy was 72.75% as the model is unable to disentangle the rules. We observe that the final state predictors have a mean-absolute error of 2.7% (Table II).

TABLE I: **Training Rule Prediction Confusion Matrix.** Learning the rules with one and two objects leads to accuracies of 100% and 97.6%, respectively.

	Rule 0	Rule 1	Rule 2	Rule 3
RLAP with single object				
Predicted Rule 0	264	0	0	0
Predicted Rule 1	0	261	0	0
Predicted Rule 2	0	0	228	0
Predicted Rule 3	0	0	0	247
RLAP with two objects				
Predicted Rule 1	11754	7	86	81
Predicted Rule 2	27	12113	75	230
Predicted Rule 3	321	78	12084	10
Predicted Rule 4	185	56	1	11934

TABLE II: **Final State Prediction Error.** Comparison of RLAP's final position prediction performance in both single object and two object domains with respect to GNN.

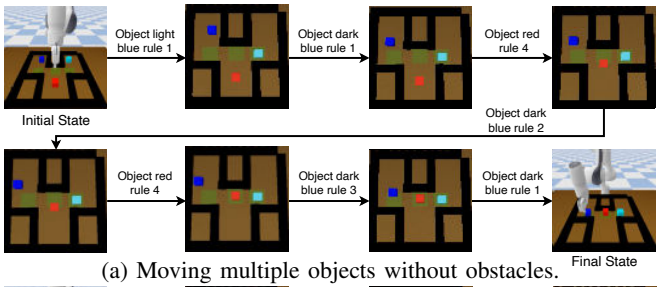
Model	Domain	Mean Absolute Error
RLAP	1 Object	0.027 ± 0.0156
RLAP	2 Object	0.015 ± 0.0256
GNN	2 Object	1.200 ± 2.5600

TABLE III: **Baseline Comparison and Ablation Analysis.** We compare goal reaching rate (GRR) and planning time of RLAP with baselines.

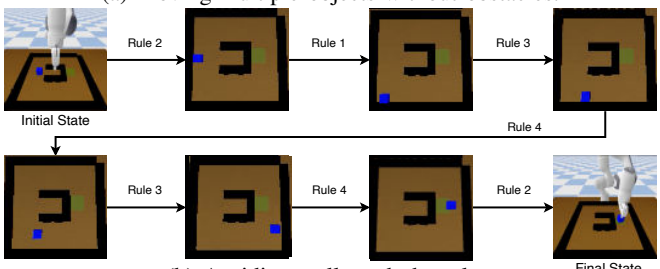
# objs	1	2	3	5	10	15	20
Planning Time (s)							
SAC	5284	5379	5047	6458	5221	4939	5266
PPO	352	333	347	332	337	**353**	**346**
NPS	10123.1	8954.01	8756.39	8299.95	8178.12	8215.32	8133.08
RLAP	**178.24**	**200.7**	**238.59**	**126.89**	**126.07**	401.53	394.34
GRR (%)							
SAC	35.6	15.09	6.89	1.78	0.11	0.02	0.13
PPO	2.71	4.19	5.09	1.97	1.39	1.12	0.72
NPS	**100.0**	90.90	80.00	80.00	60.00	33.33	30.00
RLAP	**100.0**	**93.33**	**93.33**	**86.66**	**73.33**	**53.33**	**40.00**

Rule-based Planning for Novel Goals. Using the advancements described in Section IV, we use RLAP to plan in novel settings with one or more objects, and compare against baselines. The planning space for the experiments consists of a table-top and objects with movable and immovable obstacles. The goal destinations for the objects of interest were also taken from Sokoban puzzle games where the plan to reach the goal region requires the agent to move movable obstacles to clear the path and move around immovable objects to avoid collisions. Table III shows the goal-reaching rate (GRR) and planning time for RLAP and baselines.

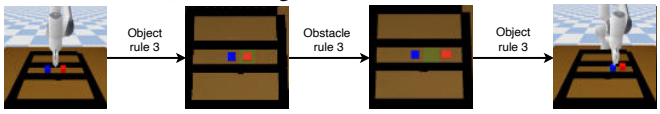
To show the generalizability of our technique, we also ran tests on hard hand-crafted domains that consisted of multiple objects and obstacles that cordoned off certain areas of the table. The goal locations were also set in a way to showcase the various use cases that highlight how RLAP is able to outperform baselines (see Figure 3). The walls are shown in black and the coloured blocks are movable objects with regions marked in green being the goal regions. RLAP is



(a) Moving multiple objects without obstacles.



(b) Avoiding walls and obstacles.



(c) Removing movable obstacles to reach goal.

Fig. 3: **Sample plans generated by RLAP.** (top) no obstacles and direct line of path to goal regions. (middle) need to move around immovable obstacles, i.e., walls. (bottom) movable obstacle on top of the goal region that needs to be removed to clear the region.

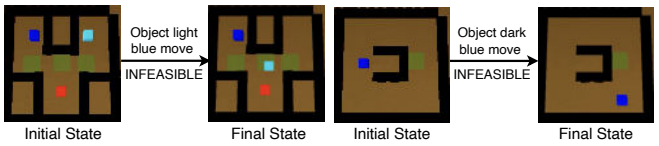


Fig. 4: **Failure cases for SAC.** The baseline model predicts to directly cross an immovable object.

able to reach goal states in all cases, when there is direct path to the goal, we need to avoid walls or move obstacles to reach goal states. Due to the absence of learnt rules, model-free approaches (PPO and SAC) are unable to scale well with number of objects as they predict actions based on the input state and fail to move the object of interest towards the goal in novel settings (such as >2 objects, see figure 4). This can also be observed by the sudden drop in GRR for settings with more than 2 objects. Further, due to no proactive feasibility checks in NPS, it often generates an infeasible plan, *i.e.*, when there is a collision, which justifies its poor GRR compared to RLAP. In terms of the planning time, NPS performs explicit collision check for each action in the MCTS rollouts, which leads to much higher planning overhead compared to RLAP.

Model Explorations. Table IV shows the fraction of plans in which recomputation was required by the check-and-repair protocol. Without this recomputation, the goal reaching rate falls significantly, demonstrating the importance of plan

TABLE IV: **Analysis of Plan Recomputations.** The percentage of times RLAP recomputes plan.

# objects	1	2	3	5	10	15	20
	Fraction of recomputations (%)						
RLAP	0	6.66	26.66	20	26.66	6.66	13.33
	GRR (%)						
RLAP (w/o recomp.)	**100.0**	86.66	66.66	66.66	46.66	46.66	26.66
RLAP	**100.0**	**93.33**	**93.33**	**86.66**	**73.33**	**53.33**	**40.00**

TABLE V: **Model Exploration (curriculum).** Rule learning without curriculum

	Rule 0	Rule 1	Rule 2	Rule 3
	RLAP (w/o curriculum) with two objects			
Predicted Rule 0	19	11	14	15
Predicted Rule 1	0	252	0	0
Predicted Rule 2	230	0	230	0
Predicted Rule 3	0	0	0	229

TABLE VI: **Model Exploration (feasibility checker).** Comparing Random Forest and MLP based feasibility checkers.

Model Type	Class	Precision	Recall	F1-score
Random Forest	Feasible	1.0	0.99	0.99
	Not Feasible	0.98	0.97	0.98
MLP	Feasible	0.8	0.99	0.88
	Not Feasible	0.07	0.20	0.1

checking in case of infeasible action being predicted. Further, when we train RLAP directly with the combined dataset of one and two object domain demonstrations, the rule prediction accuracy of the model drops from 97.64% to 72.75% (see Table V). Thus, having a graded curriculum facilitates training and enables the model to *learn* rules and contexts required to apply rules independently, to apply in downstream feasibility checking and planning tasks. Further, Table VI compares Random Forest and MLP based feasibility checkers. We keep the same parameter size of both models for fairness. We observe that for the given task, Random Forest outperforms MLP due to data sparsity.

VI. Conclusions

This paper proposes a novel technique, namely RLAP, for unsupervised rule learning using just the demonstration data to factor out modular and context-aware domain rules. Such rules act as independent action primitive priors, that can be applied independently and incrementally to reach goal states. We demonstrate that the rule and context aware planning with a Monte-Carlo style plan search engine facilitates generalization to unseen domains and scale with the complexity of the world state. Further, our check-and-repair protocol improves goal reaching rates. We qualitatively demonstrate that despite being trained on demonstrations with up to two objects, the proposed model is able to reach goals in unseen domains, more objects, and more complex settings, requiring temporal reasoning and long-horizon planning. Future work will investigate learning of complex and hierarchical rules (e.g., grasping and stacking), outputting uncertainty in rule prediction and experiments on a real robot.

VII. Acknowledgements

Rohan Paul acknowledges travel fund support from the Yardi School of Artificial Intelligence (ScAI), IIT Delhi.

References

[1] A. P. Goyal, A. Goyal, A. Didolkar, *et al.*, "Neural production systems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 25 673–25 687, 2021.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1861–1870. [Online]. Available: https://proceedings.mlr.press/v80/haarnoja18b.html.

[3] Y. Wang, H. He, and X. Tan, "Truly proximal policy optimization," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, R. P. Adams and V. Gogate, Eds., ser. Proceedings of Machine Learning Research, vol. 115, PMLR, 2020, pp. 113–122. [Online]. Available: https://proceedings.mlr.press/v115/wang20b.html.

[4] S. Miglani and N. Yorke-Smith, "Nltopddl: One-shot learning of pddl models from natural language process manuals," in *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). ICAPS*, 2020.

[5] D. Höller, G. Behnke, P. Bercher, *et al.*, "Hddl: An extension to pddl for expressing hierarchical planning problems," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 9883–9891.

[6] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[7] G. Canal, M. Cashmore, S. Krivić, G. Alenyà, D. Magazzeni, and C. Torras, "Probabilistic planning for robotics with rosplan," in *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019, London, UK, July 3–5, 2019, Proceedings, Part I 20*, Springer, 2019, pp. 236–250.

[8] H. L. Younes and M. L. Littman, "Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects," *Techn. Rep. CMU-CS-04-162*, vol. 2, p. 99, 2004.

[9] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," Springrer, Tech. Rep., 2008.

[10] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[11] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[12] Z. Qian, M. You, H. Zhou, X. Xu, and B. He, "Goal-conditioned reinforcement learning with disentanglement-based reachability planning," *IEEE Robotics and Automation Letters*, 2023.

[13] R. He, E. Brunskill, and N. Roy, "Efficient planning under uncertainty with macro-actions," *Journal of Artificial Intelligence Research*, vol. 40, pp. 523–570, 2011.

[14] N. Shah, A. Srinet, and S. Srivastava, "Learning and using abstractions for robot planning," *arXiv preprint arXiv:2012.00658*, 2020.

[15] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 39–48.

[16] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1395–1476, 2021.

[17] G. D. Konidaris and A. G. Barto, "Building portable options: Skill transfer in reinforcement learning.," in *Ijcai*, vol. 7, 2007, pp. 895–900.

[18] L. P. Kaelbling and T. Lozano-Pérez, "Learning composable models of parameterized skills," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 886–893.

[19] J. Wu, X. Sun, A. Zeng, S. Song, S. Rusinkiewicz, and T. Funkhouser, "Learning pneumatic non-prehensile manipulation with a mobile blower," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8471–8478, 2022.

[20] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[21] V. Xia, Z. Wang, and L. P. Kaelbling, "Learning sparse relational transition models," *arXiv preprint arXiv:1810.11177*, 2018.

[22] R. Zhang, C. Yu, J. Chen, C. Fan, and S. Gao, "Learning-based motion planning in dynamic environments using gnns and temporal encoding," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 003–30 015, 2022.

[23] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[24] F. Locatello, D. Weissenborn, T. Unterthiner, *et al.*, "Object-centric learning with slot attention," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 525–11 538, 2020.

[25] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[26] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[27] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[28] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[29] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[30] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2016–2019.

[31] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.