2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Learning to Recover from Plan Execution Errors during Robot Manipulation: A Neuro-symbolic Approach

Namasivayam K*[1], Arnav Tuli*[2], Vishal Bindal*[2], Himanshu Singh[2], Parag Singla[1] and Rohan Paul[1]
[1]Affiliated with IIT Delhi. [2]Work done when at IIT Delhi. * denotes equal contribution.

***Abstract*— Automatically detecting and recovering from failures is an important but challenging problem for autonomous robots. Most of the recent work on learning to plan from demonstrations lacks the ability to detect and recover from errors in the absence of an explicit state representation and/or a (sub-) goal check function. We propose an approach (blending learning with symbolic search) for automated error discovery and recovery, without needing annotated data of failures. Central to our approach is a neuro-symbolic state representation, in the form of dense scene graph, structured based on the objects present within the environment. This enables efficient learning of the transition function and a discriminator that not only identifies failures but also localizes them facilitating fast re-planning via computation of heuristic distance function. We also present an anytime version of our algorithm, where instead of recovering to the last correct state, we search for a sub-goal in the original plan minimizing the total distance to the goal given a re-planning budget. Experiments on a physics simulator with a variety of simulated failures show the effectiveness of our approach compared to existing baselines, both in terms of efficiency as well as accuracy of our recovery mechanism.**

I. Introduction

An essential aspect of robot plan execution is *recovering* from errors. While plans may be perfect in simulation, they are often marred by imperfections in the real world caused by sensor noise, motor failures, unexpected collisions or external disturbances. This results in deviations from the original path of planned execution, which the robot needs to recover from. Once an error is detected, an efficient plan repair is necessary to return to a nominal state in the original plan, accommodating progress made before the error.

Error recovery in classical task planning [1]–[3] detect errors by explicit symbolic reasoning, such as checking the precondition of the next action in the plan, and recover through replanning to the original goal or last correct state. Alternate approaches for planning under uncertainty use reinforcement learning [4]–[6] to learn a reactive policy for the robot to recover from a deviant state to a nominal state. However, learning such a policy requires extensive offline exploration of error states, which is challenging in complex manipulation domains.

Recently, *learning-to-plan* methods have been proposed that predict a plan - essentially a sequence of symbolic actions - to reach a goal specified through language instruction [7]–[10] or logical expression[11]. They are trained in a supervised manner via a dataset comprising demonstrations of goal reaching plans. A key limitation in these approaches is the absence of a feedback mechanism to assess goal attainment, resulting in their inability to detect execution errors.

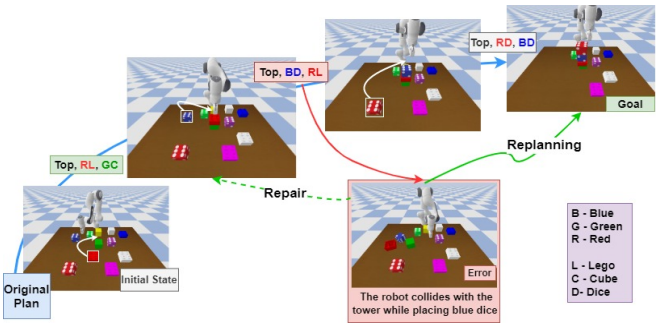


Fig. 1: **Error Recovery Problem.** The diagram illustrates an error resulting from the robot colliding with a partially constructed tower. The states along the blue line represent the expected progression in the plan execution devoid of errors. However, the collision results in a significant deviation from this expected path, requiring error recovery planning.

Even when the model has such an ability to detect whether the goal has been attained, they require additional annotation data to learn such a goal-check function [11]. Further, there is no way to monitor the correctness of the intermediate states during the execution. In such mechanisms, the error recovery can be cast as re-planning *directly* to the goal, albeit *disregarding* prior planning effort.

Our goal is to address the challenge of monitoring and recovering from errors during the execution of multi-step task plans generated by learning-to-plan methods. We aim to learn to detect and recover from errors solely using demonstrations of goal-reaching plans without any need for annotated data of failures. Fig. 1 illustrates a situation where the robot is assigned the task of constructing a tower by stacking blocks on top of each other. During its execution, the robot collides with a portion of the tower, causing the entire structure to collapse. Not only the robot has to be able to detect that error has occurred, it also needs to recover from the error by constructing a multi-step plan to re-stack the blocks from the current (erroneous) state.

In general, designing such an error recovery mechanism opens up two key questions: (a) *"Can we learn a state discrimination function capable of identifying and localising differences between states in a self-supervised way without the need for manually annotated data?"* (b) *"Can we leverage localised information about the failure to efficiently reach the original goal while minimising the replanning time and re-use a part of the original plan as appropriate?"*.

In response, we propose an approach which answers both

the above questions in affirmative. We represent the world as a scene graph over neural object representations. Our key insight is to learn (i) a scene-graph transition model predicting the next scene graph for an input action; providing a foresight into how the scene will look as the plan is executed, and (ii) *neural discriminators* to distinguish between two states as well as object representations. The discriminator trained in a *self-supervised* way serves a dual role: it identifies the objects that are responsible for the failure and serves as a heuristic between states by discriminating object representations. Once an error is detected, the recovery plan is generated through a directed plan search toward a sub-goal in the original plan that is nearest to the error state, while being faithful to the progress achieved prior to the error. The search for the recovery plan is (i) *neuro-symbolic* with explicit symbolic search operating over neural action models and (ii) *discrepancy*-aware using context of erroneous object states during plan search.

The approach is evaluated on a data set of simulated robot execution errors such as grasp failure during transport, adversarial/cooperative interventions in the scene and non-determinism in action outcomes due to motion planning errors. Experiments demonstrate significantly better recovery in multi-step plans marred by sequential errors compared to baselines consisting of an RL-based reactive error recovery mechanism and a fully re-planning based strategy inspired by Mao et al. [11], both in terms of recovery rate, and the time taken to reach the sub-goal for recovery.

II. Related Works

Robotic systems deployed with symbolic task planners [12], [13] monitor plan execution by repeatedly checking for errors between the intended and the actual states encountered. Formalisms such as [1]–[3] synthesize a plan closest to the original or return back to the last correct state. Others, maintain a library of hand-designed recovery behaviours [14], [15] in anticipation of failures switching to recovery behaviours when errors occur. Such strategies inherit the same brittleness as pure symbolic planning due imperfections in extracting state knowledge from sensor data.

Alternative RL-based approaches learn reactive neural policies that prescribe an action for a given state that the robot may encounter [16]–[19]. Notable successes include learning manipulation skills. In this paradigm, error recovery is *implicit* in the learned policy, if during learning, the agent has explored the state space *well enough* so as to generalize to any state that the robot may encounter online. Such generalization is difficult in complex manipulation domains and hence RL-based reactive planners are used for local error recovery or applied in simpler state spaces. Whereas such approaches attain *myopic* short horizon recovery, our work addresses plan repair over a longer horizon inherent in complex manipulation task (e.g., recovery from a fallen stack of blocks).

Recent *learning-to-plan* methods fuse neural representations with symbolic reasoning for generalized long-horizon planning [7], [9]–[11], [20]–[22]. Such planners allow data-driven learning of both spatial and action representations that can be composed for goal-directed reasoning. Error recovery in these models is less studied and largely accomplished by full re-planning to the goal. This paper presents a formalism for error recovery in such models. Our work is also closely related to [22], who explore plan recovery in the context of task and motion planning problems that additionally consider planning in metric space. Their approach predicts the cause of plan failure and performs back jumping to construct a recovery plan. However, this approach uses supervised learning to predict the cause for a plan failure using supervised learning with a data set of failed plans. In contrast, our work ameliorates the need for an explicit corpus of failures.

III. Background and Problem Statement

We consider a robot in a tabletop setting that observes the environment through a depth sensor. The robot is capable of positioning its end-effector at a pose, and grasping/releasing the object at a given pose. The robot is provided high-level goals such as stacking objects of a certain colour, moving objects to a certain region, etc. We assume that the robot possesses a high-level task planner $\mathcal{P}$ that operates on spatial abstractions such as $\mathrm{Top}(), \mathrm{Left}()$ and $\mathrm{Right}()$ and can synthesize a plan as a sequence of actions such as $\mathrm{Move}()$, $\mathrm{Place}()$ etc. Such representations lie at the core of learning-to-plan planning systems such as [7], [8], [10], [11]. Formally, given a high-level goal specification, $\mathcal{P}$ generates a plan $\Pi = (a_1, a_2, .., a_T)$, where a_i is an action from $\mathcal{A}$ and $\mathcal{A}$ denotes the high-level actions. Further, we assume a data set $\mathbb{D}$ consisting of goal-reaching plans.

We model the execution of the given plan as a goal-conditioned Markov Decision Process (MDP), denoted by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}^{env}, S_I, g, \mathcal{R} \rangle$. We assume an object-centric state space: a state $S \in \mathcal{S}$ is characterized by the appearance (colour, material etc) and the metric location of the rigid objects in the scene. The action space comprises high-level actions with discrete parameters, such as $\texttt{MoveTop}(\texttt{a}, \texttt{b})$, intending to achieve the desired spatial relationship between the objects. The plan Π defines an implicit goal g, representing the expected final state in an *error-free* execution from the initial state S_I. However, reaching the expected state in real robot executions are hindered by errors such as action execution issues (e.g., an object falling from the robot's gripper), collateral effects (e.g., disturbances causing objects to fall in a tower-like assembly), or actions by an external agent, such as a human (adversarial or cooperative). The transition model $\mathcal{T}^{env} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ unknown to the robot characterizes these erroneous transitions, determining the next state based on the current state and action during the actual robot execution. The robot's objective is to execute the long-horizon plan Π successfully, reaching the implicit goal state g despite the presence of erroneous transitions $\mathcal{T}^{env}$ in the actual robotic executions. The robot is provided with a sparse reward for goal attainment.

Our approach consists of two key components which are introduced next. The next section details learning architecture

for detecting erroneous states by comparing "imagined" future states with states encountered during execution. Section IV details the efficient synthesis of recovery plan once the error is detected.

IV. Learning to Detect Error States

Our error detection pipeline consists of the following predictors (detailed subsequently) trained using a dataset $\mathbb{D}$ comprising demonstrations of error-free plan executions (without explicit failure annotations).

- *Scene Graph Encoder*, H_ψ, that extracts a object-centric neural state representation from the RGB-D image of the environment.
- *Scene graph prediction function:* $\mathcal{T}_\theta^{ideal}$ that estimates the *intended* changes in the current scene graph when the robot takes an action. Successive application of this function predicts the future state sequence during nominal (error-free) plan execution.
- The *Scene graph discriminator function:*, K_ϕ, that estimates the normalized degree of similarity between any two scene graphs. At every step of plan execution, the intended state on applying an action is imagined using the state transition function, and it is compared with the actual state reached using the discriminator; a significant discrepancy indicates an error.

A. Scene Graph Encoder H_ψ

The state is represented by a neural scene graph extracted from visual data comprising an RGB-D image and object bounding boxes. Following [7]–[9], [21], the scene graph is factored in terms of the objects present in the scene. Visual features of each object are extracted using a pre-trained ResNet [23] based feature extractor [7], [21]. The visual features for every object are concatenated with the bounding boxes (with depth) to include the metric (positional) information. The concatenated feature vector is passed through an MLP encoder (E_N), resulting in scene graph nodes that capture dense object representations For each pair of objects, we form the *edge-embedding* by simply concatenating the corresponding directed pair of *node-embeddings*.

B. Learning the Scene Graph Predictor T_θ^{ideal}

The scene-graph predictor predicts the changes in the scene graph when an action is applied. Actions typically have a local effect on the objects they are applied to. For instance, when a robot moves object A onto object B, only A's location and its relations with other objects change. Specifically, this involves modifying the node embeddings for A and the edge embeddings for edges of the form (A, *) or (*, A). We assume actions in the action space $\mathcal{A}$ manipulate one object at a time. Consequently, the scene graph prediction at any given time step involves predicting a single node change. We observed that node embeddings alone are sufficient to effectively predict scene changes when a single object is moved. Incorporating edge embeddings into the prediction did not improve performance.e.

Fig. 2 describes the architecture of the scene-graph predictor. An action encoder transforms the one-hot action into a dense representation. These action features are then concatenated with the node embeddings of its arguments and passed through the node predictor, which predicts the new node embedding for the manipulated object. The predicted node embedding is fed into both the object and bounding box decoders to reconstruct the object embedding and bounding box of the manipulated object. This reconstructed object embedding and bounding box are compared (using MSE-Loss) with the corresponding gold embedding and bounding box extracted (using H_ψ) from the RGB-D image of the next state in the demonstration. All encoders and decoders are implemented as MLPs, with the weights H_ψ and $\mathcal{T}_\theta^{ideal}$ jointly trained using the obtained loss. The training process takes approximately 10 hours on a single Quadro RTX 5000 GPU with 16GB VRAM.

The trained scene-graph predictor can iteratively imagine the scene graphs of future states encountered during the execution of plan Π. Let $\mathcal{E}(S_I, \Pi) = \mathcal{E}(\mathcal{T}^{ideal}(S_I, a_1), (a_2, a_3, \ldots, a_T))$ be the final *intended* state and $tr(\Pi) = (S_0 = S_I, S_1, \ldots, S_T = S_G)$ denote the *nominal* error-free trace of the plan. These imagined graphs are free from physical errors. Hence, comparing them with the actual scene graph helps identify errors in real robotic execution.

C. Learning the Scene-Graph Discriminator K_ϕ

The scene graph discriminator evaluates the similarity between two scene graphs, predicting either 0 (for dissimilarity) or 1 (for similarity). Since the edges in the scene graph are a function of the nodes, we can discriminate between two graphs by discriminating the two *node* sequences.

Fig. 2 illustrates the discriminator's architecture. Given two scene graphs, their corresponding pairs of nodes are concatenated and fed into an MLP to generate a score in $[0, 1]$. The similarity score between the scene graphs is then calculated as the product of the individual node dissimilarity scores. We train the discriminator after training the H_ψ and $\mathcal{T}_\theta^{ideal}$ using the demonstrations $\mathbb{D}$. Let Z and $\widetilde{Z}$ be the scene graphs of consecutive states in a demonstration, with m denoting the index of the moved object in the action. We treat pairs of the form $((o_Z)_i, (o_{\widetilde{Z}})_j)$ as negative examples (i.e., label $= 0$), for all $i \neq j$. When $i = j$, we treat $((o_Z)_m, (o_{\tilde{Z}})_m)$ as negative example (label $= 0$) and remaining $((o_Z)_i, (o_{\tilde{Z}})_i)$ as positive examples (label $= 1$). Training is done via backpropagation of cross-entropy loss using the generated self-supervised data.

The learned Scene-Graph Discriminator identifies failures during plan execution by comparing the scene graphs of the actual state with those predicted by $\mathcal{T}_\theta^{ideal}$ after each action. The prediction from the network weighed with confidence is used to determine if the current world state deviates from the predicted, necessitating plan recovery, which we address in the next section. Moreover, breaking down graph discrimination into nodes offers local information about the errors. The discrepancy, which represents the number of objects

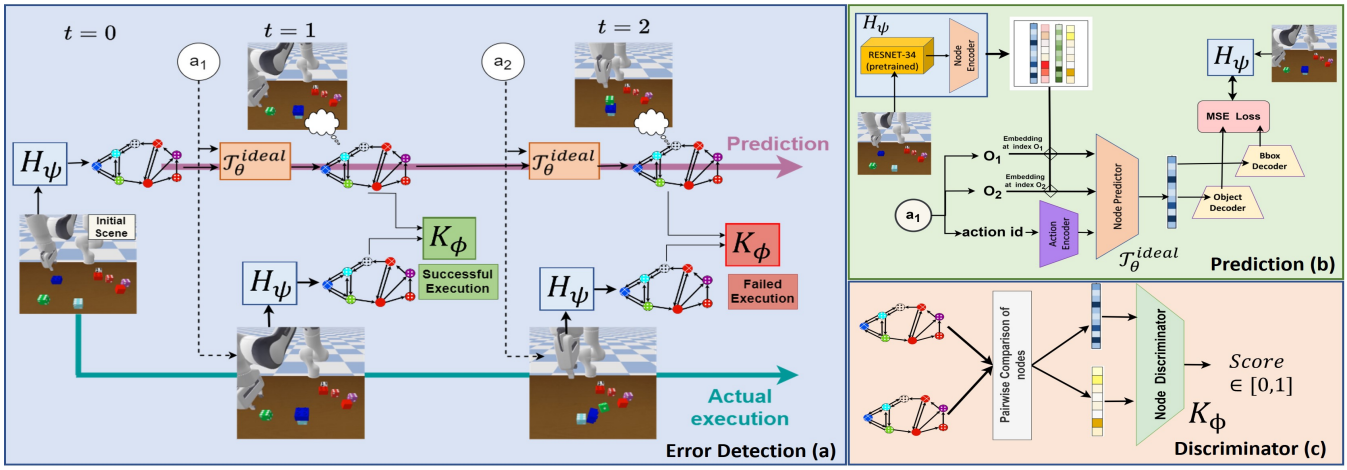


Fig. 2: **(a) Model Overview:** The learnt scene graph predictor $\mathcal{T}_\theta^{ideal}$ is used to imagine the effect of an action execution, which is then compared with the actual scene graph to detect errors. Significant deviations are marked as error by Discriminator K_ϕ, and the error recovery module will be triggered. **(b) Scene graph Predictor:** The architecture of scene-graph extractor and predictor (c) **Discriminator:** Architecture of the discriminator.

displaced between two states, is calculated by aggregating the dissimilarity scores of individual nodes. Formally, the *discrepancy* function $\mathcal{D} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{Z}_{\geq 0}$ is defined as: $\mathcal{D}(S_1, S_2) = n - \sum_{i=1}^{n} K_\phi((o_{S_1})_i, (o_{S_2})_i)$ Here, o_{S_1} and o_{S_2} are the corresponding node embeddings for S_1 and S_2, respectively, and K_ϕ is the learned function predicting 0 or 1 based on the similarity of the two node embeddings. Further, this function also serves as a search heuristic while generating the recovery plan (discussed subsequently).

V. Recovery Plan Synthesis

Given the detection of an erroneous state, we now consider the synthesis of a recovery plan. The recovery is done through a neuro-symbolic search: a symbolic search over the space of neural scene-graphs. Formally, our goal is to determine a *recovery* plan, Π_{S_E} from the erroneous state S_E that reaches a nominal state on the originally intended plan Π, while minimizing the total distance to the original goal subject to a time constraint.

Denote by $\mathtt{P}(S_k)$, the space of all plans Π_K that reach S_k from S_E, i.e., $\mathcal{E}(S_E, \Pi_K) = S_k$. Then the objective is:

$$S_k^*, \Pi_K^* = \underset{\substack{S_k \in tr(\Pi),\\ \Pi_K \in \mathtt{P}(S_k)}}{\arg\min} \Big[\mathcal{C}_{\Pi_K}(S_E, S_k) + \mathcal{C}_{\Pi}(S_k, S_G)\Big], \quad (1)$$

subject to a planning time budget. Here, $\mathcal{C}_\pi(S, S^{'})$ is cost incurred in following the plan π to reach $S^{'}$ from S. The required recovery plan Π_{S_E} is then the set of actions in Π_K^* together with the actions in Π that take from S_k^* to S_G.

In general, we expect our errors to be local in nature, and we would like to exploit much of the original plan, thereby saving on re-planning time. In order to materialize this idea, we identify a set of intermediate state(s) in the trace of the given plan (referred to as sub-goals) $\{S_{t_1}, S_{t_2}, \cdots, S_{t_k}\}$ such that if we can latch on to one of these sub-goals, say, S_{t_l}, then we can simply follow rest of the plan from S_{t_l} to S_G. We begin by explaining how to efficiently re-plan from S_E once a set of sub-goals has been provided. Next, we outline two different strategies for determining these sub-goals.

Given a set of sub-goals $\{S_{t_1}, S_{t_2}, \cdots, S_{t_k}\}$ to latch on, we perform a multi-goal forward search to reach one of the possible subgoals. The minimum of the discrepancies between S_E and the given subgoals is used as heuristics to guide the search. The following strategies are employed to prune the unrelated actions and reduce the branching factor:

1) We construct a Directed Acyclic Graph (DAG), iteratively during plan execution, whose nodes represents the objects, and edges denote actions. Two objects are said to be linked via an action if they appear as arguments in some predicate corresponding to its post condition. This graph aids in prioritizing object recovery by first moving those objects which appear earlier in the topological order specified by the DAG.
2) We introduce the notion of free-space, learned using a transformer encoder network, that allows us to directly predict a collision-free movement without doing an explicit search anytime an object has to be moved. The free-space transformer is provided with the bounding boxes of all objects in the scene, and the ID of the object to be moved, and it outputs a collision-free bounding box. The network is trained in a self-supervised manner where the loss consists of two components (a) MSE between the predicted free space and the bounding box of the object to ensure minimal movement (b) IoU with all other objects ensuring collision-free movement.
3) Finally, we consider only those actions whose arguments are objects that need to be moved as identified by the discrepancy function, further pruning the search space significantly.

We employ two strategies for determining the set of sub-goals to plan for. First, we simply identify the state S_t moving to which the error occurred and simply plan from S_E

to reach this singleton state. While simplistic, and effective in case of simple errors, this will fail in case of more complicated, especially cooperative errors. To address this, we devise the following strategy: we identify the set of sub-goals $\{S_{t_1}, S_{t_2}, \cdots, S_{t_k}\}$, as a set of those states whose distance to S_E is minimum based on the discrepancy based heuristic. In general, the value of k can be chosen based on the time available for re-planning. We can start with a small value of k and keep increasing it until the planning budget is exhausted. The latter strategy is an *AnyTime* recovery mechanism since (1) we increase the value of k iteratively depending on the remaining planning budget (2) we stop sub-goal discovery once the planning budget is exhausted.

VI. Evaluation Setup

A simulated Franka Emika Panda robot [24] in the PyBullet physics simulator is used to collect data set for evaluation. Scenes are synthesized by simulating $3-5$ blocks in the physics engine by randomly sampling sizes, types and metric locations. The neural state transition model and the scene-graph discriminator are trained from 3000 atomic state transitions extracted from goal-reaching demonstrations. A data set of $2000+$ erroneous plan executions (see Table I) is created with the following types of execution errors: (i) grasping failures (e.g., object slipping from the gripper during picking or movement), (ii) erroneous action outcome (e.g., inexact placement of an object on top of another leading to instability and falling of a tower), (iii) motion planning errors (e.g., unexpected collision between the arm and partially constructed assembly due to uncertainty in exact object locations), (iv) actions of an external agent (e.g., a human unexpectedly moves blocks randomly or aids plan progress by pro-actively performing the next action) and (v) explicit errors (e.g., random disturbances or swapping of objects). The set of resulting plans possess $1-10$ steps, the scenes have $5-10$ objects with a maximum of 5 errors introduced at a single step.

TABLE I: **Evaluation Dataset Characteristics**

Dataset	Size	Error Types	#Objects	Plan len
Data set I	750+	(1-5) on random step	5	5
Data set II	1300+	1 on each step	5	1-8
Data set III	200+	5 on random step	(6-10)	5

The proposed approach (**Ours**), is compared with the following baseline approaches. For a fair comparison, the action pre-conditions and association between object bounding boxes (between different states) are assumed to be known.

- **RePlan:** A neuro-symbolic approach inspired by PDSketch [11] that uses a learned transition/goal-check function and performs symbolic forward search to the goal each time an error occurs. For a fair comparison the discrepancy predictor and domain specific heuristic are provided.
- **NoFree:** Our planner with the *discrepancy*-detection function but *without* the *free*-space transformer network.
- **RL-based:** A Soft Actor Critic (SAC) agent was used to learn a goal-reaching policy in the given domain. Note that error recovery in the RL is inherent in the policy execution, given extensive exploration during training.

Finally, we also analyse the plan lengths arising from the anytime variant. Table II lists the evaluation metrics, which include goal-reach-ability, length of the recovery plans and time efficiency of recovery plan synthesis.

TABLE II: **Efficacy and Efficiency Metrics**

	Metric	Description
Correctness	Detection accuracy ($Dec.\%$)	Accuracy in detecting errors using the learnt predictor and discriminator.
	Recovery accuracy ($Rec.\%$)	Accuracy in generating correct recovery plan given the correct error detection.
	Goal completion rate ($Comp.\%$)	Rate of successful goal-reaching executions despite errors.
Efficiency	Recovery plan length (L)	Length of the generated recovery plan from the error state to a given subgoal in the original plan. Usually measured relative to the number of errors introduced (L/N_{err}) and length of the most optimal recovery plan possible (L/L_{opt}).
	Recovery planning Time (T)	Time to generate the recovery plan. Usually measured relative to the number of errors introduced (T/N_{err}) and length of the optimal plan (T/L_{opt})

VII. Results

Our experiments evaluate (i) the effectiveness of the model in error recovery in relation to alternative approaches, (ii) an analysis of model components and (iii) its effectiveness in relation to an increase in plan length and compounding of errors, and recovering from multiple errors.

A. Evaluation of Generated Recovery Plans

We first consider the setting where *multiple errors* are introduced at a *single* step during plan execution. Table III evaluates performance on *data set I*, where 1 to 5 errors occur at a randomly chosen step. Note that recovery accuracy is reported for cases where error detection is correct, and other metrics involving L and T are reported when the recovery plan is correct for all the models. Fig. 3 evaluates the recovery accuracy and planning time against an increasing number of errors. The length of the optimal recovery plan reflects the impact of introduced errors, with longer plans indicating more complex errors. Our model is more effective and generates a recovery plan significantly quicker compared to the other two models, due to its discrepancy-awareness and effective action pruning from the free-space network. A low recovery time leads to a high recovery accuracy, due to the availability of an overall planning time budget ($\sim 30s$).

TABLE III: **Performance Comparison with Multiple Errors at a Single Execution Step**

Model	Rec %	L/N_{err}	L/L_{opt}	T/N_{err}	T/L_{opt}
Ours	**99.61**	1.48	1.06	**0.06s**	**0.04s**
RePlan	98.83	1.48	1.07	0.18s	0.11s
NoFree	87.83	1.46	1.05	0.75s	0.42s

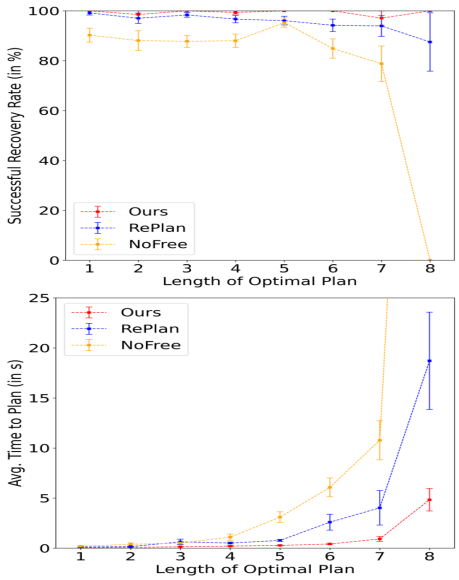


Fig. 3: **Recovery Plan Success with Increasing Errors.** Experiments analyse the recovery accuracy and time to generate a recovery plan with an increasing number of introduced errors. Note that the length of the optimal recovery plan measures the complexity of the errors.

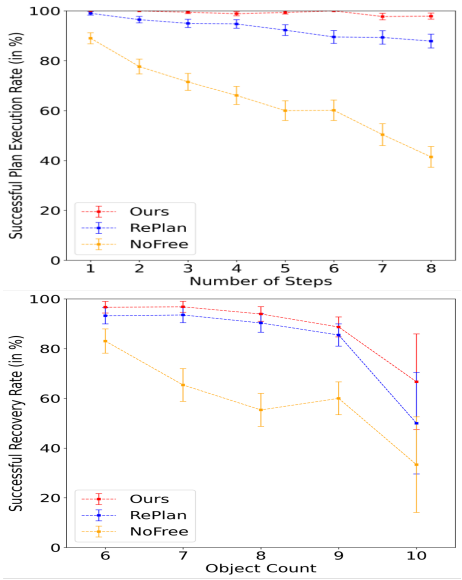


Fig. 4: **Scalability of Recovery Plan Generation Approach.** Experiments analyse the reachability of the final goal in long-horizon plans (top) and with an increasing number of objects (bottom). The proposed recovery method (in red) performs effectively for long-horizon plans and can scale to larger scenes as compared to the baselines.

Next, we consider the setting where *multiple errors* can occur at *multiple steps* during plan execution. Table IV reports the performance of various models on *data set II*, where a single randomized error occurs after each step in the original plan ($N_{err} = 1$). Fig. 4 (top) shows the recovery accuracy against an increasing number of steps. Our model can recover a significantly larger part of the original plan, due to its high recovery accuracy and rapid re-planning ability.

TABLE IV: **Performance with Compounding Errors**

Model	Rec. %	L/N_{err}	L/L_{opt}	T/N_{err}	T/L_{opt}	Comp. %
Ours	**99.85**	2.43	1.06	**0.11s**	**0.04s**	**99.68**
RePlan	98.46	2.42	1.06	0.46s	0.17s	96.67
NoFree	89.50	2.40	1.04	2.43s	0.59s	77.03

Finally, we evaluate the generalization performance of the proposed error recovery model in environments with an increasing number of objects. A large number of objects increase the computation required for both error detection and recovery plan synthesis stages. Table V reports the performance on *data set III*, where the scenes contain $6-10$ objects. We also study the variation of accuracy with an increasing number of objects in Fig. 4 (bottom). The proposed approach generalizes well to larger scenes owing primarily to discrepancy awareness and the use of learned heuristics (e.g., free space sampler) during plan search.

TABLE V: **Performance Comparison on Larger Scenes**

Model	Rec %	L/N_{err}	L/L_{opt}	T/N_{err}	T/L_{opt}
Ours	**93.77**	4.60	1.06	**0.94**	**0.19**
RePlan	92.09	4.59	1.03	1.63	0.29
NoFree	65.35	4.50	1.02	7.02	1.12

TABLE VI: **Recovery Plan Synthesis with Any-time Search**. The use of heuristic-guided forward search to the nearest state in the anytime version results in an improvement in re-plan quality (shorter plans). The recovery plan quality improves with increasing the re-planning budget (K).

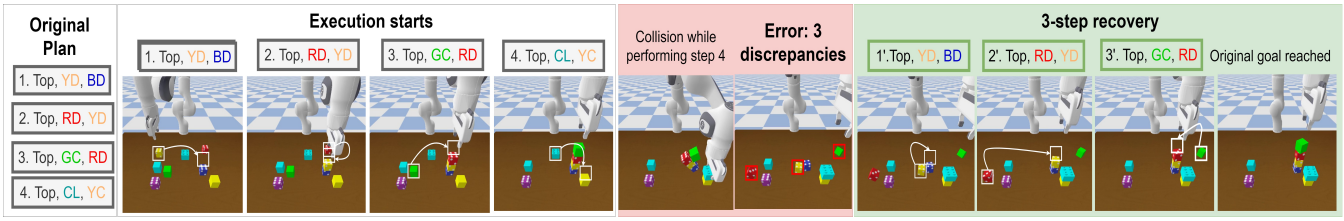
Model	Rec %	L_{plan}/L_{orig}	T/N_{err}
Anytime, K = 1	95.76	0.77	0.10s
Anytime, K = 3	95.07	0.70	0.11s
Anytime, K = 5	94.52	0.64	0.15s

B. Analysis of Anytime Variant

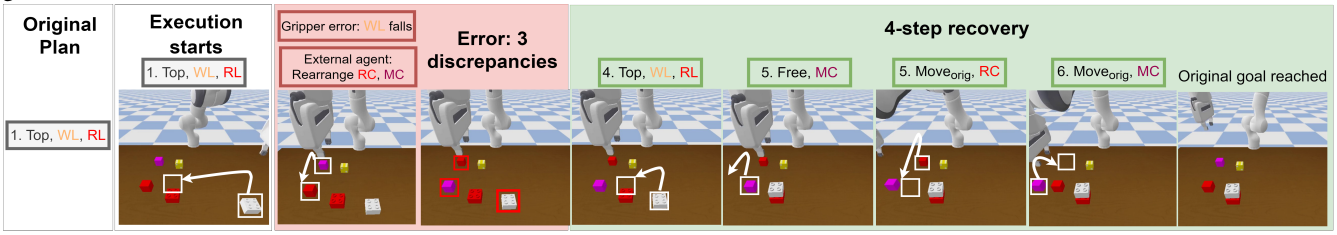
For evaluating the anytime version of our approach, we consider a subset of Data set 3 containing $6-8$ objects. Table VI reports the performance of our original and anytime algorithms with increasing value of the planning budget, K as $1, 3, 5$. Note: in these experiments reported, the heuristic distance from a candidate sub-goal on the original plan and the true goal are approximated as being equal. The evaluation metric L_{plan}/L_{orig} represents the ratio of the length of the recovery plan (for the given anytime version of the algorithm) to the length of the recovery plan in the case of the original algorithm. With increasing K, this ratio *decreases*, indicating more efficient plans. However, the time to plan *increases*, contributing to a smaller recovery accuracy illustrating the inherent length-time trade-off.

C. Qualitative Results

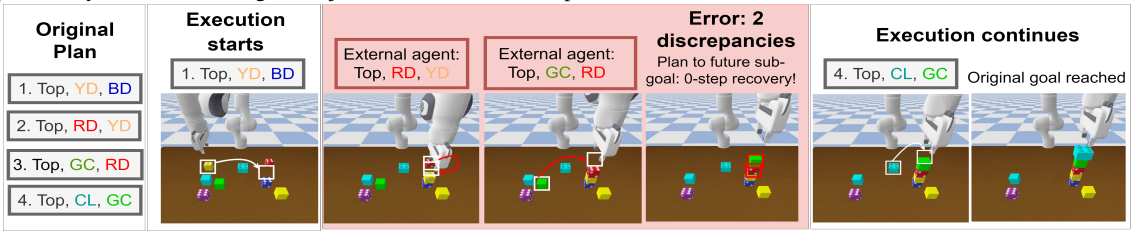
Fig. 5 demonstrates our approach on a simulated 7-DOF Franka Emika Panda Robot Manipulator in a simulated table-top setting. In Fig 5a, the execution of the original 4-step plan is erroneous due to the collision of the robot arm with the



(a) The robot follows a 4-step plan, during which a collision leads to the falling of the formed block tower. A 3-step recovery plan is generated.



(b) Error in robot gripper during object movement, followed by an external agent rearranging two objects, leads to an erroneous state. The 4-step recovery includes moving an object to a collision-free space.



(c) Anytime recovery: An external cooperative agent helps the robot during a 4-step plan by performing two intermediate steps. The robot identifies the constructive change and simply performs the last step of the plan.

Fig. 5: **Demonstration with a Franka Emika Robot in PyBullet Physics engine.** Detection of errors during plan execution and generation of recovery plans, ultimately attaining the intended goal. Errors include (a) unexpected collisions between objects and the robot, (b) grasping failure and dropping of the block during arm movement and (c) unanticipated actions by a (cooperative) human aiding one step in plan progress. *Abbreviations: Y=Yellow, R=Red, G=Green, B=Blue, C=Cyan, W=White, M=Magenta, D=Dice, C=Cube, L=Lego.*

tower of blocks during the last step. The discrepancy between the actual and predicted scene graphs is detected, and a 3-step recovery plan restores the blocks to their original positions. Fig. 5b demonstrates error recovery in an adversarial scenario. During the first execution step, the white Lego blocks fall due to a gripper error, and an adversarial agent swaps the positions of the red and magenta cubes. These discrepancies are identified, leading to the creation of a four-step recovery plan. The magenta block must first be moved to a free space (action $[Free, MC]$) before returning the red cube to its original position (action $[Move_{orig}, RC]$), as directed by the scene-graph predictor.Additionally, Fig. 6 illustrates the prediction of free space in a different scene. Finally, Fig. 5c demonstrates the advantage of anytime recovery. After the first step of a four-step plan, an external agent completes the next two steps. Instead of recovering to the immediately next predicted state and unstacking unnecessarily, the anytime recovery method detects that the state closely matches the sub-goal after three steps, so only the final step is needed.

D. Accuracy of Error Detection

The total error detection accuracy is 94.5% (99% recall and 79% precision) on the evaluation data set. The accuracies of contributing modules are: (i) 88.2% for object matching, (ii) 94.2% for correctly detecting the pre-conditions and (iii) 98% for correct outputs of the scene graph discriminator.

E. Comparison with RL baselines

We also evaluated an RL-based reactive planner [25], trained using *Soft Actor Critic (SAC)* with replay buffer. The learned goal-conditioned policies performed poorly (after 12 hours of training) in terms of goal-reaching rate, attributed largely to the domain complexity. The RL policy learnt had a 6% success rate in predicting single-step plans for scenes containing three objects, as compared to 3.7% by a random planner. When the scene complexity increased to five objects, the recovery rate dropped to 1.45%. In plans with two steps, the planner could not achieve complete recovery, and demonstrated a *partial* recovery rate of 1.88%. For long-horizon plans involving more than two steps, the recovery accuracy remained zero.

VIII. Conclusion

We present a discrepancy-aware neuro-symbolic approach for plan recovery from failures. Unlike existing approaches, we do not require hand-annotated data of failures, rather we

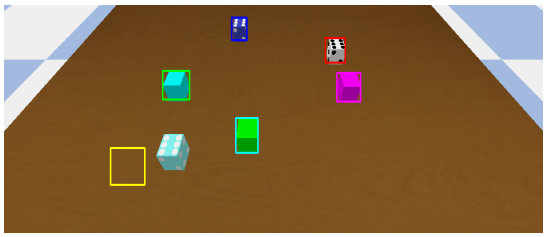


Fig. 6: **Predictions of *free*-space pose prediction module**. The task is to move the *cyan lego* to a collision-free position. The learnt module predicts collision free locations (yellow bounding box) from prior data. The learning accelerates the plan search by ameliorating the need for exhaustive enumeration of possible locations.

make use of self-supervision to train our recovery model. Our approach makes use of object-centric representation of the state in the form a dense scene-graph. We train neural modules to learn the transition function based on data gathered from an existing neuro-symbolic planner. Additionally, we train neural discriminators, trained via the help other states encountered during execution as negatives, to help us distinguish the representations of the simulated state (desired) from the failure states. Once a failure is detected, a recovery plan is constructed to join back the originally constructed plan at an appropriate point. Incorporating exploratory actions in planning, recognizing and communicating to a human when recovery plans do not exist and evaluation on a real robotic test bed remain part of future work.

IX. Acknowledgements

Parag Singla was supported by the IBM AI Horizon Networks (AIHN) grant, IBM SUR awards and Shanthi and K Ananth Krishnan Young Faculty Chair Professorship in AI. Rohan Paul acknowledges the Pankaj Gupta Young Faculty Fellowship for support. We acknowledge the support of the CSE Research Acceleration Fund of IIT Delhi. We would like to thank IIT Delhi HPC facility for computational resources. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views or official policies, either expressed or implied, of the funding agencies.

References

[1] M. Fox, A. Gerevini, D. Long, and I. Serina, "Plan stability: Replanning versus plan repair.," in *Proc. of ICAPS*, vol. 6, 2006, pp. 212–221.

[2] P. Bercher, S. Biundo, T. Geier, *et al.*, "Plan, repair, execute, explain—how planning helps to assemble your home theater," in *Proc. of ICAPS*, 2014, pp. 386–394.

[3] A. Saetti and E. Scala, "Optimising the stability in plan repair via compilation," in *Proc. of ICAPS*, 2022, pp. 316–320.

[4] B. Thananjeyan, A. Balakrishna, S. Nair, *et al.*, "Recovery RL: Safe reinforcement learning with learned recovery zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021.

[5] S. Vats, M. Likhachev, and O. Kroemer, "Efficient recovery learning using model predictive meta-reasoning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 7258–7264.

[6] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *Proc. of ICRA*, IEEE, 2021, pp. 12 618–12 624.

[7] N. Kalithasan, H. Singh, V. Bindal, *et al.*, "Learning neuro-symbolic programs for language guided robot manipulation," in *Proc. of ICRA*, 2023.

[8] R. Wang, J. Mao, J. Hsu, H. Zhao, J. Wu, and Y. Gao, "Programmatically grounded, compositionally generalizable robotic manipulation," in *Proc. of ICLR*, 2023.

[9] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," in *Proc. of ICRA*, IEEE, 2021.

[10] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, and L. F. Fei-Fei, "Regression planning networks," in *Proc. of NeurIPS*, vol. 32, 2019.

[11] J. Mao, T. Lozano-Pérez, J. Tenenbaum, and L. Kaelbling, "Pdsketch: Integrated domain programming, learning, and planning," in *Proc. of NeurIPS*, 2022, pp. 36 972–36 984.

[12] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *Proc. of ICRA*, IEEE, 2013, pp. 855–862.

[13] N. Hudson, T. Howard, J. Ma, *et al.*, "End-to-end dexterous manipulation with deliberate interactive estimation," in *Proc. of ICRA*, IEEE, 2012, pp. 2371–2378.

[14] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy, "Asking for help using inverse semantics," *Robotics: Science and Systems*, 2014.

[15] D.-A. Huang, S. Nair, D. Xu, *et al.*, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," in *Proc. of CVPR*, 2019, pp. 8565–8574.

[16] K. Rana, M. Xu, B. Tidd, M. Milford, and N. Sünderhauf, "Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics," in *Proc. of CoRL*, PMLR, 2023, pp. 2095–2104.

[17] S. Kumar, J. Zamora, N. Hansen, R. Jangir, and X. Wang, "Graph inverse reinforcement learning from diverse videos," in *Proc. of CoRL*, PMLR, 2023, pp. 55–66.

[18] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[19] H. Ryu, M. Yoon, D. Park, and S.-E. Yoon, "Confidence-based robot navigation under sensor occlusion with deep reinforcement learning," in *Proc. of ICRA*, IEEE, 2022, pp. 8231–8237.

[20] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Proc. of CoRL*, PMLR, 2022, pp. 894–906.

[21] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, "The neuro-symbolic concept learner: Interpreting scenes words and sentences from natural supervision," *ArXiv*, vol. abs/1904.12584, 2019.

[22] Y. Sung, Z. Wang, and P. Stone, "Learning to correct mistakes: Backjumping in long-horizon task and motion planning," in *Proc. of CoRL*, PMLR, 2023, pp. 2115–2124.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR*, 2016, pp. 770–778.

[24] S. Haddadin, S. Parusel, L. Johannsmeier, *et al.*, "The franka emika robot: A reference platform for robotics research and education," *IEEE Robotics & Automation Magazine*, vol. 29, no. 2, pp. 46–64, 2022.

[25] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *Proc. of ICRA*, IEEE, 2020, pp. 4051–4058.