Learning Robot Manipulation Programs: A Neuro-symbolic Approach

Vishwajeet Agrawal*, Rahul Jain*, Parag Singla and Rohan Paul
Indian Institute of Technology Delhi, India
* Equal Contribution

***Abstract*—We address the problem of learning a model for neuro-symbolic robotic manipulation. Given the data in the form of triplets, with each triplet representing (a) a natural language instruction (b) an input scene, (c) an output scene, our goal is to train a model which when presented with a natural language instruction, and an input scene, can output a program (composed of learned actions and reasoning on the input scene), explaining how the input scene can be affected to result in the output scene. The inferred program also provides sequential sub-goals for the robot's low-level motion planner to complete the intended task. Unlike previous approaches, which represent the program to be executed as a sequence of action labels [15], our model works with explicit neural representation for actions (as a transformation to objects in a latent space), and is amenable to composition with pure symbolic reasoning. This makes our approach highly modular, interpretable, and generalizable to unseen settings. From a modeling perspective, we first parse the natural language command in the form of a symbolic program, which is then executed over the input scene, using the neural representation of various symbolic concepts. The key building blocks of our architecture include, training of the parser via reinforcement learning, and the use of *gumble-softmax* operation to work as a selector, while still allowing for back-propagation, facilitating disentanglement in the action space. The entire system is differentiable end-to-end with no intermediate supervision. Our experiments on a simulated environment, with commands consisting of variations, such as single actions, multiple actions, and scenes consisting of objects with varying attributes, demonstrate that our model is robust to all these variations, and significantly outperforms the existing baseline.**

I. Introduction

Constructing robots which can learn to act and achieve the intended goal based on natural language instructions is a desirable task to achieve. Applications can be found in several domains, including, home navigation and control, working in an assembly line, and a general purpose robotic assistant. The task is hard due to multiple reasons: (1) grounded actions have to be parsed from the underlying sentence requiring complex natural language reasoning (2) the affect of executing these actions has to be deciphered on the underlying scene, again requiring complex reasoning now at the image level. Further, if the supervision is provided as an image of the final target scene, the model must learn a representation for intermediate actions to be executed for achieveing the desired affect.

In response, a series of approaches have been proposed in the literature which address this problem, by inferring latent actions to be executed and require a representation for low-level robotic actions. Many of these approaches treat actions simply as 'labels' output by the model, without any deeper semantics. For example, Prospection framework [15] achieves this by converting the natural language sentence into a series of sub-goals that the robot needs to execute to achieve the task. The sub-goals are represented in the form of actions and their parameters represented as a set of labels output by the model. Additionally, they predict the future state if the robot had taken a certain action which allows them to map actions to sub-goals. Hristov et al. [3] solve a task which is similar in nature to ours, but still does not work will full natural language semantics. They take the latent space representations of various concepts in demonstrations, and map it to a set of pre-defined relational concepts, such as left, right etc. This allows them to later specify a starting image, and a set of relational concepts of the end scene extracted from a demonstration, which are then used to predict the end effector poses using supervised learning without learning embeddings for specific actions. Other efforts such as Neural Task Graphs [4], learn to complete a given task given a single video demonstration. Gredila et al. [10] present a framework for direct extracting concepts from a set of demonstrations, which are then translated into executable programs. It is shown that the learned concepts can now be applied to novel settings in a zero-shot learning framework. Zhu et al. [23] propose to achieve long term planning using a hierarchical scene graph representation of the underlying image. The manipulations are achieved through the use of a Graph Neural Network which allows them to capture various interaction between the objects in the scene. These efforts focus on learning semantic concepts from demonstrations without incorporating language instructions are and do not learn representations for robot actions.

In this work, we build on the current state-of-the-art, and propose a model for grounding of concepts in a natural language instruction, to those present in the image, and translate them into actions specified as part of an executable program. Our action representation is purely neural, and our model results in a disentangled representation for actions. The output of our model is a program, which when executed by the robot, results in the desired world state. Our model is trained end-to-end without any intermediate supervision. The concepts in our model, including actions, are represented using a pre-defined *Domain Specification Language (DSL)*. Broadly, our model consists of the following three modules: (a) *Language Reasoning* module consists of a parser, which identifies various concepts from the natural language instruction using a hierarchical parser based on [1]. The parser is trained in an end-to-end

manner using REINFORCE [19] and can handle multi-step commands. (b) *Visual Reasoning* module consists of an object extractor, which learns the concept embeddings for various attributes, such as various colors, which are then used to filter objects of interest based on the concepts extracted from the language using parser. This helps ground the program output by language reasoner with specific object references (c) *Action Simulator* takes the locations of grounded objects identified by the visual reasoning module, and applies the desired action on the object(s) present at those position(s), resulting in updated location of the (first) object argument. In this process, a neural representation (disentangled) for each action is learned, which transforms the input object location into a new one, based on the semantics of the action to be executed and reference (argument) objects locations. Since our action representations are disentangled, we need to backpropagate over an action selection box, which is done using gumble-softmax function. Our approach is *neuro-symbolic* since it operates with both the neural (dense) representation of various concepts (and actions), and their symbolic representation as executable programs. Our language reasoning and visual reasoning modules are inspired by recent work [12] which proposes a neuro-symbolic concept learner for the task of visual question answering. Whereas they stop at the task visual question answering on a given image, we take the idea one step further, by actually allowing for manipulation of objects using learned action representations, possibly requiring a sequencing of multiple manipulation steps as shown in the following example.

Figure 1 shows the example execution for the natural language command *"put small green block to the right of green lego block, then put magenta block on small green block."*. The end-to-end process involves, processing the natural language command to extract concepts such as `small`, `green`, `lego`, as well as identifying the action to be executed, resulting in the parse corresponding to a two-step program (language reasoning), identifying the relevant objects in the image based on the filtering of attributes and extracting their positions, and then grounding the program based on filtered objects (visual reasoning), and finally passing the action coming from parser, and locations of object arguments, to compute the new location of object being operated on in the resulting scene, as well as sequence of sub-goals for the robot to execute (action simulation). Since, this is a multi-step command, processing it involves sequencing of multiple manipulation steps, i.e., the filtered objects corresponding to the second command will be operated on after the first manipulation step is complete.

The contributions of this work are: (i) a neuro-symbolic model for explaining human demonstrations as grounded robot manipulation programs; (ii) learning grounded dense representations for robot manipulation actions; (iii) a novel loss function and a selection mechanism based on gumble soft-max to facilitate disentangling in the action space. (iv) simulation experiments demonstrating generalization to novel settings. The code and data set is available online.[1]

[1]https://github.com/dair-iitd/nsrmp

II. Related Works

Action Representations for Task Planning. The ability to reason and plan tasks is conditioned on the robot possessing knowledge of when it can perform actions and how it affects the world state. Traditional approaches model robot actions as symbolic pre-conditions and effects [7]. Such representations are often hand-coded making them brittle and error-prone in practice. Learning based efforts aim at acquiring actions representations through human demonstration or via self-supervision. Other efforts learn the *initiation set* for an actions; implicitly learning to classify regions in the robot's configuration space where an action can possibly be initiated [8, 18]. Zettlemoyer et al. [22] address the complementary problem of acquiring the transition model and present an algorithm for determining *planning rules* expressing possible symbolic states resulting from the robot performing an action. Xia et al. [20] present an learn the a symbolic transition function in the building on an object-centric world model for tasks such as block stacking. The afore-mentioned approaches either (i) forgo learning the grounded representation focusing only on inferring symbolic associations or (ii) do not possess executable or modular structure that can be composed arbitrarily for reasoning tasks.

Language Grounding to Robot Control. Related efforts address the problem of inferring manipulation constraints [2, 13] that capture the intended goal from language utterance for the purposes of commanding a robot. Prominent models such as learn to associate linguistic constituents with a symbolic representation of the environment. The likely association or the *grounding* for the instruction is used to derive a motion plan for the robot. Such approaches assume the presence of spatial concepts and focus on learning the association between language and they physical world state. The work presented in this paper, learns such spatial and action concepts directly from natural supervision forgoing the need for explicit hand coding of concepts. Other related efforts [14, 16] learn groundings for actions and spatial relations assuming a set of hand-coded features or directly from data but lack an explicit representation of symbolic reasoning which may be needed for interpreting an instruction. The explicit notion of a program space used in this work makes learned grounded concepts amenable to symbolic reasoning. Paxton et al. [15] introduce a recurrent architecture that translates language instructions into a sequence of sub-goals for the manipulator to follow. Central to the approach is a recurrent architecture that learns to predict sequential changes in the world model caused by robot actions. Although the model makes significant progress in predicting future world states, the model lacks the notion of grounded and executable representation for actions. As a result, the generalization performance degrades with tasks possessing repetitive structure that are unseen in training.

Neuro-symbolic Concept Learning. This paper builds on the neuro-symbolic concept learning framework [12] that enable learning of concepts such as object attributes and spatial relations in visual question answering (VQA) dataset. Our

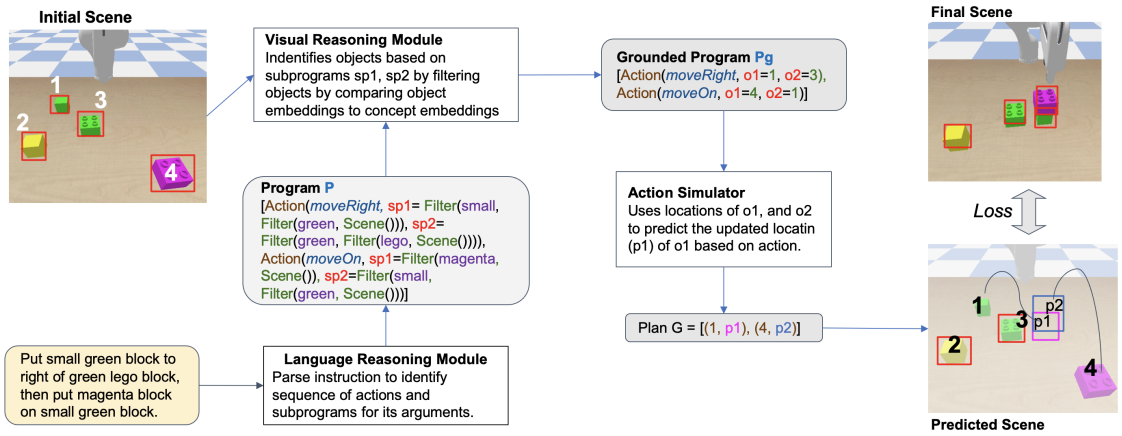

Fig. 1: Neuro-Symbolic Manipulation. The figure shows three core modules in our approach (a) Language Reasoning Module (b) Visual Reasoning Module (c) Action Simulator. In the first step, natural language instruction is input to the language reasoner, which outputs a Program (P), i.e., a sequence of parsed instructions. This program P, along with the visual scene (S_I) is passed to the visual reasoner, which filters relevant objects from the scene, and uses them to the output a grounded program P_g in which filter commands have been replaced by specific object references. the action simulator takes the action identifier (from the parser), locations of object arguments as output by visual reasoner, and produces resulting location of argument 1 object of the action command. It also results in outputting of sub-goals (o, p), which is pair of object and position combination denoting that object o has to be moved to position p during actual robot execution. During training, loss is back-propagated by comparing the location (bounding box) of each object in the final scene with those predicted by the model.

approach introduces the physical agent affecting the scene and focuses on learning a deep representation for robot actions that affect the environment. Lázaro-Gredilla et al. [10] take a cognitively-inspired approach for inferring symbolic programs for modifying block patterns to a target configuration. However, this work assume a deterministic grounding from sensory inputs to discrete symbols. In contrast, this work builds an action representation directly on the perceived objects in the manipulation area. Shah et al. [17] learn LTL task specifications from humans demonstrations and introduce a model for effectively searching in the space of programs. However, the grounding for symbolic program constructs are assumed known. In contrast, this paper learns executable concepts that can be reasoned when combined in programs, allowing synthesis of world state after program execution. More generally, neuro-symbolic approaches have also found applications in modeling scene dynamics [21], inferring repetitive visual structures [11] and motion programs [9] etc.

III. Problem Formulation

Robot and Environment Model. Consider a robot manipulator operating in a table-top environment populated with a set of rigid objects such as blocks or trays. The robot perceives the world state via a depth sensor that outputs a depth image $S \in \mathbb{R}^{H \times W \times C}$, where, the superscript denotes the height, width and the number of channels (including depth) of the imaging sensor. Further, the robot's perception system includes object detector that can identify object proposals and determine bounding boxes for object instances detected in the environment. Let $b_i \in \mathbb{R}^4$ represent a bounding box for the i^{th} object represented with two 2D coordinates of the diagonal end-points. Finally, the data association between observed bounding boxes and object instances is assumed known in images across time. The workspace is co-habited by a human partner who provides language instructions Λ such as *"put the blue block on the red block"*, *"put the small green block to left of yellow lego block"* etc. for robot to perform assembly tasks. Assuming a closed world setting, changes to the world state are caused by the robot manipulator, thereby ignoring any exogenous changes.

Language-guided Task Execution. The robot's goal is to interpret the human's instruction Λ in the context of the initial world state S_I and determine a sequence of low-level motion motions that result in the final world state S_F conforming to the human's intention. Performing an instructed task can involve complex interactions requiring an extended sequence of low-level motions to perform a long-horizon objective (e.g., completing a stack assembly of blocks). Following [6, 23], planning a complex task is factorized into (i) high-level task planning to determine a sequence of sub-goals and (ii) the generation of low-level motions to attain each sub-goal. Formally, the task planning model $TaskPlanner(.)$ takes the initial scene S_I and the instruction Λ as input and determines a sequence of sub-goals as $(g_0, g_1, \ldots, g_n) = TaskPlanner(S_I, \Lambda)$. Each sub-goal $g_i \in \mathcal{G}$ aggregates the knowledge of the object to be manipulated o_i and its target Cartesian $SE(3)$ pose p_i and is provided to the low-level motion planner to synthesize the end-effector trajectory to execute. The robot's motion planner includes grasping an object and synthesizing a collision-free trajectory for positioning it at a target pose (on the table or on top of another object). Online, when the robot is instructed, it uses the learned model

to interpret the instruction as per the current world state and predicts sub-goals which are then sequentially attained.

Learning Task Planning Model. This paper addresses the problem of learning the task planning model introduced above. Let the task planning model be realized as a function parameterized with trainable parameters Θ. During training, the human provides a demonstration for a task (consisting of a sequence of object manipulations) in the form of the initial and the final world states (S_I, S_F) and the task instruction Λ. Given a data set $D = \{S_I^i, S_F^i, \Lambda^i\}_{i=1}^M$, the model parameters are trained by optimizing a loss $\mathcal{L}(D; \Theta)$. The overall loss aggregates $\mathcal{L}(\tilde{S}_F^i = Simulate(TaskPlanner(S_I^i, \Lambda; \Theta)), S_F^i)$ for the i^{th} datum, where $\tilde{S}_F^i$ is the final state estimated by simulating the plan inferred by $TaskPlanner(S_I^i, \Lambda))$ on intitial state S_I^i. Further, we seek strong generalization on novel world scenes and instructions beyond those encountered during training and interpretability in sub-goals for a task.

IV. Technical Approach

This section discusses how the task planning model introduced in the previous section can be realized as an end-to-end trainable neuro-symbolic architecture possessing strong generalization and interpretability. Given the challenge of directly inferring the sub-goals from a natural language instruction, motivated by recent work [12], we take a neuro-symbolic approach for this problem. We start with a set of pre-identified concepts, which will be used to ground the concepts in the natural language to the attributes of specific objects present in the visual scene (S_I), so that desired objects can be identified and filtered for further processing. These concepts are specified in the form a *Domain Specification Language (DSL)*. We will use the word concepts to refer to the concepts related to object attributes as well as specific actions in the DSL. The rest of our pipeline consists of three stages.

In Stage I of our pipeline, referred to as *Language Reasoning Module*, we use a hierarchical parser [1] to map the tokens in the natural language instruction to a program in the DSL consisting of its operators and concepts. When the instruction consists of a multi-step command, they are broken (automatically) into multiple single step commands by the parser, before extracting the concepts, and passing them to the next stage. In Stage II of our pipeline, referred to as the *Visual Reasoning Module*, we ground the objects in the scene, by identifying the relevant concepts (object attributes) from the program extracted in Stage I, and then filtering the desired set of objects based on the extracted concepts. In Stage III of our pipeline, referred to as *Action Simulator*, we take an action, as extracted by our language parser, sets of its arguments, i.e., locations of objects to be operated on as output by Stage II, and output new bounding box for the first object argument, [2] performed iteratively at every time step t of a multi-step instruction. The entire operation results in a series of sub-goals to be achieved by the robot during its execution, which was the original objective that we started with. The first two stages of our pipeline are inspired from Mao et al. [12], whereas the final stage is entirely novel, and the central contribution of this work.

[2]This work assumes binary actions, where only the first object is being moved, and second gives a reference, but this can easily be relaxed.

TABLE I: Domain Specification Language (DSL)

Attributes. Type: ValSet • Color: {red,green,blue,magenta,yellow,cyan,while} • Size: {lego, small}
Actions • {MoveOn, MoveRight, MoveLeft}
Operators • Scene() → vec //*return a vector of 1's of size # of objects in scene* • Filter(vec1, val) → vec2 // *return vec2 after Filter on vec1 wrt val* • Unique(vec1) → index // *returns index via gumbel-softmax on vec1* • Action(aid, o1, o2) → loc //*execute action indexed aid on o1, o2*
NOTE: vec is a vector of 1's. vec1 (input), vec2 (output) represent vectors of scores ($\in [0, 1]$) for a subset of objects based on (prior) matching on some attribute values. loc represents location of obj1 after executing action

A. Domain Specification Language

Our DSL is described in Table I. For each attribute type, DSL defines a set of values that it can take. These attribute values form the concepts in the DSL. We assume that attribute typing is known to us, i.e., for each attribute type, we exactly know the set of values it can take. In our current formulation, we work two attribute types, i.e., color and size, but the formulation is extensible to any finite number of attribute types, and corresponding sets of values. We also have a symbol (*ActionId*) corresponding to each action. In addition, the DSL consists of the following operators: `Scene, Filter, Unique, Action`, whose semantics is given in Table I. The semantics/implementation of these operators are detailed in sections where they are used, i.e., in visual reasoning (Section IV-C) for first three operators, and in action simulator (Section IV-D) for the last one.

B. Language Reasoning Module

From the instruction Λ, the parser outputs a program P as a sequence of one-step programs $[\Pi_1, \Pi_2, ..., \Pi_T]$, each representing an action execution.

$$[\Pi_1, \Pi_2, ..., \Pi_T] \equiv \leftarrow \texttt{LanguageReasoner}(\Lambda; \Theta_l)$$

Each Π_t is of the form Action(a_t, sp_1, sp_2), where a_t is ActionId and sp_1 refers to the sub-program whose output is the object to be acted upon, and sp_2 refers to the sub-program whose output is the object in relation to which the first argument (object) needs to be acted on. These sub-programs are typically represented as a sequence of filter operations. We use the ideas detailed in semantic parser of Dong and Lapata [1], and later used by Mao et al. [12]. We train our parser by adding the concepts corresponding to actions executions, which is not available in Mao et al.'s work. We also need to modify the parser to work for multi-step commands, which is described after we explain single-step commands.

Take the single-step command, *"put small green block to the right of green lego block"*. Then, the resulting program output by our parser would be: $\texttt{Action}(\texttt{MoveOn}, sp_1, sp_2)$, where sp_1 is $\texttt{Unique}(\texttt{Filter}(\texttt{small}, \texttt{Filter}(\texttt{green}, \texttt{scene}())))$, and sp_2 is $\texttt{Unique}(\texttt{Filter}(\texttt{green}, \texttt{Filter}(\texttt{lego}, \texttt{scene}())))$. Like Mao et al., we assume that concepts corresponding to attribute values are uniquely identified in the language. Further, we assume that all the attribute related concepts qualifying a particular object appear together in the instruction [3]. This allows us to process the concepts *a priori*, and replace them by concept identifiers treated as a single token. For instance, the instruction above gets processed to (*"put* [[X]] *block to the right of* [[Y]] *block"*, X=[small, green], Y=[green, lego]). The modified instruction is passed through a GRU-based encoder to get the hidden representation for each token, and the final instruction embedding $emb(\Lambda)$ which is passed through an action decoder to get a logit vector of the size of the total number of actions. This is passed through a gumbel-softmax operation [5] to implement a selector over possible actions. To obtain the sub-program sp_l ($l \in \{1, 2\}$), we follow the approach used by Mao et al. [12] - the instruction embedding $emb(\Lambda)$ is passed through another decoder, one for each l, to get an embedding c_l which is then compared (through a dot product) with embeddings of each concept token (embeddings of [[X]] and [[Y]] in the above example) to get a probability over the tokens, from which one of them is sampled.

In the current implementation, since we only have filter operation which selects objects based on attribute values, the concept tokens are directly translated into the corresponding program. E.g., [small, green], is translated into $\texttt{Unique}(\texttt{Filter}(\texttt{small}, \texttt{Filter}(\texttt{green}, \texttt{scene}())))$. Since, this part of the parser requires selecting concept tokens, it is trained using REINFORCE through a reward signal depending on the final loss term (described later) between predicted and ground final world state S_F. Extending our parser to process multiple level of hierarchy in the program is a direction for future work.

Finally, to handle multi-step instructions, in the current version, we assume each one-step instruction appears consecutively, one after another (separated by 0 or more tokens). We use an encoder-decoder setup that at each step, selects a position at which the complex instruction needs to be split [4]. This is done recursively until the complex instruction is completely broken into one-step instructions. These single-step instructions are then passed through the pipeline as detailed above to generate corresponding one-step programs. This part of the parser, splitting multi step into single step instructions, is also trained using REINFORCE through the common reward signal depending on the final loss term. As mentioned earlier, the action selection is handled using gumbel-softmax operation, and so the parser also gets a back-propagation signal.

[3]Relaxing this assumption, and allowing for linguistic variations is a direction for future work.

[4]Extending our formulation where multi-step instructions are constructed by interleave single-step instructions is a direction for future work

C. Visual Reasoning Module

The visual reasoning module aims at substituting the argument sub-programs sp_1, sp_2 for each $\Pi \in \{\Pi_t\}_{t=1}^{T}$, with objects from the scene to get a grounded program, which in turn is a sequence of grounded one-step programs, $[\pi_1, \pi_2, ..., \pi_T]$, where each π_t is of the form $\text{Action}(a_t, o_{t1}, o_{t2})$, where o_{t1} and o_{t2} are references to objects acted upon at time step t.

$$\pi_t \leftarrow \texttt{VisualReasoner}(S_I, \Pi_t; \Theta_v) \quad \forall t$$

We note that our image representation already assumes that we have object bounding boxes available with us (see Section III). For each object, we extract a r-dimensional dense embedding $emb(o_i)$ by cropping the image according to its bounding box and passing through a CNN; $emb(o_i) \leftarrow \texttt{ConvNet}(\texttt{Crop}(S_I, b_i))$, where b_i represents the bounding box for object o_i. We refer to location, loc_i of an object i in the image space as the concatenation of the bounding box, b_i and mean depth, d_i of the object. The depth information is required to uniquely map a location in the 2D pixel space to robot's world coordinates. For each attribute type u (e.g., $u \in \{color, size\}$ for our setting), we have a neural operator f_u^v that maps the dense object embedding $emb(o_i)$ to a vector $f_u^v(emb(o_i))$ in the k-dimensional attribute value concept embedding space. For each attribute type taking value=$\texttt{val}$, we learn a corresponding embedding $emb(val)$ in our model. There is an embedding for each color, e.g., `red, green, blue`, etc. in our model, and similarly for each size, i.e., `small` and `lego`. When a sub-program requires to filter the objects with attribute of type u with value $\texttt{val}$, then the degree of similarity between the embedding of the corresponding attribute value in each object o_i, i.e., $f_u^v(emb(o_i))$ and embedding of the attribute value concept, $emb(val)$ is computed using cosine-distance, and stored in a vector, one element for each object. Later, when another filter operation is applied, for each object index i, minimum of the two vector elements is kept at each index i. In the beginning, $scene()$ operations returns a vector with all $1's$, representing that initially all objects are preferred equally in the absence of any filter operation. After all the Filter operations are done, Unique operation outputs the index corresponding to the highest of these elements using a gumbel-softmax operation to keep the entire operation differentiable. These operations are also described in the first three rows of Table II. Note that most of these operations are similar to the ones implemented by Mao et al. [12]. While they use it for the downstream task of VQA, we feed these into the next stage of our pipeline which is action simulation.

D. Action Simulator

Visual reasoning module gives us a sequence of grounded one-step programs $P_g \equiv [\pi_1, \pi_2, ..\pi_T]$, where $\pi_t = \text{Action}(a_t, o_{t1}, o_{t2})$. In the last stage of our pipeline, π_t's are executed sequentially starting from the initial scene S_I, and at each step, updating the scene and producing the sub-goal g_t, i.e., the target pose p_t for object o_{t1}.

$$S_{t+1}, p_t \leftarrow \texttt{ActionSimulator}(S_t, \pi_t; \Theta_a)$$

TABLE II: Implementation of various operations

Signature	Implementation
Scene() $\rightarrow vec$	$vec[i] := 1, \forall i$
Filter($vec1$, val) $\rightarrow vec2$	$vec2[i] := \min(vec1[i], \texttt{cos\_sim}(f_u^v(emb(o_i)), emb(val)), \quad \forall i.$ Here, $u = AttrType(val)$
Unique($vec1$) $\rightarrow$ id	$id = \texttt{gumbel}-\texttt{softmax}(vec1)$
Action(aid, o1, o2) $\rightarrow$ loc	$loc := f^{aid}(\texttt{onehot}(aid)\|\|l_1\|\|l_2)$ $l_1 := loc(o1), l_2 := loc(o2)$

Here, S_t denotes the scene at time step t. Note that $S_I \equiv S_1$. S_{t+1} differs from S_t only in terms of the location of object o_{t1}. The sequence of these executions at every time step t, leads to the final scene $S_F \equiv S_{T+1}$, the scene obtained by executing actions up to time step T. As a by-product of this process, we can read-off the sequence of sub-goals $[g_1, g_2, \cdots, g_T]$, which can then be passed to the robot for execution at inference time. The computation of the loss in detailed in Section IV-E.

We next describe how $\pi_t \equiv \text{Action}(a_t, o_{t1}, o_{t2})$, is implemented in our model. The implementation is identical $\forall t$, so we can describe this in generic terms for any given t. Action execution is achieved using a neural operator f^{a_t} (an MLP) that takes the one-hot representation of action id a_t, and the current locations of objects o_{t1}, o_{t2} (i.e., at step t) as input and produces the new location of moved object o_{t1} as follows:

$$loc(o_{t1}) \leftarrow f^{a_t}(a_t||loc(o_{t1}||loc(o_{t2}); \Theta_a)$$

Also, see the last row of Table II. We note that since we are passing in the one-hot representation of actions (due to the gumbel-softmax operation at the end of action decoder of the parser), our implementation results in a disentangled representation for actions by design. This results in better interpretability as well modular nature helping in better generalization. Finally, the new location $loc(o_{t1})$ is transformed to a position p_t in the world space using a pre-learned function as described in Section III. This gives us a sub goal $g_t = (o_{t1}, p_t)$. The updated location is also used to transform the scene which is to be operated at the next time step. Thus the whole program P_g is executed sequentially giving us with the required plan $G = [(o_{11}, p_1), (o_{21}, p_2), ...(o_{T1}, p_T)]$.

E. Loss Function

We do not have direct access to the true plan G. We also do not have access to intermediate scenes S_t, $2 \leq t \leq T$. The only information we have is how the final scene $S_F \equiv S_{T+1}$ looks like. Therefore, the loss must be computed in terms of S_{T+1}. Let $loc_i \equiv (b_i, d_i)$ denote the true location (in the image space) of object i, in the final scene S_{T+1}. Recall that b_i, d_i are bounding box and mean depth of the object i respectively. Let $\widehat{loc_i}$ denote the final location of object i as predicted by our model. Then, we compute the loss (for object i) $\mathcal{L}(\widehat{loc_i}, loc_i)$ as a combination of two terms, $w_1 * \texttt{MSE}(\widehat{loc_i}, loc_i) + w_2 * (1 - \texttt{IoU}(\widehat{b_i}, b_i))$. The first term is the mean squared error between true and predicted locations, and the second term is one minus intersection over union (IoU) between the true and predicted bounding boxes. w_1, w_2 are hyper-parameters determining the important of two loss terms, and are set using a validation set. The total loss, L, is simply average of the loss over each object, i.e., $\sum_{i=1}^{N} \mathcal{L}(\widehat{loc_i}, loc_i)/N$. The reward signal passed to the parser is computed as, $R = R_0 - L$ where R_0 is a hyper-parameter. Recall location is concatenation of bounding box b and mean depth d.

F. Training and Inference

Since there are multiple stages of our pipeline, and loss is available only it at the end, it is important to define a curriculum for the training to be effective. In particular, we need to first train in simpler settings, followed by freezing of certain modules, and then moving on to more complex instructions. Such curriculum training has been found to be effective in prior neuro-symbolic approaches [12]. We perform our training using the following steps (a) We first train on single step commands, and with only selection on a single attribute type (e.g., color or size) for any given object. This allows our action simulator to learn disentangled action representations. (b) In the second step of curriculum training, we allow for instructions with selection on multiple attribute types. In this step, the action simulator is kept frozen; this can be done since action simulator does not directly depend on the linguistic variations or the number of attribute types being used to qualify the objects (c) In the last step, we allow for instructions involving multiple steps. In this step of curriculum training, we freeze the rest of the pipeline, and only train the parser component which splits a given instruction into multiple single step instructions. This can be done, since rest of the pipeline can operate as earlier, once a multi-step instruction has been split into its respective single-step equivalents. The entire model is then fine-tuned jointly after curriculum training.

During inference, the learned model inputs a natural language instruction and a starting scene. The model first outputs a latent program by parsing the instruction using language reasoning module. The visual reasoning module uses this latent program and the initial scene, to ground the objects mentioned in the latent program. Finally, the grounded program is passed to the action simulator to output sub-goals for robot execution.

V. Evaluation and Results

A. Data set Generation

A Franka Emika Panda manipulator operating in a PyBullet simulation environment was used to collect an evaluation dataset. The work space consisted of a table top with blocks of different sizes and colors, with two possible size attributes (a small block, and a (large) lego block) and 7 color types (red, green, blue, magenta, yellow, cyan, and white). Notions of robot manipulation actions were included where an object was positioned to the left, right or on top of another object. Randomized scenes were sampled with the number of blocks ranging from $4-6$ with further variations in the color and the size attributes for each object. The object positions were sampled uniformly on the table within the reachable work space of the manipulator. The block orientations were randomized

in an angular range of 10 degrees in the (x, y)-plane along the robot base. A set of assembly instructions were collected to convey an assembly task for the robot to perform. The set of instructions conveyed a single physical interaction *"put the red block on the green block"* or an extended interaction involving a composition of interactions such as *"Put the red block to right of the small blue block, then put the lego block on top of red block"*. Further, the instruction corpus was augmented with inclusion of synonyms from a data base. For example, the phrase *"on top of"* in the above instruction could be replaced by phrases such as *"on"*, *"above"*, *"atop"*, etc.

Next, we discuss creation of the ground truth manipulation programs and sub-goal traces for the specified instructions. Each instruction was human annotated with a manipulation program $P_g = [\pi_1, \pi_2, ..\pi_T]$ where the constituent actions were grounded to the appropriate object instances in the environment. The robot's motion skill were implemented using a crane-like manipulator movement to grasp and re-position the grasped object at the target pose. The inferred manipulation program P_g was executed by the robot arm affecting the initial world state S_I towards the final world state S_G. The placement of objects on top, the left or the right of an object was realized by picking the target pose at the centre of mass of the base object. Typically, 1-3 manipulation actions were performed by the robot to complete a task in our data set. The intermediate sub-goals locations attained by the robot end-effector were recorded for evaluating the sub-goal predictions of the learned model. The procedure above resulted in a training data set of size 2500 with each data point consisting of the initial scene, final scene and a language instruction. In order to assess model accuracy, a test data set of size 2000 was created. The evaluation data-set consisted of novel scenes and instruction pairs that were unseen during training; assessing model generalization in novel settings.

B. Baseline Model

To evaluate the relative efficacy of our proposed approach, we construct a baseline which combines the various stages of our pipeline, and tries to directly output the sub-goals, which can be thought of as a neural version of our model, which does not explicitly argue about various concepts such as attribute values, and actions. Our baseline is also similar to the approach used by Paxton et al. [15] by adapting it into an object centric world without assuming grounding of visual concepts (colors) to objects in the scene and supervision of intermediate goals as assumed by the authors. We first describe the baseline for single step commands. We first compute the instruction embedding $emb(\Lambda)$, which is passed through a decoder to give a dense action embedding, represented by a_Λ. Since we are working in a neural framework, we do not have an action selector operation unlike in our case. We also compute the embedding of each object $emb(o_i)$. Another module takes $emb(\Lambda)$ and $emb(o_i)$, projects them in the same dimensional space, and computes the similarity. The similarity is computed for every object embedding, which is then used to compute an attention score, to get the final locations, as a linear combination of input object locations, which would be passed the action execution module. This is done separately (using two different neural networks) for the two arguments of the binary actions. Let the resulting locations be $l_{1\Lambda}$ and $l_{2\Lambda}$, respectively. Finally, we pass a_Λ, $l_{1\Lambda}$ and $l_{2\Lambda}$ through another neural (action execution) module which outputs the sub-goal g for Λ. It should be noted that baseline's action execution module has the same architecture as our action simulator, with the difference that it is passed a linear combination of object locations, since it does not have a notion of object selector, and a dense embedding for action instead of a one-hot representation. For multi-step commands, we give the baseline full knowledge of how the command should be split, and then we produce the sub-goals independently for each command.

C. Accuracy

We assess the model's ability to learning grounded and executable action representations. We evaluate the accuracy of our proposed model on 1 - 3 step tasks, and compare it with the Baseline. We report the predicted program P accuracy for our model. This metric called program accuracy captures the accuracy of predicting the type and sequence of DSL operators and concepts (attribute and action types) for each instruction-scene pair. We compute a $(0/1)$-loss and consider the predicted program to be correct only if it exactly matches the ground truth. Next, we compare the accuracies of predicting only the first argument object O_1 correctly and the second argument object O_2 correctly. For multi-step predictions, we assume a sequence of O_1's as correct only when O_1 at each step is correct. Next, we turn our attention to evaluate the accuracy of the overall plan G. Let predicted plan be $G = [g_1, g_2, .., g_{T_p}]$ and the correct plan be $\hat{G} = [\widehat{g_1}, \widehat{g_2}, .., \widehat{g_{T_c}}]$. Then we define a metric for the plan accuracy as,

$$\frac{\sum_{t=1}^{\min(T_c,T_p)} \mathcal{M}(g_t, \widehat{g_t})}{\max(T_c, T_p)}$$

where $\mathcal{M}$ calculates an IOU metric between $g_t = (o_{1t}, b_t)$ and $\widehat{g_t} = (\widehat{o_{1t}}, \widehat{b_t})$ as $\mathcal{M}(g_t, \hat{g_t}) = \mathbb{1}_{o_{1t}=\widehat{o_{1t}}} * IOU(b_t, \widehat{b_t})$ For the goals $g = (o_{1t}, p_t)$ in the robot's world space, we calculate a 3D IOU by calculating volumetric intersection between predicted and ground positions of object by assuming a 3D cube with size as dimension of the given block, centered around position p_t. Table III reports the accuracy results. Both test (around 1700 samples) and train set (around 1700 samples) contain 1 - 2 step commands. Further, test and train do not have any 1-step command in common. Both models achieve a high accuracy during training but the proposed model shows significant improvement over the baseline for the test set.

D. Generalization

Next, we assess model generalization to instructions that refer to new object attribute pairs (e.g., *small* and *green* block) that are distinct from those encountered during training. Table IV reports the accuracies for the proposed and the baseline models. The results indicate a stronger generalization for the proposed approach compared to the baseline model.

TABLE III: Accuracy Comparison for the Proposed Model and the Baseline on the Test and Train Sets

	Model	Prog (P)	O_1	O_2	IOU-2D	IOU-3D
Train	Baseline	-	95.7	98.0	54.0	33.3
	Ours	99.1	98.0	97.8	59.8	29.1
Test	Baseline	-	90.2	93.0	30.3	14.7
	Ours	97.3	95.8	96.2	52.6	25.5

TABLE IV: Generalization On Instructions with Novel Object Attributes

Model	Prog (P)	Arg (O_1)	Arg (O_2)	IOU-2D	IOU-3D
Baseline	-	87.5	89.1	29.0	16.6
Ours	98.5	96.3	96.3	51.6	25.2

The result illustrates the model's ability to reason about language instructions with novel object attribute references in the context of the scene. Such visual-linguistic reasoning operates on a single scene. Next, we assess generalization over multiple time steps. We explicitly evaluate the model on a test cases where an instruction involves multi-step actions that exceed those seen in training. In effect, we study model generalization in a setting where the model receives training data 1-2 step commands but needs to generalize to instructions requiring 3 actions to attain the intended goal. Table V presents the results for generalization on multi-step commands. The models were only trained on 1-2 step commands, and are tested on 3 step commands. The proposed model shows improved generalization compared to the baseline in the IOU metrics. Our model has lower accuracy on O_1 and O_2 primarily due to parser's inability to always split 3 step commands as can be seen from low accuracy of program P. The baseline model, one the other hand is provided the instructions split manually, and so does not has to split the command.

Table VI presents evaluation on task instructions that involve multi-step reasoning over attributes in the setting where the model is trained only on a single attribute. For example, the training set instructions only involve singe attributes such as 'red' and 'lego' with two different objects but during testing the model must compositional reason over both attributes to understand and follow the instruction. As before, the ability to combine grounded concepts with reasoning allows the proposed model to generalize better in this setting achieving significantly higher generalization accuracy compared to the baseline.

TABLE V: Generalization over Three-step Instructions with One/Two-step Instructions in Training

Model	Prog (P)	Arg (O_1)	Arg (O_2)	IOU-2D	IOU-3D
Baseline	-	76.0	80.9	13.3	6.6
Our	56.8	54.7	53.6	27.2	12.9

TABLE VI: Generalization to Novel Multi-Object Attribute Instructions

Model	Prog (P)	Arg (O_1)	Arg (O_2)	IOU-2D	IOU-3D
Baseline	-	55.0	32.4	3.2	1.5
Ours	98.0	75.7	89.5	27.3	9.3

In essence, the above results demonstrate stronger generalization to novel scenes with unseen object attribute pairs as well as action sequences that extend beyond those encountered during training. The neuro-symbolic approach involves learning of data-driven learning of concept representations that are amenable to rich compositional reasoning. The neuro-symbolic approach transfers better to novel settings compared to the direct neural-only approach that does not make use of the modular structure during training.

VI. Conclusions and Future Directions

We considered the problem of learning a task planning model for language-guided robot manipulation. We present a neuro-symbolic architecture that learns grounded and executable programs via visual-linguistic reasoning for instruction understanding over a given scene as well as grounded actions that transform the world state towards the intended goal.Our central contribution is learning a dense and disentangled representation for robot actions that on one hand can predict effects on the world scene, and on the other, are amenable to symbolic reasoning. We show how the neuro-symbolic model can be trained end-to-end and demonstrate a strong generalization to novel scenes and instructions compared to a neural-only baseline. We note that our work takes a first step towards learning grounded representation for robot manipulation tasks that are naturally amenable to symbolic reasoning.

While it is true that the present experiments are constrained to be in a relatively small action space and deal with a simplistic block world environment, in the future, we plan to advance our neuro-symbolic model towards learning rich program representation required for long horizon tasks in realistic environments. Specifically, the following directions emerge. First, the symbolic aspect of the model provides a path towards learning more complex programs incorporating a notion of repetition inherent in instructions such as *"stack all the red blocks on the blue tray"* where we would like the robot to infer the repetitive grasping-releasing behaviour required to complete the task. Such inductive learning would require the language reasoning module to induce a more expansive space of programs with notions of repetition. Second, the present work demonstrates the ability to learn dense neural representations for actions, which can be expanded to include a larger set of actions such as pushing, pulling or sliding one or more objects that are likely to be required for realistic assemblies. Third, we observe that our approach may not be directly scalable to larger horizons, i.e. to large number of steps, as error in initial positions will have a cascade effect on the later sub goals, which may eventually prohibit learning long range plans. Tackling incremental error accumulation by interleaving execution and planning and simultaneously explaining any resulting program changes to the human partner (via language descriptions) are exciting avenues for future.

ACKNOWLEDGEMENTS

We thank IIT Delhi HPC facility for computational resources. Parag Singla is supported by the DARPA Explainable Artificial Intelligence (XAI) Program with number N66001-17-2-4032, IBM SUR awards, and Visvesvaraya Young Faculty Fellowship by Govt. of India. Rohan Paul is supported by the Pankaj Gupta Young Faculty Fellowship. 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] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, 2016.

[2] Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6652–6659. IEEE, 2014.

[3] Yordan Hristov, Daniel Angelov, Michael Burke, Alex Lascarides, and Subramanian Ramamoorthy. Disentangled relational representations for explaining and learning from demonstration. In *Conference on Robot Learning*, pages 870–884. PMLR, 2020.

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

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[6] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[7] Ross A Knepper, Todd Layton, John Romanishin, and Daniela Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International conference on robotics and automation*, pages 855–862. IEEE, 2013.

[8] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[9] Sumith Kulal, Jiayuan Mao, Alex Aiken, and Jiajun Wu. Hierarchical motion understanding via motion programs. *arXiv preprint arXiv:2104.11216*, 2021.

[10] Miguel Lázaro-Gredilla, Dianhuan Lin, J Swaroop Guntupalli, and Dileep George. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Science Robotics*, 4(26), 2019.

[11] Yikai Li, Jiayuan Mao, Xiuming Zhang, Bill Freeman, Josh Tenenbaum, Noah Snavely, and Jiajun Wu. Multi-plane program induction with 3d box priors. *arXiv preprint arXiv:2011.10007*, 2020.

[12] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

[13] Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. In *Robotics: Science and Systems Foundation*, 2016.

[14] Rohan Paul, Andrei Barbu, Sue Felshin, Boris Katz, and Nicholas Roy. Temporal grounding graphs for language understanding with accrued visual-linguistic context. *arXiv preprint arXiv:1811.06966*, 2018.

[15] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Fox. Prospection: Interpretable plans from language by predicting the future. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6942–6948. IEEE, 2019.

[16] Subhro Roy, Michael Noseworthy, Rohan Paul, Daehyung Park, and Nicholas Roy. Leveraging past references for robust language grounding. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 430–440, 2019.

[17] Ankit Shah and Julie Shah. Interactive robot training for temporal tasks. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 603–605, 2020.

[18] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[19] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[20] Victoria Xia, Zi Wang, Kelsey Allen, Tom Silver, and Leslie Pack Kaelbling. Learning sparse relational transition models. In *International Conference on Learning Representations*, 2018.

[21] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*, 2019.

[22] Luke S Zettlemoyer, Hanna Pasula, and Leslie Pack Kaelbling. Learning planning rules in noisy stochastic worlds. In *AAAI*, pages 911–918, 2005.

[23] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *arXiv preprint arXiv:2012.07277*, 2020.

ANNEXURE I: DEMONSTRATION ON A SIMULATED ROBOT

We demonstrate the utility of the learned model on a simulated 7-dof Franka Emika robot manipulator in a table top setting. The robot is provided language instructions and uses the model to predict sub-goals. The sequence of predicted object poses is used to generate low-level motions to complete the instructed task. In our experiments, the program and sub-goal inference took less than 0.01 seconds on a CPU machine and the manipulator takes ≈ 15 seconds to execute one action.

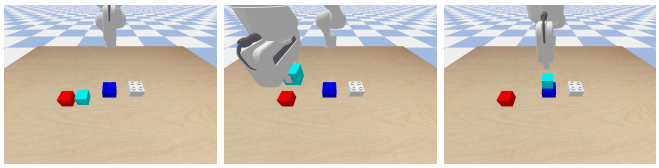

Fig. 2: A robot manipulator correctly performing a single step instruction *"put cyan block on blue block"* in a simulated table top environment.

Figure 2 illustrates the robot successfully performing a single step instruction, *"put cyan block on the blue block"* using the sub-goal predicted by the inferred program.

Figure 3 illustrates the robot manipulator performing the instruction *"put yellow block to the left of white block and put magenta small block on yellow block"*. the model reasons about the input instruction in the context of the environment correctly positions the yellow block on the left side of the intended white block. Subsequently, the robot grasps the magenta block and correctly positions the block on the yellow block that was previously manipulated.

Figure 4 shows the execution for the instruction *"put red block on blue block and put magenta block on red block"*. The model correctly predicts a program consisting of a composition of two sequential manipulation actions. The robot correctly identifies the intended objects for manipulation, then grasps and re-positions them to form the intended assembly. In the final step, the robot correctly placed the magenta-colored block on top of the red-colored block, however, the inferred placement was unstable and the block fell from the assembly. Note that the current model infers the task plan which is then handed over to the motion planner. Such a staged approach possesses the inherent limitation of not being able to recover from execution failures. The possibilities of a reactive approach by interleaving planning and execution are to be explored in future work and is likely to benefit error-recovery.

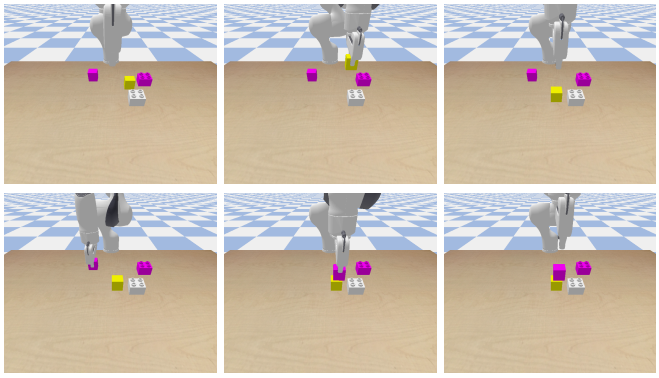

Fig. 3: Robot manipulator performing a two-step instruction *"put yellow block to the left of white block and put magenta small block on yellow block"*

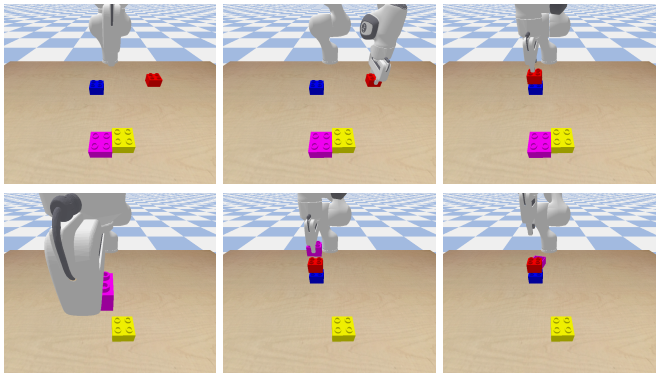

Fig. 4: Robot manipulator performing a two-step instruction *"put red block on blue block and put magenta block on red block"*. The robot correctly inferred the sequence of symbolic actions but failed during execution error due to the unstable placement of the second block.

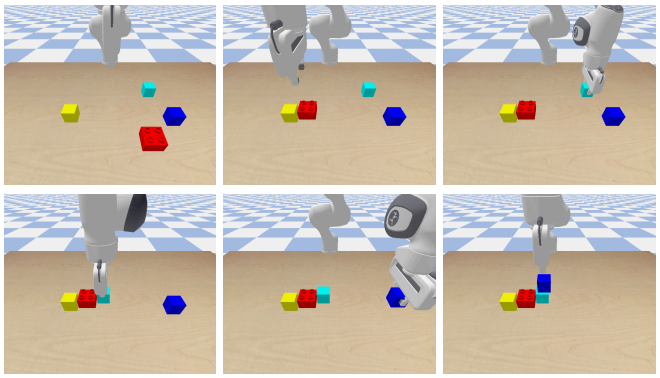

Fig. 5: Robot manipulator performing a three-step instruction *"put red block to the left of yellow block and put cyan block to the left of red block and then put red block on cyan block"*. The robot correctly inferred the sequence of actions.

In Figure 5, the robot is provided with the instruction, *"put red block to the left of yellow block and put cyan block to the left of red block and then put red block on cyan block"* which requires three successive manipulations. The model used in this experiment is trained only on two step instructions. However, at inference time, the model can generalize to a longer instruction by leveraging the compositionality inherent in our program representation.