Neuro-symbolic Architectures for Strongly Generalizable Language-guided Robot Manipulation

Thesis submitted by

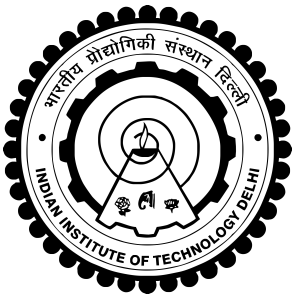
Himanshu Gaurav Singh
2019CS10358

under the guidance of

Prof. Rohan Paul, Prof. Parag Singla
Indian Institute of Technology Delhi

in partial fulfilment of the requirements
for the award of the degree of

Bachelor of Technology



Department Of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

July 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **Neuro-symbolic Architectures for Strongly Generalizable Language-guided Robot Manipulation**, submitted by **Himanshu Gaurav Singh (2019CS10358)**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Rohan Paul
Assistant Professor
Computer Science and Engineering
IIT-Delhi

Prof. Parag Singla
Professor
Computer Science and Engineering
IIT-Delhi

Place: New Delhi

Date: 12rd July 2023

ACKNOWLEDGEMENTS

I express my heartfelt gratitude to Prof. Rohan Paul and Prof. Parag Singla for the continuous support in this research endeavour. Both of them were a constant source of inspiration, encouragement, and knowledge. Lots of lengthy discussions on research ideas and possible directions have taught me not only the subject matter but also the fundamental art of scientific pursuit. I thank them for maintaining a very understanding and enjoyable research culture, and always motivating us to put in our best efforts, even working with us at odd hours whenever needed.

I would also like to thank fellow IIT Delhi students: Arnav Tuli, Namasivayam K and Vishal Bindal, for being the best teammates I could have asked for. I will fondly cherish our late-night sprints during paper submissions and stimulating discussions regarding minute details of our work. I will always remember the discipline, sincerity and amazing work ethic that these guys have instilled in me. I will miss our movie-nights and lunch parties after every project milestone.

Special thanks to my girlfriend Gurusha for being there for me all of the time I was busy with research work. I admire and feel gratified by her patience and affection towards me.

Last but not the least, I would like to thank my parents for constant support. They have supported each and every one of my career choices and encouraged me to carve my own path. My mother's enthusiasm and curiosity in daily-life has been a constant source of inspiration for me. I am grateful to them for cultivating in me curiosity, ambition, discipline and making me a good human being.

ABSTRACT

In this thesis, we present approaches to design and train neuro-symbolic architectures for the problem of *language-guided robot manipulation*. Given a natural language instruction and an input scene, the task is to compute a sequence of low-level actuations for the end-effectors of a robot enabling it to reach the desired final scene. We aim to build a system that generalizes to arbitrary combinatorial and inductive compositions of concepts in the scene and arbitrarily complex reasoning in the instruction.

Prior approaches to this task possess one or more of the following limitations: (i) Fail to generalize to instances that require deeper visuo-spatial reasoning than seen during training (ii) Require dense sensorimotor supervision [**?**] (iii) Require extensive amount of data to acquire new concepts (iv) Fail to generalize to combinatorially larger instances than present in the training distribution. In contrast, human intelligence seamlessly transfers to unseen situations and enables acquisition of novel concepts from very few examples. Central to this is the ability to arbitrarily compose acquired concept representations for understanding more complex visuo-spatial concepts and long-horizon reasoning. This provides the motivation to the core idea of this thesis: *augmenting neural-network based function approximators with programmatic compositionality to achieve strong out-of-distribution generalization*.

In the first part of this thesis, we introduce a neuro-symbolic approach for jointly learning disentangled representations for pre-designed visuo-spatial concepts and learning to parse natural language into *manipulation programs*: symbolic programs on top of concept symbols that explain how the scene is likely to affected by the input instruction. The program space is defined using a Domain Specific Language (DSL): a type grammar defined on symbolic operators and primitve concepts whose semantics are learned using distant, sparse supervision. Our model demonstrates end-to-end learning and strong generalization to novel scene compositions and longer instructions.

In the second and final part of this thesis, we propose a lifelong learning framework that continually acquires programmatic representations of concepts leading to an ever-growing library of *concept programs*. These *concept programs* belong to a hierarchical program space defined on top of previously acquired concepts and dense representations of a few primitive concepts, allowing composing concepts sequentially and repeating a concept multiple times. This enables us to learn inductively generalizable representation of complex visuo-spatial concepts from very-few human demonstrations, as demonstrated by our experiments. Given a human provided training curriculum, our system learns to build *staircases* of arbitrary sizes using a single human demonstration.

Contents

List of Tables

List of Figures

ABBREVIATIONS

IITD	Indian Institute of Technology, Delhi
RTFM	Read the Fine Manual
MVM	Matrix Vector Multiplication
DNN	Deep Neural Network

NOTATION

r	Radius, m
α	Angle of thesis in degrees
β	Flight path in degrees

Chapter 1

INTRODUCTION

This document provides a simple template of how the provided `iitddiss.cls` LaTeX class is to be used. Also provided are several useful tips to do various things that might be of use when you write your thesis.

To compile your sources run the following from the command line:

```
% pdflatex thesis.tex
% bibtex thesis
% pdflatex thesis.tex
% pdflatex thesis.tex
```

Modify this suitably for your sources.

To generate PDF's with the links from the `hyperref` package use the following command:

```
% dvipdfm -o thesis.pdf thesis.dvi
```

1.1 Package Options

Use this thesis as a basic template to format your thesis. The `iitddiss` class can be used by simply using something like this:

```
\documentclass[PhD]{iitddiss}
```

To change the title page for different degrees just change the option from `PhD` to one of `MS`, `MTech` or `BTech`. The dual degree pages are not supported yet but should be quite easy to add. The title page formatting really depends on how large or small your thesis title is. Consequently it might require some hand tuning. Edit your version of `iitddiss.cls` suitably to do this. I recommend that this be done once your title is final.

To write a synopsis simply use the `synopsis.tex` file as a simple template. The synopsis option turns this on and can be used as shown below.

```
\documentclass[PhD,synopsis]{iitddiss}
```

Once again the title page may require some small amount of fine tuning. This is again easily done by editing the class file.

This sample file uses the `hyperref` package that makes all labels and references clickable in both the generated DVI and PDF files. These are very useful when reading the document online and do not affect the output when the files are printed.

1.2 Example Figures and tables

Fig. 1.1 shows a simple figure for illustration along with a long caption. The formatting of the caption text is automatically single spaced and indented. Table 1.1 shows a sample table with the caption placed correctly. The caption for this should always be placed before the table as shown in the example.

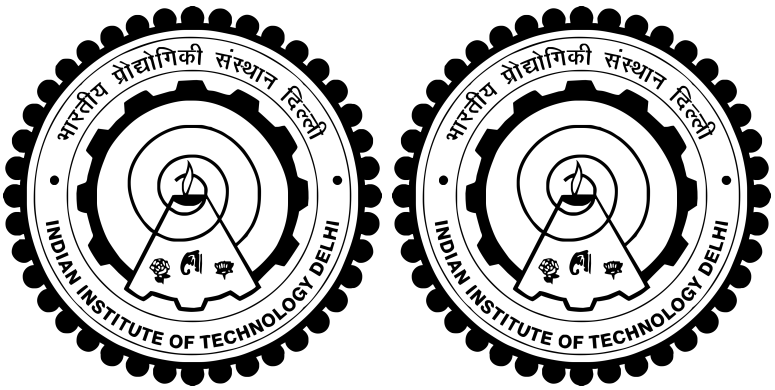


Figure 1.1: Two IITD logos in a row. This is also an illustration of a very long figure caption that wraps around two two lines. Notice that the caption is single-spaced.

Table 1.1: A sample table with a table caption placed appropriately. This caption is also very long and is single-spaced. Also notice how the text is aligned.

x	x^2
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64

1.3 Bibliography with BIBTEX

I strongly recommend that you use BIBTEX to automatically generate your bibliography. It makes managing your references much easier. It is an excellent way to organize your references and reuse them. You can use one set of entries for your references and cite them in your thesis, papers and reports. If you haven't used it anytime before please invest some time learning how to use it.

I've included a simple example BIBTEX file along in this directory called `refs.bib`. The `iitddiss.cls` class package which is used in this thesis and for the synopsis uses the `natbib` package to format the references along with a customized bibliography style provided as the `iitd.bst` file in the directory containing `thesis.tex`. Documentation for the `natbib` package should be available in your distribution of LATEX. Basically, to cite the author along with the author name and year use `\cite{key}` where `key` is the citation key for your bibliography entry. You can also use `\citet{key}` to get the same effect. To make the citation without the author name in the main text but inside the parenthesis use `\citep{key}`. The following paragraph shows how citations can be used in text effectively.

More information on BIBTEX is available in the book by [**?**]. There are many references [**?** **?**] that explain how to use BIBTEX. Read the `natbib` package documentation for more details on how to cite things differently.

Here are other references for example. **?**] presents a Python based visualization system called MayaVi in a conference paper. **?**] illustrates a journal article with multiple authors. Python [**?**] is a programming language and is cited here to show how to cite something that is best identified with a URL.

1.4 Other useful LATEX packages

The following packages might be useful when writing your thesis.

- It is very useful to include line numbers in your document. That way, it is very easy for people to suggest corrections to your text. I recommend the use of the `lineno` package for this purpose. This is not a standard package but can be obtained on the internet. The directory containing this file should contain a lineno directory that includes the package along with documentation for it.
- The `listings` package should be available with your distribution of LATEX. This package is very useful when one needs to list source code or pseudo-code.
- For special figure captions the `ccaption` package may be useful. This is specially useful if one has a figure that spans more than two pages and you need to use the same figure number.

- The notation page can be entered manually or automatically generated using the `nomencl` package.

More details on how to use these specific packages are available along with the documentation of the respective packages.

Chapter 2

Learning Neuro-symbolic Programs for Language-guided Robot Manipulation

2.1 Introduction

As robots enter human-centric environments, they must learn to act and achieve the intended goal based on natural language instructions from a human partner. We address the problem of learning to translate high level language instructions into executable programs grounded in the robot's state and action space. We focus on multi-step manipulation tasks that involve object interactions such as stacking and assembling objects referred to by their attributes and spatial relations. Figure 2.1 provides an example where the robot is instructed to *"put the block which is behind the green dice to the right of the red cube"*. We assume the presence of natural supervision from a human teacher in the form of input and output scenes, along with linguistic description for a high-level manipulation task. Our goal is to learn representations for visual and action concepts that can be composed to achieve the task. The learning problem is hard since (1) object attributes and actions have to be parsed from the underlying sentence (2) object references need to be grounded given the image and (3) the effect of executing the specified actions has to be deciphered in the image space, requiring complex natural language as well as image level reasoning. Further, the model needs to be trained end-to-end, learning representations for concepts via distant supervision.

Prior efforts can be broadly categorized as (i) Traditional methods which learn a mapping between phrases in the natural language to symbols representing robot state and actions in a pre-annotated dataset [? ? ? ? ? ? ?]; they lack the flexibility to learn the semantics of concepts and actions on their own, an important aspect required for generalizability (ii) Approaches that model an instruction as a sequence of action labels to be executed, without any deeper semantics, and requiring intermediate supervision for sub-goals, which may not be always be available [? ? ? ? ? ? ? ?] (iii) Recent end-to-end learning approaches [? ? ? ? ? ? ? ?]; they have limited reasoning capability both at the level of instruction parsing, and their ability to learn varied action semantics.

In response, we introduce a neuro-symbolic approach for jointly learning representations for concepts that can be composed to form *manipulation programs*: symbolic programs that explain how the world scene is likely to affected by the input instruction. Our approach makes use of a Domain Specific Language (DSL) which specifies various concepts whose

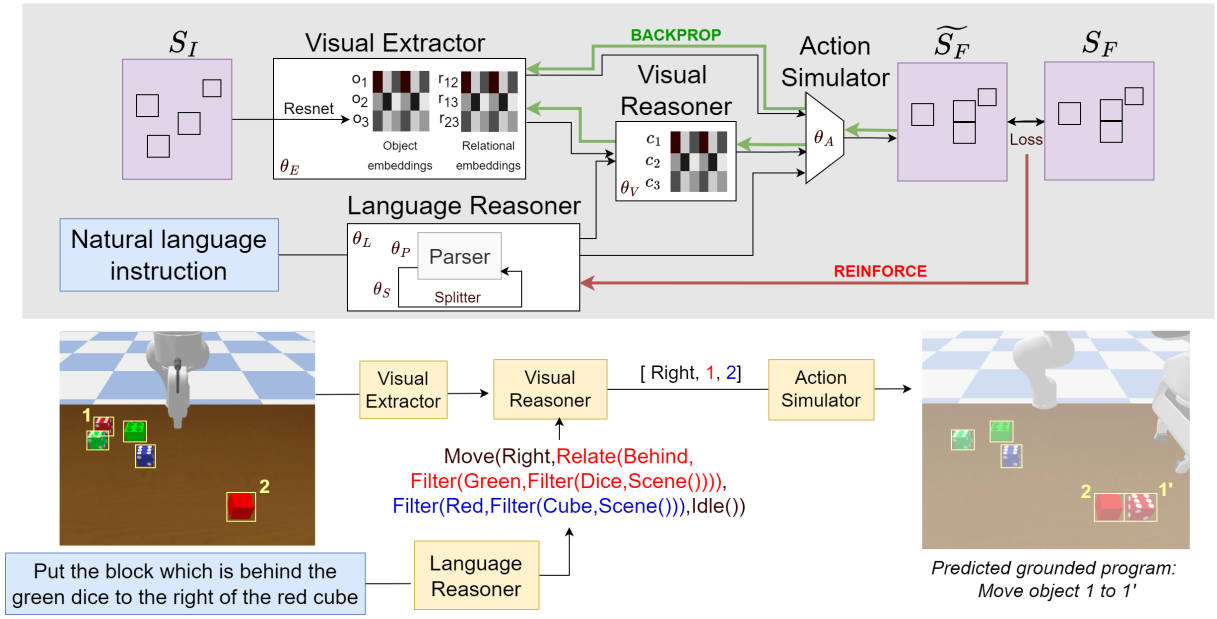


Figure 2.1: **Model architecture.** The *Visual Extractor* forms dense object representations from the scene image using pre-trained object detector and feature extractor. The *Language Reasoner* auto-regressively induces a symbolic program from the instruction that represents rich symbolic reasoning over spatial and action constructs inherent in the instruction. The *Visual Reasoner* determines which objects are affected by actions in the plan using symbolic and spatial reasoning. The *Action Simulator* predicts final location of the moved object. The model is trained end-to-end with a loss on the bounding boxes, backpropagated to action and visual modules. REINFORCE is used to train [**?**] the language reasoner from which symbolic programs are sampled.

semantics are learned by the model. We build on the concept learning framework by [**?**] for grounding of concepts in a natural language instruction in the image. The *manipulation program* is grounded into the scene to get a sequence of actions that are fed to the low-level motion planner of the robot, which is assumed to be given.

The contributions of this work are: (i) A novel neuro-symbolic model that learns to perform complex object manipulation tasks requiring reasoning over scenes for a natural language instruction, given initial and final world scenes. (ii) A demonstration of how dense representations for robot manipulation actions can be acquired using only the initial and final world states (scenes) as supervision without the need for any intermediate supervision. (iii) An RL-based learning approach that facilitates parsing of the complex instruction into symbolic concepts, and allows for learning of disentangled actions in the robot's space. (iv) Evaluation in instruction following experiments with a simulated 7-DOF robot manipulator, demonstrating robust generalization to novel settings improving on the state-of-the-art.

2.2 Related Works

Grounding Instructions to Robot Control. Traditional approaches for grounding instructions assume a symbolic description of the environment state and robot actions. These

approaches learn to associate phrases in an instruction with symbols conveying their meaning from annotated datasets. For example, [? ? ?] map language to discrete motion constraints that can be provided to a motion planner for trajectory generation, [? ?] parse language to a logical parse that includes robot control actions and [? ?] use sequence to sequence modeling to ground language to LTL formulae capturing high-level task specifications. In contrast to such approaches, the proposed work doesn't assume a symbolic model of the world and learns spatial and visual concepts and action semantics from data.

Inferring Plans from Instructions. Another set of approaches focus on multi-stage tasks (e.g., cooking, assembly etc.) and propose learners that can infer action sequences by observing human task demonstrations. The work in [?] proposes a recurrent neural architecture that converts a natural language instruction to a sequence of intermediate goals for robot execution. Efforts such as [? ? ?] learn LTL task specifications from humans demonstrations and introduce a model for effectively searching in the space of programs. In another effort, [?], authors introduce a cognitively-inspired approach that infers a likely program from a symbolic generative grammar for a robot manipulator to form and re-arrange block patterns in a table top setting. Misra et al. [?] present a similar approach but additionally assess plan feasibility for inferred plan candidates using a symbolic planner in the training loop. Despite successful demonstrations, these approaches require dense intermediate supervision and treat robot actions in a purely symbolic manner without learning their deep grounded semantics. However, our method learns a latent space of composable symbolic programs which can be executed in the latent space of object representations. The composability of the symbolic programs enables incremental learning through a specified curriculum from only the initial and final state data, forgoing the need for dense intermediate supervision.

Learning Action Models. This work builds on the rich literature on acquiring action models for planning and captures *when* actions are applicable and *how* they affect the world. Efforts such as [?] and [?] learn initiation sets for actions implicitly learning to classify the robot's configuration space where an action can be initiated. [? ? ?] present neuro-symbolic approaches for learning a latent object-centric world model for tasks such as stacking, re-arrangement etc. In [?], authors introduce an end-to-end model that jointly predicts object affordances and object displacements specifying metric goals for robot re-arrangement and sorting tasks. Recent parallel work by Wang et al. [?] builds on [?] and represents the task as an executable program parsed from the instruction using a CCG parser, restricting the length and variation of the instructions it can handle. Even though these approaches successfully learn modular action models that are amenable to planning, the scope of reasoning is limited to one-hop reasoning over spatial relations and attributes. In contrast, this work focuses on interpreting task instructions that require compositional/hierarchical spatial reasoning over an extended planning horizon.

2.3 Problem Formulation

The robot perceives the world state comprising a set of rigid objects placed on a table via a depth sensor that outputs a depth image $S \in^{H \times W \times C}$, where H, W, C respectively denote the height, width and the number of channels (including depth) of the imaging sensor. The workspace is co-habited by a human partner who provides language instructions to the robot to perform assembly tasks. We assume a closed world setting, i.e. changes to the world state are caused only by the robot manipulator.

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 motions that result in the final world state S_F conforming to the human's intention. Following [**?** **?**], planning for 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, a semantic model for a manipulation task denoted as $ManipulationProgram(.)$ takes the initial scene S_I and the instruction Λ as input and determines a sequence of sub-goals as $(g_0, g_1, \ldots, g_n) = ManipulationProgram(S_I, \Lambda)$. Each sub-goal g_i aggregates the knowledge of the object, o_i, to be manipulated and its target Cartesian $SE(3)$ pose p_i. This is provided to the low-level motion planner to synthesize the end-effector trajectory for the robot to execute. The robot's motion planner, which is assumed to be given, includes grasping an object and synthesizing a collision-free trajectory for positioning it at a target pose.

Training data consists of demonstrations, corresponding to task instructions Λ, in the form of initial and final world states (S_I, S_F). Given data $D=\{S_I^i, S_F^i, \Lambda^i\}_{i=1}^M$, the model parameters are trained by optimizing a loss $\mathcal{L}(\tilde{S}_F^i, S_F^i)$ where $\tilde{S}_F^i = Simulate(ManipulationProgram(S_I^i, \Lambda;$ is the final state estimated by simulating the plan inferred by $ManipulationProgram(S_I^i, \Lambda)$ on initial state S_I^i. We seek strong generalization on novel scenes, instructions and plan lengths beyond those encountered during training, along with interpretability in sub-goals.

2.4 Technical Approach

We propose a neuro-symbolic architecture to solve the task planning problem described in section 2.3. Our architecture is inspired by [**?**] and is trained end-to-end with no intermediate supervision. We assume that the reasoning required to infer the sub-goals can be represented as a program determined by a domain specific language (DSL). Table 2.1 lists the keywords and operators in our DSL, along with the implementation details of the operators. We assume a lexer that identifies all the keywords that are referred to in the instruction Λ. We do not assume prior knowledge of the semantics of the DSL constructs, and they are learned purely from the data. Our architecture (ref. Figure 2.1) consists of the following key modules (a) Language Reasoner (LR) to parse the instruction into a

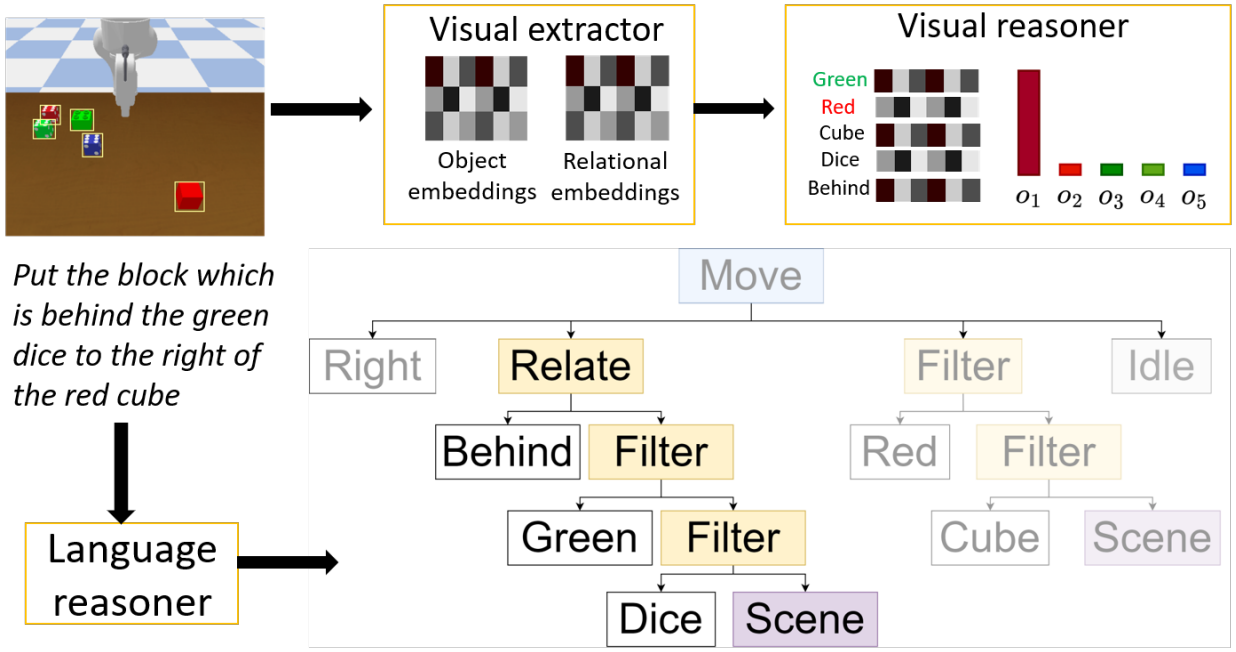


Figure 2.2

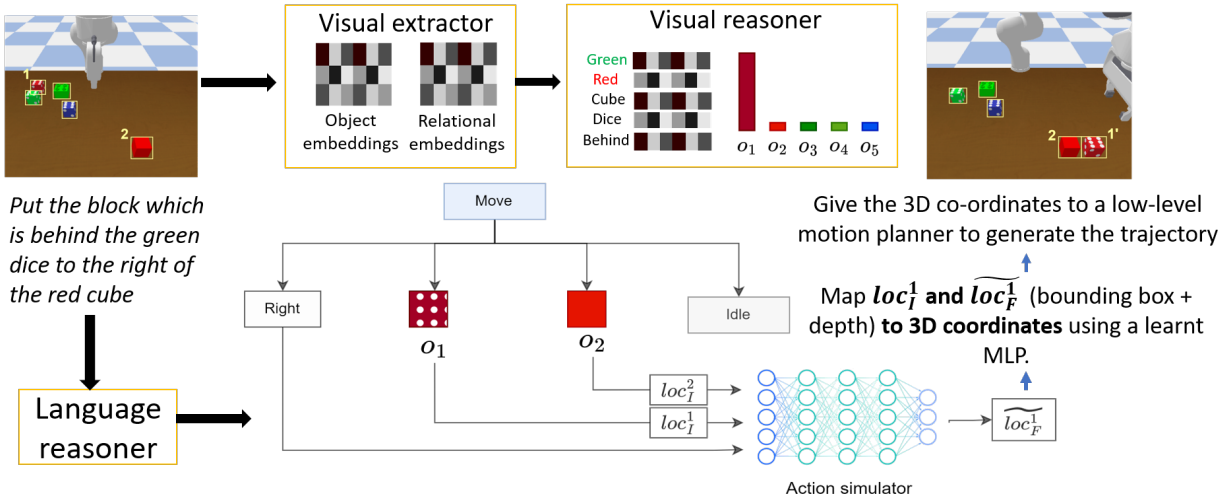


Figure 2.3

Figure 2.4: **Quasi-symbolic program execution.** We postulate a latent space of hierarchical, symbolic programs that performs explicit reasoning over action, spatial and visual concepts. a) The language reasoner infers a program belonging to this space, capturing the semantics of the language instruction. The program is executed over the latent object representations extracted from the initial scene to get a grounded program. b) The grounded program is used by the action simulator to compute the final location of the moved object. This is fed into a low-level motion planner for computing the low-level trajectory.

Keywords and their classes	
Object-level concepts	**Other concepts**
Color: {Red, Blue, Cyan,...}	**RelCpt**: {Left, Behind, Front,...}
Type: {Cube, Lego, Dice}	**ActCpt**: {MovRight, MovTop,...}
Operators: (Input → Output)	
Scene : None → ObjSet	Unique: ObjSet → Obj
Filter : (ObjSet, ObjCpt)→ ObjSet	Relate : (Obj, RelCpt) → ObjSet
Move : (ActCpt, World) → World	Idle : World → World
ObjectSet $\in R^{\mathrm{N}}$, N = Num objects, Object = one-hot ObjectSet	
World $= \{(b_i, d)\}_{i=1}^{\mathrm{N}} \in R^5$, bounding boxes and depth for all objects	

Table 2.1: Domain Specific Language.

hierarchical plan consisting of references to visual and action concepts and operators (b) Visual Extractor (VE) to obtain the initial bounding boxes from the input image (c) Visual Reasoner (VR) to ground the object related concepts present in the parsed instruction with respect to the input image (d) Action Simulator (AS) to learn the semantics of actions to be executed to produce a sequence of sub-goals. Figure 3.3 illustrates our approach.

2.4.1 Language Reasoner (LR)

The language reasoner (LR) deduces a hierarchical symbolic program that corresponds to the manipulation task implied by the human's utterance to the robot. The deduced program consists of symbolic reasoning constructs that operate on neural concepts grounded in the state space of the scene and the action space of the robot. Since a high-level task instruction may imply a sequence of actions, we adopt a hierarchical model where an LSTM [**?**] network first splits the instruction into a sequence of sub-instructions, each corresponding to one grounded action. The semantic parse of each sub-instruction is then inferred using a hierarchical *seq2tree* architecture similar to [**? ?**]. The sequence of programs thus generated are composed to produce the symbolic program for the original language instruction.

2.4.2 Visual Extractor (VE)

The visual extractor forms an object-centric view of the world by extracting object proposals in the form of bounding boxes for the objects (and class labels) using a pre-trained object detector [**?**]. Following [**?**], dense object representations are obtained by passing the bounding boxes (after non-maximal suppression) through a feature extractor as [**?**]. The data association between object proposals is estimated greedily based on cosine similarity between the dense object features in the initial/final scenes.

2.4.3 Visual Reasoner (VR)

The visual reasoner performs visuo-spatial, object-centric reasoning to compute a sequence of sub-goals grounded in the scene from the symbolic program. E.g., the instruction *"put the block which is behind the green dice"* requires the robot to perform reasoning to resolve the specific world object that conforms to the relations/attributes as being behind a dice with colour attribute of green.

The visual reasoner parameterizes object-level and relational concepts in the DSL (Table 2.1) with neural embeddings. Similar to [**?**], a differentiable framework is defined for the execution of operators such as *Filter, Unique, Relate, Idle.*
As in [**?**], intermediate results of execution are represented in a probabilistic manner. For e.g., a set of objects is represented as a vector of probabilities where the i-th element represents the probability of the i-th object being an element of the set. *Filter, Unique, Relate* etc. sequentially modify this vector to ultimately yield the referred object/set of objects.
This execution framework is used to execute the symbolic program on top of the visual features (extracted by the visual extractor) resulting in the grounding of the program, Π, on the world state.

2.4.4 Action Simulator (AS)

The action simulator learns the semantics of action concepts of the DSL. It takes as input, the action concept, the initial locations of the object to be manipulated and the reference object respectively and outputs the target location of the former. The outputted target location of the object being manipulated serves as a sub-goal for the low-level motion planner. We determine the location of an object by the corners $b = (x_1, y_1, x_2, y_2)$ of the enclosing bounding box and the depth of the object from the camera face.

Overall, the model can be summarized as follows:

- $\texttt{P} \leftarrow \texttt{LR}(\Lambda; \theta_P, \theta_S)$, where P is a symbolic program composed of DSL constructs.
- $\Pi \leftarrow \texttt{VR}(\texttt{P}, \texttt{VE}(S_I; \theta_E); \theta_V)$. The visual reasoner then grounds P and outputs Π, the grounded program. For example, in Figure 2.1, red and blue subprograms are grounded to object 1 and 2 respectively.
- $G \leftarrow \text{AS}(\Pi; \theta_A)$. The action simulator then takes Π and returns a sequence of sub-goals, $G = (g_0, g_1, ..., g_{n-1})$, for the motion planner.

Here, θ's are the parameters of the corresponding module.

2.4.5 Loss Function and Model Training

Given a single-step instruction Λ, the parser predicts a symbolic program `P`. The visual reasoner grounds `P` to predict a sequence of sub-goals. The action simulator computes the low-level action corresponding to each sub-goal. The sequence of low-level actions is executed on the initial scene S_I to get the predicted final locations of the objects $\{\widetilde{loc}_F^i\}_{i=1}^N$. Let $\{loc_F^i\}_{i=1}^N$ be the true locations in the gold final state, S_F. As mentioned above $loc = (b, d)$, where b is the corners of bounding box and d is the depth. The loss function $L_{act} \sum_i^N \|\widetilde{loc}_F^i - loc_F^i\| + \beta(1 - \text{IoU}(\widetilde{b}_F^i, b_F^i))$ is used to train the action simulator and the visual modules. Since there is no explicit supervision to the parser, we train the parser using the policy gradient algorithm REINFORCE with the reward set to $-L_{act}$. During initial training, an explicit expectation (subtracting the mean action loss as the baseline) is computed over all programs to inform the loss, ameliorating the variance issue arising in REINFORCE. The Language Reasoner is trained using REINFORCE and the other modules using backpropagation.

The following training curriculum is adopted: (i) training on single step commands with reasoning involving individual object features only, (ii) the trained action simulator module is frozen and additional training on single step commands involving reasoning over spatial relations between objects is carried out. (iii) the sentence splitter in the Language Reasoner module is fine-tuned on multiple-step instructions.

2.4.6 Scene Reconstruction

We additionally train a neural model to synthesize the scene corresponding to each sub-goal, given only the initial scene and predicted object locations. This enables us to visualise the scene modification without the need of execution by a robot manipulator along with providing interpretability to the model's latent program space. The reconstruction architecture is adapted from [**?**], where the scene graph is constructed with nodes having object features and bounding boxes. This graph is updated with predicted bounding boxes at each step, yielding generated scenes. Presence of initial and final scenes in our data means we can train in a supervised manner, unlike [**?**].

2.5 Evaluation and Results

2.5.1 Experimental Setup

Data collection. The dataset is collected in a PyBullet tabletop environment using a simulated Franka Emika Panda robot arm. The workspace consists of a tabletop with blocks of different shapes and colors. Each datapoint consists of an initial scene paired with

Table 2.2: Comparison between Proposed Model and NMN+ (BB: Bounding Boxes)

Model	Overall			Single-step		Double step		S
	IOU	IOU-M	Program (Action/Subj/Pred)	IOU	IOU-M	IOU	IOU-M	IOU
NMN+ (gold BB)	0.77	0.55	–/0.81/0.76	0.80	0.56	0.71	0.52	0.90
Ours (gold BB)	0.87	0.72	0.99/0.99/0.94	0.91	0.64	0.87	0.62	0.92
NMN+ (extracted BB)	0.69	0.32	–/0.81/0.55	0.78	0.49	0.58	0.22	0.79
Ours (extracted BB)	0.76	0.41	0.99/0.74/0.76	0.80	0.43	0.69	0.54	0.83

Table 2.3: Comparison b/w Proposed Model and CLIPort

Model	Overall		Simple		Complex	
	Id. Acc.	Pl. Acc.	Id. Acc.	Pl. Acc.	Id. Acc.	Pl. Acc.
CLIPort	0.46	0.39	0.55	0.47	0.32	0.26
CLIPort(5x)	0.76	0.61	0.90	0.74	0.55	0.39
Ours (gold BB)	0.94	0.93	1.0	1.0	0.83	0.81

a language instruction and the expected resulting final scene. For training, close to 5000 synthetic scenes and language instructions(with a 80:20 train-test split) are sampled with 3-5 blocks of varying colors and shapes and placed at randomized orientations on the table.

The dataset consists of both single-step(e.g. *"Put the red block on the green block"*) and double step commands(e.g. *"Put the red block which is behind the white dice to right of the blue cube, then put the lego block on top of red block"*). On the basis of the level of reasoning involved in the given instruction, we can classify the dataset into (i) *simple* that involve reasoning over individual object features only and *complex* that additionally involve reasoning over inter-object relationships.

2.5.2 Baselines, Metrics and Comparisons

The proposed model (i) performs visuo-linguistic reasoning in a symbolic latent space and (ii) assumes an object-centric state representation. We provide comparisons with baselines that forego these two assumptions.

NMN+, inspired from Neural Module Networks [**?**], assumes an object-centric state representation. However, in contrast to the proposed model, visuo-linguistic reasoning required for manipulation is performed using language-guided attention over object embeddings instead of using symbolic reasoning constructs. Figure 2.5 provides the detailed architecture. The entire architecture is neural and is trained end-to-end with the loss same as that of the proposed model.

The following metrics are used for comparing the proposed model with *NMN+*: (i)

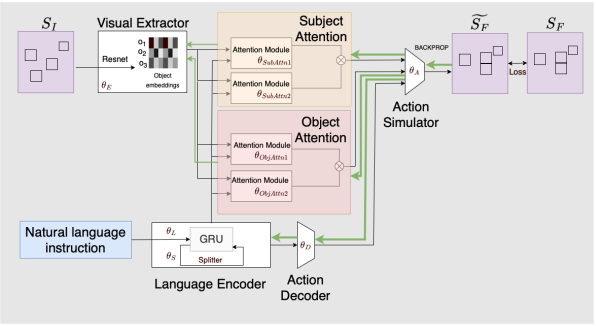
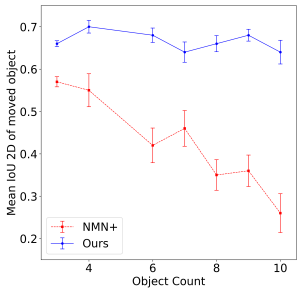


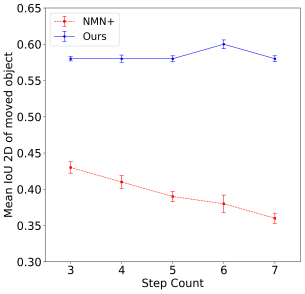
Figure 2.5: **Object-centric baseline NMN+.** For a fair comparison, our model and NMN+ share the same language encoder (with LSTM-based splitter) and the visual extractor. Attention blocks compute language-guided attention over object embeddings to get the *subject* and *predicate* for the manipulation action. This is fed into the action simulator along with the action embedding from the action decoder to get the predicted final location of the object.

Intersection over Union (IOU) of the predicted and groundtruth bounding box is calculated in the 2D image space assuming a static camera viewing the scene. Average IOU over all objects in the scene and mean IOU for objects moved during execution, termed IOU-M, are reported. (ii) *Program Accuracy*: The grounded program inferred for an (instruction, scene)-pair using the proposed model is compared with the manually annotated ground truth program. We separately report the grounding accuracy for the subject and predicate of our action (assumed binary) and the accuracy of the predicted action inferred from the instruction. Since, there is no explicit notion of grounded actions in the baseline, we do not report this metric for the baseline.



0.5

Figure 2.6: IoU vs # of objects



0.5

Figure 2.7: IoU vs varying # of steps

Figure 2.8: Performance in generalization settings

Table 2.2 reports the performance of our model and the baseline on the test set. We

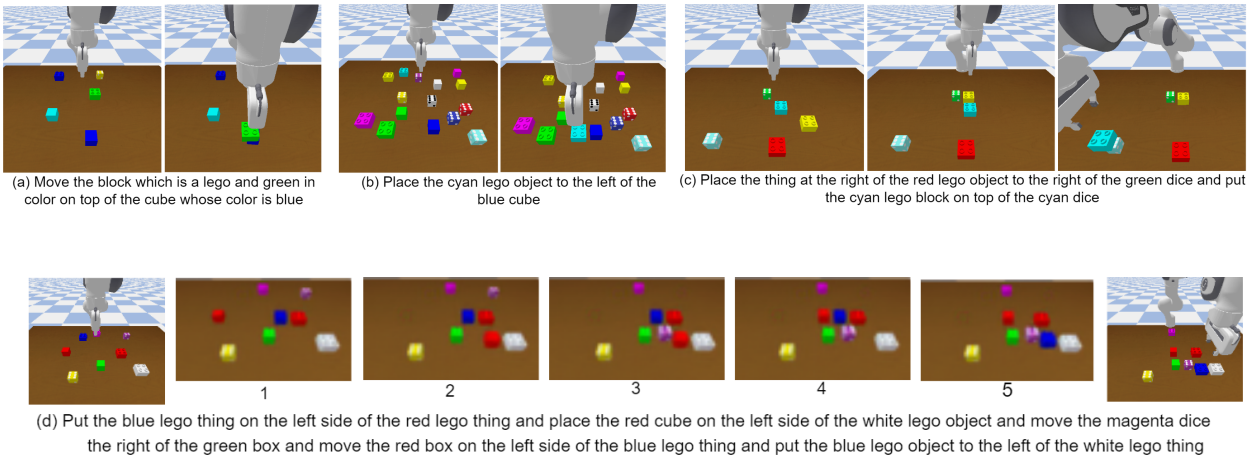


Figure 2.9: Execution of robot manipulator on (a) compound instructions, (b) scene with 15 objects, (c) double step instruction with relational attributes, (d) 5-step instruction. (d) also shows reconstruction of the predicted scene before each step of the simulation

report both numbers: using gold set of bounding boxes and using bounding boxes extracted [1] using the approach described in Section 2.4.2. Our model outperforms the baseline overall. For instructions with complex reasoning (resolution of binary spatial relations) involved, the proposed model outperforms the baseline by 33 points in the IOU-M metric. Disentangled representations of relations and concepts in *Visual Reasoning* module allow the proposed model to reason over complex instructions.

CLIPort [**?**] takes as input a dense, image-based state representation, in contrast to our object-centric state representation. It proposes a two-stream architecture that computes language-guided attention masks on the input scene for both pick and place locations. It lacks explicit symbolic constructs for reasoning about the scene. Instead, the reasoning required for manipulation is performed by leveraging the visuo-linguistic understanding of the large pre-trained model CLIP [**?**]. Table 2.3 compares *CLIPort* with our model. Since *CLIPort* performs only single-step pick-and-place operations, the numbers are calculated only on single-step examples. The following metrics are used (i) *Placement Accuracy*: fraction of examples in which the predicted place location lies within the groundtruth bounding box of the moved object (ii) *Identification Accuracy*: fraction of examples in which the predicted pick location lies within the groundtruth bounding box of the moved object. For the proposed model, the centre of the predicted bounding box location is used as the predicted place location.

We observe that CLIPort on our dataset shows low accuracy in both predicting the object to be moved as well as the place location. Further, its performance deteriorates in examples with relational concepts. We note that the performance and sample-efficiency of CLIPort critically depends upon spatial consistency of input images i.e. objects do not scale or distort depending on the viewing angle as in [**?**]. To provide additional leverage to

[1] Excluded about 10% no detection cases for both models.

the baseline, we train CLIPort on 5 times more data than the original, results of which are reported under the title CLIPort-5x. We observe an improvement in performance on the larger dataset, although still worse than that of the proposed model.

2.5.3 Combinatorial Generalization

To evaluate the approaches in an out-of-distribution generalization setting, we collect dataset consisting of scenes with more objects than seen during training, and with instructions that involve larger number of steps for carrying out the task.

Figure 2.6 illustrates the model generalizing combinatorially to scenes with more number of objects. The models were first trained on scenes having up to 5 objects only, and then tested on scenes having up to 10 objects.
The superior generalization demonstrated by the model can be attributed to reliance on an object-centric state representation and the ability to learn dense disentangled representations of spatial and action concepts, facilitating modular and structured reasoning that scales gracefully.

Figure 2.7 illustrates model generalization to longer horizon manipulation. We observe that the proposed model is able to perform multiple scene manipulation and reasoning steps with considerable accuracy up to 7 steps after being trained with instructions translating to plans up to 2 steps.
The performance of the object-centric baseline (NMN+) is worse and the model struggles to generalize to plans extending to longer horizons. We attribute this to the modular structure of our approach compared to the baseline.

2.5.4 Demonstration on a Simulated Robot

We demonstrate the learned model for interpreting instructions provided to a simulated 7-DOF Franka Emika manipulator in a table top setting. The robot is provided language instructions and uses the model to predict a program that once executed transitions the world state to the intended one. The 2-D bounding boxes predicted by the action simulator are translated to 3-D coordinates in the world space via a learned MLP using simulated data. The predicted positions are provided to a low-level motion planner for trajectory generation with crane grasping for picking/placing. Figure 2.9 shows execution by the robot manipulator on complex instructions, scenes having multiple objects, double step relational instructions, and multi-step instructions. We also visualise reconstruction of the moved objects before each step of the actual execution. The structural similarity index (SSIM) [**?**] for the reconstruction model is 0.935.

2.6 Conclusions

We present a neuro-symbolic architecture that learns grounded manipulation programs via visual-linguistic reasoning for instruction understanding over a given scene, to achieve a desired goal. Unlike previous work, we do not assume any sub-goal supervision, and demonstrate how our model can be trained end-to-end. Our experiments show strong generalization to novel scenes and instructions compared to a neural-only baseline. Directions for future work include dealing with richer instruction space including looping constructs, real-time recovery from errors caused by faulty execution, and working with real workspace data.

Bibliography

Chapter 3

Continual Learning of Inductively Generalizable Concepts for Language-guided Robot Manipulation

3.1 Introduction

Humans are life-long learners, learning novel concepts from very few demonstrations and able to generalize inductively. Building an embodied agent capable of learning novel concepts on-the-fly from a few human demonstrations is an important problem. One key factor that enables few-shot learning and generalization in humans is the ability to arbitrarily compose previously acquired concepts. After learning to make a *row* with blocks, it becomes easier to learn to construct *rectangular enclosures* aided by an easier higher-level mental description: *row to the east, row to the south, row to the west and row to the north* rather than an elaborate block by block description. Additionally, the former compositional representation generalizes to inductively larger concept instances in contrast to the latter. In this paper, we develop a framework to build an embodied agent that continually learns inductively generalizing representations of spatial concepts from demonstrations by a human partner.

Past work has looked at the problem of learning novel concepts from data [], including some work on learning from compositions [], but there is hardly any work which looks at learning a more general class of higher order concepts, except possibly in case of purely symbolic settings which have limited applicability []. While existing work [] has addressed the problem of extracting such concepts from natural language for performing robotic execution, their primary focus is on creating an association between language and symbols, as opposed to our setting, where we would also like to learn the semantics of novel concepts in the form of dense representations which was assumed to be given in the prior works. More recent work has used LLMs to parse a natural language instruction, in terms of code or programming statements, but overly rely on the ability of language models to learn the semantics, rather than learning it directly from human demonstrations [1]. We survey the various related works in Section 3.2.

We build on recent developments in the sub-field of neuro-symbolic models, such as those for VQA [**?**], and for language guided robotic execution of simple commands[**?**]. Central to our framework is an ever-growing library of *concept programs* each of which may use previously defined concepts in its specification. The library is built on top of assumed dense

[1] we compare with some of these approaches in our experiments.

representations of visual and spatial primitives. Whenever a new concept is encountered, we launch a search in the program space, to express the new concept by breaking it down in terms of already known concepts, via the use of symbols defined in a Domain Specific Language (DSL). As new concepts are learned, they are added to the DSL, resulting in an open world semantics. Basic concepts are assumed to have a neural representation, which is learned in the style of K et al. [**?**], from human demonstrations, in a curricular fashion. Our approach can be seen as a form of *continual learning*, where we acquire knowledge of ever growing repository of novel concepts, expressed in terms of already known concepts. Once a higher-order concept has been learned, its semantics is stored in the repository, does not be learned again, and can be simply executed at inference time.

Our training pipeline has the following broad steps. We first pass the natural language instruction through an LLM to get a sketch of the program to be executed. We note that this is only a sketch, and further needs to be broken down in terms of already known concepts which have previously been learned. Once the program sketch is provided to the agent, it has following actions to perform (1) It looks for any concepts whose semantics are already learned either in terms of a neural module (referred to as *basic concepts* or in terms of already known concepts. Such concepts can directly be executed. (2) It looks for novel concept symbols which have not been encountered hitherto – their semantics needs to be learned by doing a program search over already known concepts, guided by data (3) The program search is guided by having a prior for (a) using already learned concepts (b) preferring shorter programs. (4) Loss is expressed in terms of the IoU between the final scene obtained by executing the learned program and the actual demonstration. Dense representations are learned using the standard back-propagation algorithm. Figure 3.3 shows an example.

We create a new benchmark dataset to test the efficacy of our approach compared to existing baselines. We compare with purely neural baselines, as well as recently proposed models which use LLMs to directly output a plan [**?**]. Our experiments demonstrate that our approach has significantly better generalization capabilities. In many cases, none of the existing approaches can achieve a realization of concepts to be learned, whereas our approach is able to learn them in a seamless manner.

3.2 Related Work

Our work builds on literature concerned with enabling robots to follow instructions specified in natural language [**?**]. The task of *grounding* natural language instruction assumes a set of symbols specifying the robot's state and action space and infers a likely correspondence between the input utterance and symbols [**?**]. Recent efforts have shown promise in learning to associate language with spatial relations [**?**], reward functions [**?**], manipulation constraints [**?**] as well as hierarchical abstractions such as a entity collections [**?** **?**].

These works *assume* that the agent is provided a semantic representation for concepts (or has acquired them from another learning process) and focus only inferring the likely association between language and symbols. Our work instead aims to not just interpret language instruction but simultaneously learn a dense representation of higher-order concepts in a data-driven end-to-end framework.

Our work is informed by recent emergence of large language models (LLMs) trained on internet-scale data [**?**]. Several efforts leverage the generative capabilities of LLMs to provide procedural task decomposition using environment state and robot execution feedback as prompts [**?** **?** **?**]. Alternate efforts use the code generation capabilities of LLMs to translate a high-level instruction to executable robot control code either directly [**?**] or via iterative prompting [**?**]. Whereas earlier works use LLMs to directly translate a high-level task into smaller procedures or control code, our work uses LLMs to provide *candidate sketch* of how the goal can be constructed by the robot. The induced plan sketches provides cues for when and which concepts need to be learned. The cues guide the local search in concept learning which, if otherwise performed naively, is intractable in a large concept space. Further, our work leverages the large generative capacity of LLMs to provide a cue when a concept may need to be learned to initiate the learning process. This approach is in contrast to prior works that trigger new concept learning by collecting new visual-linguistic observations [**?**] or initiating dialogue [**?**] when predictive uncertainty is estimated to be high.

Our work is closely related to the general task of learning higher-order concepts has been more widely studied in the AI community. The work of [**?**] infer a representation for hand-drawn digit images in the form of probabilistic programs that show strong generalisation from few shot data. The work of [**?**] infers programs describing generation of pictures with multiple geometric figures using a contextually guided search in program space. The seminal work of [**?**]infers programs to describe reasoning involved in VQA tasks. The programs are expressed using a symbols from a domain specific language grounded over a dense representation of objects and relational concepts trained that generalize well from few examples. The representations are neuro-symbolic and trained end-to-end. Efforts such as [**?**] and [**?**] extend on this formulation and additionally learn action models in terms of manipulation on a world scene from pre-post state data from human demonstrations. Other works learn symbolic policies [**?**], LTL formulae [**?**] or semantic sub-goals [**?**] by efficiently searching for compact and representative explanation permitted in the assumed logical theory. Whereas prior efforts demonstrated successful learning of spatial and action concepts, our work, addresses the problem of learning higher-order concepts with a notion of repetition/recursion as well as acquire them in an online continual manner.

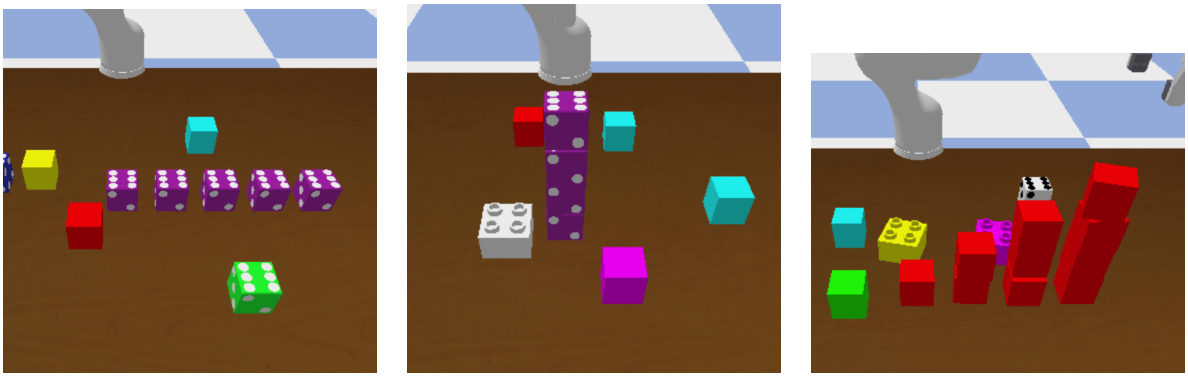


Figure 3.1: Concept instances eliciting inductive generalization. From left to right, (a) Row of 4 magenta blocks (b) Tower of 4 magenta blocks (c) Staircase of size 4 with red cubes

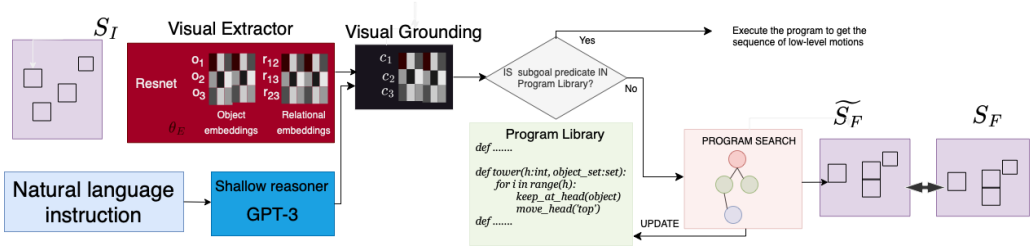


Figure 3.2: The language instruction is decomposed into a sequence of high-level goal predicates using an LLM as a *shallow reasoner* module. The goal predicates are grounded into the scene using pre-trained dense visual representations. The *program library* maintains programmatic representations of the spatial grounding of concepts acquired over time. If a novel concept is encountered, search in program space defined by the DSL 3.2 and guided by the final scene leads to acquisition of the new concept, updated in the program library.

3.3 Technical Approach

3.3.1 Problem Setup

The robot perceives the world state comprising a set of rigid objects placed on a table via a depth sensor that outputs a depth image S $\in^{\times H \times W \times C}$, where H, W, C respectively denote the height, width and the number of channels of the imaging sensor. The workspace is cohabited by a human partner who provides a language instruction Λ to the robot to perform an assembly task.The low-level action space of the robot is denoted by $\mathcal{A} \in^6$. We assume a closed world setting, i.e. changes to the world state are caused only by the agent. Given Λ and initial scene S^I, the agent is supposed to perform a sequence of low-level motions $\{g_i\}$ conforming to the instruction.

The dataset $\mathcal{D}$ contains triplets $\{S_i^I, \Lambda, S_i^F\}$, each comprising an initial scene, natural language instruction and the resultant final scene. Our learning task is to learn $\mathcal{F}$ such that $\mathcal{F}(S^I, \Lambda) = \{g_0, g_i \ldots, g_n\}$, where $S^F = Simulate(\{g_i\}, S^I)$. We seek strong generalization to inductively larger instances than seen during training, along with interpretability in concept representation.

End-to-end neural models for learning $\mathcal{F}$ struggle to generalize beyond the training distribution. Ours is a factored, neuro-symbolic approach in which we first decompose the instruction into high-level subgoals. Then, the semantics of novel, unseen concepts in the subgoals are learnt using program search guided by human demonstrations. Programmatic representation of concepts leads to strong inductive generalization. Additionally, maintaining an ever-growing library of *concept programs* enables lifelong learning. This architecture is summarized in Figure 3.3 and is detailed next.

3.3.2 Learning inductive concepts

We use an LLM to extract language semantics into a sequence of programmatic expressions, each describing a high-level subgoal. We hypothesize that novel concepts, in the form of unseen subgoal predicates, belong to a symbolic program space. A generic DSL (Table 3.2), parameterized by primitive spatial relations, defines this representation space. The rollout of the *concept program* provides a sequence of low-level motions to the robot. When an unseen concept is encountered, a symbolic search in the program space, guided by final scene, is performed to infer the correct program.

LLM for shallow, language-based reasoning Our first step is to distill the semantics of the natural language instruction into a sequence of symbolic sub-programs. For example, the instruction *Make a tower of green blocks to the right of the blue block* gets decomposed to `assign_head(Unique(Filter(Blue,Scene()))) ; move_head('right') ; tower(Filter(Green,Scene()))`. For this subtask, we leverage the in-context learning capabilities of large language models [**?**], specifically GPT-3. Note that GPT-3 is used for only the extraction of the high-level semantics of instructions into a symbolic form, rather than to extract the complete plan. This is in contrast to contemporary approaches like [**?**], [**?**], [**?**] etc, that offload the whole manipulation problem onto the reasoning and planning capabilities of the LLM.

Observe that our approach employs LLMs primarily for goal decomposition at a level of granularity that does not require visual and spatial understanding. This is motivated by the fact that LLMs do not have access to modalities other than text during training or prompting.

Visual grounding module We assume access to dense disentangled representations of visual and spatial primitives such as colors, shapes and spatial relations (Table 3.1) in the form of embedding vectors and MLPs [**?**]. We pre-train these representations using unsupervised learning on language-based primitive manipulation tasks as in [**?**]. Arguments to the goal predicate, outputted by GPT-3, are grounded into the scene by using these representations. The subprogram corresponding to arguments of goal predicates are compositional expressions of typed operators inspired from [**?**].

Type	Primitives	Example Semantics
Visual	`Red(),Green(),..`	`Red(obj)`$\in \{0, 1\}$
Spatial	`Left(),Right(),..`	`new_pos:=Left(obj_pos)`

Table 3.1: **Description of pre-trained concept primitives** Visual concepts like `Red` output probability of the object being red respectively given the object embedding (from object detectors). Spatial concepts like `Left` can be used to sample positions that are to the left of the given input position.

Non Terminal	Productions	
PROGRAM	(SUBGOAL|ITERATIVE_STATEMENT);PROGRAM	
ITERATIVE_STATEMENT	for(CONSTANT,CONSTANT,CONSTANT): PROGRAM	Program
SUBGOAL	concepts $\in$ Program Library	
CONSTANT	0 | VAR | INC CONSTANT	

Library := {assign_head, keep_at_head, move_head} $\cup$ { acquired concepts }

Table 3.2: DSL and program library for compositional representation of higher order concepts

Compositional, programmatic representation of higher order concepts We aim to learn representations for concepts that generalize to inductively larger instances. For example, after being exposed to stacks of height upto 4-5, the agent should be able to construct stacks of arbitrary heights. Programmatic representation inherently enables strong generalization as compared to blackbox-like dense representations that struggle to generalize beyond the training distribution. In response, we hypothesize a symbolic program space to represent higher-order concepts. Figure 3.2 presents the DSL that defines the concept space. Iterative constructs (`FOR`) in the DSL enable inductively generalizing representations of concepts.

Symbolic Program Search The algorithm 1 describes the symbolic program search algorithm. We define a prior over programs using a probabilistic grammar (the DSL), enumerate top K programs based on prior likelihoods and then select the program which on simulated execution results in scenes closest to the groundtruth final scenes. Proximity of final scenes is quantified by $\mathcal{R}$ defined as:

$$\mathcal{R}(S^F, \tilde{S^F}) := \Sigma_b IOU(S^F.bbox_b, \tilde{S^F}.bbox_b)$$

where $S^F.bbox_i$ refers to the coordinates of the $i-th$ bounding box in the image S^F. We circumvent the exponential blowup in program search by (i) defining the prior such that more concise programs are preferred and explored first (ii) employing a curriculum learning approach: introducing concepts in increasing order of complexity, such that harder concepts are able to reuse the representation of simpler ones, allowing more concise programs.

Algorithm 1 *ProgramSearch*

Input: subgoal : $concept(args)$, demos $\mathcal{D}$, max iters L, threshold t
for i in $1, \ldots, L$ **do**
    $p_i \leftarrow generate\_next\_program()$ #returns programs in increasing order of prior probability
    $cumulative\_loss_i \leftarrow \Sigma_{d \in \mathcal{D}} \mathcal{R}(S_d^F, simulate(p(args), S_d^I))/|\mathcal{D}|$
    if $cumulative\_loss_i \leq t$ **then return** p_i
    end if
end for

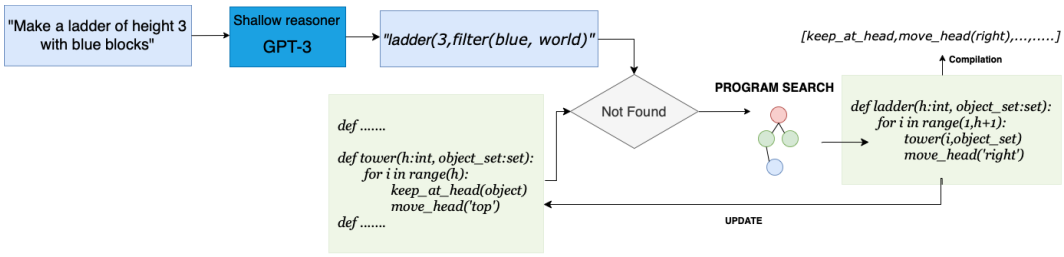


Figure 3.3: The language instruction is decomposed into a sequence of high-level goal predicates using an LLM as a *shallow reasoner* module. The goal predicates are grounded into the scene using pre-trained dense visual representations. The *program library* maintains programmatic representations of the spatial grounding of concepts acquired over time. If a novel concept is encountered, search in program space defined by the DSL 3.2 and guided by the final scene leads to acquisition of the new concept, updated in the program library.

3.3.3 Lifelong Learning

The program library acts as a repository of concept semantics that have been acquired over the course of the agent's experience. When a goal predicate comes to the agent, it searches through the library to get a specification that matches the predicate label and signature. If the search fails, the agent performs a tree search over the program space guided by the images of the final scene to acquire the concept semantics. The program library keeps growing along with the agent's experience, enabling life-long learning.

3.4 Experimental Setup and Results

The goal of our experiments was to compare the performance of our proposed approach to existing baselines. We describe the details of our dataset, baselines, experimental methodology and results.

Data collection We work with a table-top environment similar to the one used in K et al. [**?**]. It uses a Franka Emika Panda robot arm to collect the data. Each instance consists of a number of blocks each with different shape, size and color attributes. Total number of blocks in a given instance varies from 3 to 15. For an environment with a given number of objects on the table, we first randomly decide its composition (attributes), and

Model	Tower	Rows	Staircase
Neural-only	0.10	0.16	0.21
GPT-4	0.95	0.83	0.60
Ours	**1.0**	**1.0**	**1.0**

Table 3.3: Partial plan correctness of our approach compared to baselines

Time to acquire concept(s)	Tower	Staircase
Curriculum	70	69
No-curriculum	68	TIMEOUT

Table 3.4: Analysis of the effect of curriculum on learning time

then place this on an empty location with an orientation, both decidedly randomly. Each datapoint consists of a triplet consisting of a natural language instruction, initial scene and final scene. The instructions correspond to constructing towers, rows, and staircases with objects of specific color and/or shape as in 3.1.

Baselines: We compare against a purely LLM-based baseline. Given any instruction, we pass it through an LLM (GPT-4), along with a few examples of input instruction and output plan pairs in-context, and use the resulting output as the program to be executed, as is. Specifically, there is no further refinement done to the output of the LLM. This baseline is inspired by recent work on using LLMs for the off-the-shelf planning by doing careful prompt engineering [**?**]. These programs are directly passed to be executed by the underling neuro-symbolic planner [**?**] to the get final scene. Additionally, we devise a neural-only baseline inspired from [**?**]. The architecture consists of encoder-decoder transformer models that take multi-modal input in the form of language instruction and object embeddings from the input image and output a sequence of pick-place actions to be executed by the robot.

Experimental Methodology: For both our approach and the LLM baseline, we create a prompt consisting of a few examples of what the LLM is supposed to do in each case. Whereas in our case, the prompt consists of a few examples of instruction mapped to the corresponding program sketch, for the LLM baseline, the prompt provides examples of mapping a given instruction to the low-level plan in the language of the agent.

Results: Table 3.3 presents the relative comparison of the two algorithms in terms of their performance on the test set. We compare them on the metric *Partial plan correctness*: the fraction of objects that are placed correctly during plan exuection. Our approach performs significantly better than the baselines, which either fail to give a correct program or fail to generalize beyond the training set.

Inductive Generalization: We evaluate the performance of our approach and the baselines on concept instances that are significantly larger than those seen during training. The results are summarised in the plot 3.4. Our approach outperforms the baselines, demonstrating robust inductive generalisation.

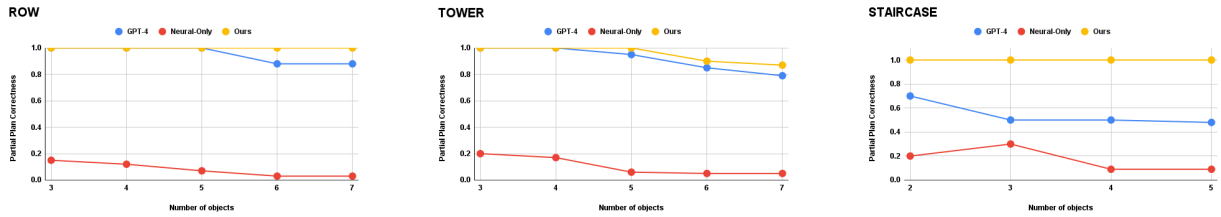


Figure 3.4: Performance comparison over inductively larger concept instances

3.5 Conclusion

We present a neuro-symbolic framework for learning inductively generalizable concepts for the problem of language guided robot manipulation. Unlike previous approaches, we do not rely solely on the knowledge of LLMs to learn *concept programs* and are able to learn complex concepts. Experiments demonstrate that our model generalizes to indcutively larger instances than seen during training, in contrast to existing baselines. uture work includes (i) learning more intricate concepts, such as making letters of the alphabet (ii) a tighter coupling of visual grounding and goal execution, required in *making a tower with blocks of alternate colours* (iii) Using neural architecture or large language models to provide stronger priors to accelerate program search to learn more complex concepts requiring longer programs.

3.6 Limitations

One of limitations of the current work is that we have not experimented with interleaved learning of novel concepts, along with fine-tuning of underlying neural modules. We believe such an experimentation will be bring out the real strength of our approach compared to existing models. Second limitation is that some of the functions (such as 'Tower') in our program space are assumed to have a single argument, assumed to be of type int. This limits the applicability by forcing any other attributes to be global, and therefore, our current approach can not handle commands such as"Build a tower of height 2 of `blue` blocks, and a tower of height 3 of `green` blocks, next to each other" which require localized handling of attribute values `blue` and `green`. We leave these limitations as avenues for future work.

Bibliography

Chapter 4

Discussion and Conclusion

To clear some clutter, you can separate chapters into folders and subfiles like this one or keep everything in the main file as per your conveniece.

Each subfile can be compiled by itself. It takes the preamble from the main file and compiles only the file you are working on. This reduces compile time and helps you focus. Citations might not work within subfile but will work in the main file.

Figure 4.1: using image from current working directory

Figure 4.2: using image from current working directory

Appendix A

A SAMPLE APPENDIX

Just put in text as you would into any chapter with sections and whatnot. Thats the end of it.