Closed Loop LLM-Based Planning and Execution with Visual Reasoning

Sohan Anisetty
College of Computing
Georgia Institute of Technology United States
sanisetty3@gatech.edu

Archana Kutumbaka
College of Computing
Georgia Institute of Technology United States
archanakutumbaka@gatech.edu

Bhushan Pawaskar
College of Computing
Georgia Institute of Technology United States
bpawaskar3@gatech.edu

Chunyue Xue
College of Computing
Georgia Institute of Technology United States
chunyuexue@gatech.edu

Keywords: Large Language models, Visual Question Answering, Feedback based Task Planning, Robot Manipulation

Abstract: This paper addresses the issue of incorporating feedback and reasoning into robot task planning methods. While humans naturally follow an act-see-reason-act paradigm, current robot planning approaches often lack the ability to incorporate feedback from actions and recover from failures. In this research, we propose a novel approach that combines large language models (LLMs) with a visual reasoning system (VRS) inspired by the human visual cortex. Our method involves using LLMs as a planner to break down high-level instructions into executable steps and incorporating a reasoning mechanism that interacts with the VRS to gather information about task progress and preconditions for successful execution of subsequent steps. By leveraging this feedback, our system is capable of recovering from failures and adapting its plans accordingly. Experimental validation is performed on tabletop manipulation tasks, comparing the performance of our closed-loop planning and execution system against baselines that lack feedback integration.

1 Introduction

In our pursuit of embodied intelligence, we aim to accomplish tasks such as ”preparing a meal” and ”arranging the table,” drawing inspiration from the innate abilities of humans. These tasks involve a subconscious combination of high-level planning, perceptual feedback, and low-level control. High-level planning entails setting goals and determining the sequence of actions required for their achievement. Perceptual feedback involves processing sensory information to understand the current state of the world and adjust the plan accordingly. Low-level control focuses on executing specific actions and movements based on the plan and perceptual feedback.

To address complex task planning problems, two predominant approaches have historically been utilized. The first approach, Task and Motion Planning (TAMP) [1, 2, 3], integrates high-level symbolic planning with low-level motion planning. The second approach, Hierarchical Reinforcement Learning (HRL) [4, 5] methods, decomposes complex tasks into a hierarchy of sub-tasks and learns policies to accomplish each sub-task independently. However, both these methods lack the ability to incorporate feedback from past actions into their planning, rendering them susceptible to compounding errors. Incorporating the outcomes of previous actions into future planning is challenging due to the requirement for a deep understanding of real-world semantics and knowledge.

Recently, Large Language Models (LLMs) have garnered attention within the AI community for their human-level text completion and instruction-following capabilities. These LLMs demonstrate a comprehensive understanding of the rich semantics of the world. Leveraging this internalized knowledge, some researchers have explored the use of LLMs in robot planning, departing from traditional symbolic approaches to high-level task planning. For instance, a few-shot LLM planner has been proposed to leverage the internalized knowledge of LLMs. Another approach introduced feedback by incorporating the success value of executed actions. However, these methods fail to reason about the scene and identify whether preconditions for successful execution of subsequent steps have been met, hindering their ability to recover from failure.

In this paper, we present a novel approach that addresses the limitations of existing robot task planning methods. Our approach integrates the semantic understanding and common-sense reasoning capabilities of large language models (LLMs) with an effective visual reasoning system (VRS), akin to the human visual cortex. By combining these two components, our objective is to enable robots to learn from action consequences, reason about execution progress, and dynamically adapt their plans. Our approach is a cyclic three-stage method that involves 1) Decomposing the overarching task to simple sub-tasks and their corresponding preconditions using Large Language Models (LLM) [6, 7], 2) Executing them using low-level language grounded policies [8, 9] which are equivalent to learnt skills of humans like 'picking' and 'placing' and 3) Providing feedback about progress of tasks executed in history back to the LLM for re-planning. This feedback-based planning at each step allows the model to adapt to unanticipated failure modes and recover effectively. Our approach relies on Visual Language Models(VLMs)[10, 11, 12, 13, 14] to infer semantic and spatial information. Being trained on large-scale internet data, these VLMs generalize poorly to synthetic tabletop environments. We adapt existing VLMs to robotic applications by fine-tuning on data from the Clevr[15] visual reasoning synthetic dataset and also develop a custom synthetic tabletop dataset to generate randomized data for multiple block based tasks like stacking and pick-and-place operations. The key contributions of our work can be summarized as follows:

1. LLM-based Planner: Leveraging the power of LLMs, we develop a planner capable of breaking down high-level instructions into a sequence of executable steps. By harnessing the semantic understanding and reasoning abilities of LLMs, our planner generates effective plans that consider the nuanced requirements of the given task.
2. Reasoning Mechanism with VRS: Our system incorporates a reasoning mechanism that interacts with the VRS. This mechanism actively seeks feedback from the visual system, mirroring how humans perceive the consequences of their actions. By asking targeted questions, the reasoning mechanism gains insights into task progress and identifies the preconditions necessary for successful execution of subsequent steps.
3. Closed-loop Planning and Execution: The integration of the planner and reasoning mechanism forms a closed-loop cycle. The planner generates an initial plan, while the reasoning mechanism continuously monitors execution progress and adjusts plans based on visual feedback. This closed-loop process enables the system to recover from failures, adapt to changing circumstances, and improve overall task execution efficiency.

To evaluate the effectiveness of our approach, we conduct experiments involving tabletop manipulation tasks. We compare the performance of our closed-loop planning and execution system against 2 baselines: no feedback, and success detection feedback and achieve an average improvement of 16% across tasks compared to other baselines.

2 Related Work

2.1 Task Planning with Language Models

The task planning problem aims to come up with a sequence of actions, or a plan, to achieve a goal. It requires the agent to autonomously reason about the state of the world using an internal model and also understand the interactions possible. Traditionally, they are approached using Planning Domain Description Language (PDDL) [16, 17]. PDDL represents the underlying principles governing a particular domain, such as the available predicates, feasible actions, the composition of complex actions, and the consequences of executing those actions. PDDL is an action language and thus describes the parameters, preconditions and effects of every possible action. However, it is impossible to scale such an action-language approach to open-world settings because we cannot give a comprehensive description of the physics of the real world. The most successful approaches for open-world task planning rely on large language models (LLMs) [18] [19]. These models have an extensive internalized semantic understanding of the world, making them ideal for open-world task planning. However, their lack of grounding in the real world makes it challenging for them to generate feasible and situated plans without the context of the current state of the environment, the set of executable actions, and the consequences of each action. [20], [6] have addressed this issue by providing context and generating plans using the structure of python code, which helps in grounding the generated plans because of their structural regulations. Inner Monologue [21] takes a closed-loop approach and notifies the planner about the success at each stage, resulting in redoing of failed actions. Building on these methods, our work uses LLMs to generate programmatic situated plans and pre-conditions/post-conditions for each executable action. We check whether the preconditions are satisfied before executing an action, and if not, prompt the LLM for a recovery plan. We use the post-conditions or effects of an action to ensure that the progress of task so far has not been nullified by a counterproductive action.

2.2 Language-Conditioned Manipulation

[22, 23] have been proposed for conditioning agents with language instructions, but require thousands of human-teleoperated behavior cloning demonstrations. Recent methods like PerAct [9] learn perceptual representations of actions for 6-DoF manipulation and can learn robust multi-task policy with just a few minutes of training data using Perceiver transformers[24]. CLIP[25] has been employed in several robotics and embodied settings in a zero-shot manner [26], or combined with Transporter networks [27] as in CLIPort [8]. Socratic Models[28] combines several foundation models (e.g., GPT-3 [29], ViLD [13]) and language-conditioned policies, for manipulating objects in a simulated vision-based robotic manipulation environment. We use CLIPort to execute actions provided by the high-level planner in the Ravens[27] simulation environment.

2.3 Visual Reasoning

Masked language modeling and representing images as quantized tokens in vision-language tasks have seen a rise in popularity following the success of BeIT[30, 31]. OFA[14] introduces a unified Seq2Seq framework for task-agnostic modeling using an encoder-decoder framework. They pre-train the model simultaneously on 5 multi-modal and 3 uni-modal tasks. In contrast, BeIT3[32] regard images as a foreign language and use only one pretraining task; mask-then-predict, to train a modular Multiway Transformer[33]. However, calculating the full self-attention sequences of image-text input is computationally expensive and prone to information asymmetry. Instead of fusing the modalities at the same level, mPLUG[34] fuses vision and language at disparate levels by using skip connections across the vision encoder layers increasing efficiency. Flamingo[11], a few-shot model, added cross-attention between image and text encoders and introduced the first model that can ingest arbitrarily interleaved images, videos, and text. Prior research has indicated that VLMs suffer in encoding relational information.[35, 15]. Although previous works attempt to train models on datasets specific to spatial reasoning,[36, 37] they generalize poorly to novel scenes. The closest body of work to our objective is SORNet[38]. They perform spatial reasoning on a tabletop stacking task by jointly encoding an image of the scene and an arbitrary number of target object images using a ViT[39] network. The encodings corresponding to the object queries are passed through classification networks to classify unary and binary object relations. However, the requirement of queried object images and task-specific classification networks makes it generalize poorly to arbitrary objects and renders it unable to function on different tasks without re-training.

The VQAv2[40] and VG-QA[41] datasets are popularly used for visual question answering (VQA). However, both of these datasets consist of general internet data and are not suitable for tabletop spatial reasoning tasks. Clevr[15] generates both attribute and spatial relationship labels for synthetically generated scenes making it highly customizable and enabling the extraction of ground truth data for novel scenes.

3 Problem Statement

3.1 Problem Definition

Consider a discretized system of n steps from start state s_0 to goal state $g(L)$. We consider the problem of embodied task planning where an agent needs to generate a sequence of n discrete low level actions a_k, in order to accomplish a high-level natural language instruction L given in english like "Place all fruits in the bowl". The agent has access to observations o_k of the world state s_k, at the start of every action a_k, in the form of RGB image inputs. The agent has to find a policy function π such that, given an instruction L, the function maps the observation space to actions that drive the state toward $s_n = g(L)$:

$$Find : \pi(o_k, L) \rightarrow a_k$$
$$\forall k \in Z : 0 \leq k \leq n$$

constraint to:

$$f(s_k, a_k) = s_{k+1}$$
$$s_0 = start$$
$$s_n = goal = g(L)$$

Our model formally consists of the following elements:
s_k : world state
o_k : world observation
a_k : actions
$f(s_k, a_k)$: system dynamics
L : Language instruction
c_k : code plan
i_k : instructions
q_k : question queries
r_k : response feedback
Refer appendix for details on each of these.

4 Method

Our method comprises of four sub-modules:

1. **Controller** The controller acts as the central hub, coordinating the planner, actor, and VRS to execute high-level instructions. It processes information from sub-modules, directing them to ensure successful task execution and facilitating adaptive decision-making based on real-time feedback. It is similar to our brain
2. **Actor** The actor is responsible for carrying out the actions specified by the controller, translating the planned steps into tangible movements and interactions with the environment.
3. **Visual Reasoning System (VRS)** The VRS, akin to the human visual cortex, captures visual feedback and perception of the task execution progress. It actively interacts with the controller, providing responses for the controller's queries that are generated by the LLM during planning. It allows the controller to ensure smooth execution or revert back to LLM for re-planning when things aren't progressing as expected.

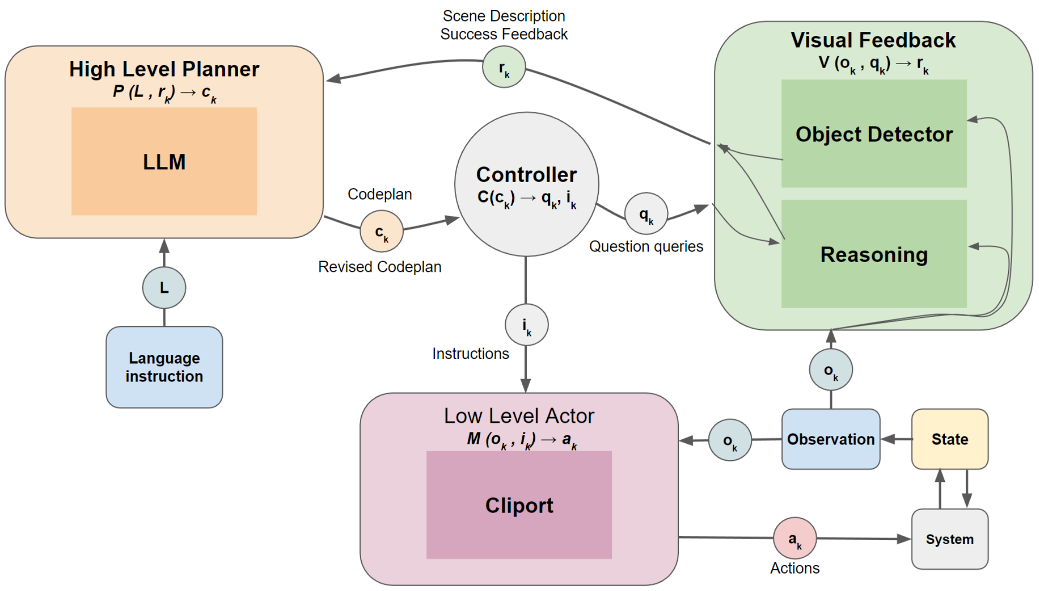
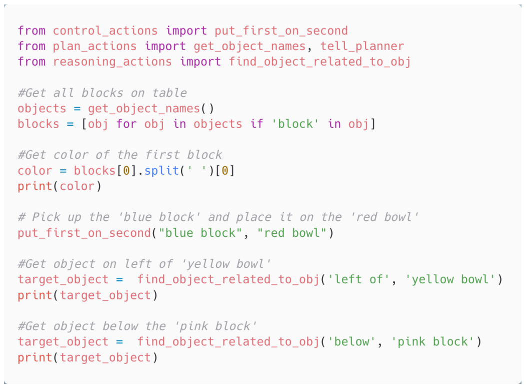


Figure 1: The LLM planner takes input a high-level instruction and generates a codeplan. The controller gets the scene description (objects on table) from out-of-box object detection method (ViLD) and creates an executable from the codeplan which comprises of actions and queries for this tabletop setting and particular task. The actions are executed using an actor (CLIPort) and any pre-condition failures are detected using our reasoning system and the corresponding feedback is passed to controller to adaptively modify the executable or prompt the LLM for a new code-plan for recovery.

4. **Planner** The LLM planner, with its the semantic understanding breaks down high-level instructions into a sequence of executable steps and also specifies necessary preconditions as progress checks. It generates effective plans that consider the nuanced requirements of the task, enabling the controller to coordinate and direct the execution process.

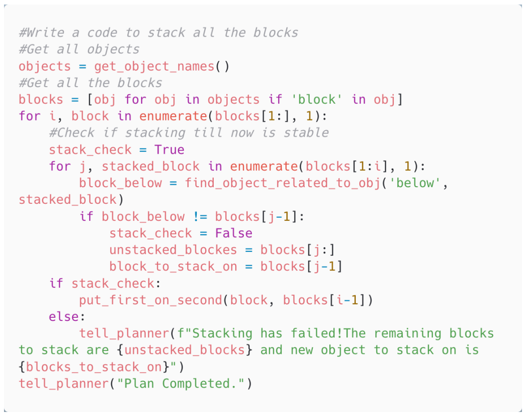
4.1 Generating codeplans

LLMs [7], [42] have shown great performance as hierarchical planners with few-shot prompting [6], [43]. Following this line of work, we carefully craft a few simple prompts for the TableTop environment assuming low-level skills (pick and place) as action primitives. These prompts are used to convey the skill repertoire of the actor and the reasoning capabilites of the VRS to the LLM Planner. To restrict the primitives (skills) available to the LLM, we import them as functions with their prototype from the ”actions” module. Similarly the interface with the VRS is also imported as a function from a ”reasoning” module. The planner expects an ”objects” list which contains existing objects on the tabletop that are identified using an object detection module (ViLD)[13]. Our prompt needs to teach the LLM usage of these skills and reasoning abilities and to this end, we provide give example usage of these imported functions Fig(2(a)) and an example code-plan for task Fig(2(b)). Based on this, we generate context-agnostic code plans for a set of 3 tasks by asking our chat completion API to ”Write a code to (instruction)”. Our plans are context agnostic because we only assume access to an ”objects” list but no assumptions are made about the elements of the list. This helps make our code plan robust to different initialisations of the tabletop as long as our object detector is reliable. In our work, we use ”gpt-3.5-turbo” model from the OpenAI Chat Completion API as our LLM planner. This code-style task-planning is preferable over conversational style in [21] because it strictly enforces the grounding of actions and reasoning interactions and also effectively abstracts the different capabilities of our systems as simple function calls.



```python
from control_actions import put_first_on_second
from plan_actions import get_object_names, tell_planner
from reasoning_actions import find_object_related_to_obj

#Get all blocks on table
objects = get_object_names()
blocks = [obj for obj in objects if 'block' in obj]

#Get color of the first block
color = blocks[0].split(' ')[0]
print(color)

# Pick up the 'blue block' and place it on the 'red bowl'
put_first_on_second("blue block", "red bowl")

#Get object on left of 'yellow bowl'
target_object =  find_object_related_to_obj('left of', 'yellow bowl')
print(target_object)

#Get object below the 'pink block'
target_object =  find_object_related_to_obj('below', 'pink block')
print(target_object)
```

Figure 2(a) Prompts for demonstrating abilities



```python
#Write a code to stack all the blocks
#Get all objects
objects = get_object_names()
#Get all the blocks
blocks = [obj for obj in objects if 'block' in obj]
for i, block in enumerate(blocks[1:], 1):
    #Check if stacking till now is stable
    stack_check = True
    for j, stacked_block in enumerate(blocks[1:i], 1):
        block_below = find_object_related_to_obj('below',
stacked_block)
        if block_below != blocks[j-1]:
            stack_check = False
            unstacked_blockes = blocks[j:]
            block_to_stack_on = blocks[j-1]
    if stack_check:
        put_first_on_second(block, blocks[i-1])
    else:
        tell_planner(f"Stacking has failed!The remaining blocks
to stack are {unstacked_blocks} and new object to stack on is
{blocks_to_stack_on}")
tell_planner("Plan Completed.")
```

Figure 2(b) Sample codeplan for the planner

4.2 Visual Reasoning

We use OFA[14] as our backbone model for visual question answering. OWe use the base variant of the architecture which has 180M parameters, a ResNet152 vision encoder to obtain image embeddings, byte-pair encoding (BPE)[44] tokenization for text, and an encoder-decoder style transformer with 12 layers and 16 heads. Image and text embeddings are passed into the encoder as input while the decoder is trained in an autoregressive manner conditioned on encoder output embeddings. We freeze the encoder and opt to only finetune the autoregressive decoder to retain the rich joint internal representation of image and text .

To improve OFA's[14] performance for our tasks, we followed a data generation approach outlined in CLEVR to produce new randomized scenes in Blender and used ground truth knowledge of object relations to create question-answer pairs to develop robotic task datasets. To replicate scenes from the cliport environment, we modified CLEVR to resemble cliport and focused on two specific scenes: stack-block-pyramid and put-bowl-in-block.

Pre-computed spatial relation scene data was passed through the CLEVR question generation pipeline which enabled automated structured extraction of answers to the questions. We generated over 4000 images, with approximately 20-25 questions for each image, resulting in a total of around 100k question-answer pairs for the training dataset.

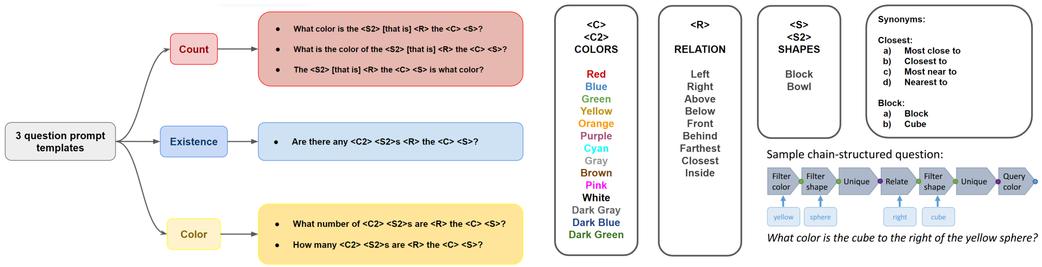

3 question prompt templates
Count
Existence
Color
What color is the <S2> [that is] <R> the <C> <S>?
What is the color of the <S2> [that is] <R> the <C> <S>?
The <S2> [that is] <R> the <C> <S> is what color?
Are there any <C2> <S2>s <R> the <C> <S>?
What number of <C2> <S2>s are <R> the <C> <S>?
How many <C2> <S2>s are <R> the <C> <S>?
<C> <C2> COLORS
Red
Blue
Green
Yellow
Orange
Purple
Cyan
Gray
Brown
Pink
White
Dark Gray
Dark Blue
Dark Green
<R> RELATION
Left
Right
Above
Below
Front
Behind
Farthest
Closest
Inside
<S> <S2> SHAPES
Block
Bowl
Synonyms:
Closest:
a) Most close to
b) Closest to
c) Most near to
d) Nearest to
Block:
a) Block
b) Cube
Sample chain-structured question:
Filter color
Filter shape
Unique
Relate
Filter shape
Unique
Query color
yellow
sphere
right
cube
What color is the cube to the right of the yellow sphere?

Figure 2: This section provides an overview of the question generation process. The questions are generated using six distinct question templates, which are categorized based on their answer types into three categories: integer numbers, binary Yes/No, and color. Each template includes parameters that are replaced by corresponding elements of the same parameter type, namely shape, color, and relation. These elements are selected randomly and filtered using a rejection sampling heuristic to ensure the quality of the questions. Once an acceptable question has been identified, the parameter elements are randomly replaced with a dictionary of synonyms to diversify the question prompts. Additionally, the prompts enclosed in square brackets are also randomly selected to further enhance the variety of the question prompts.

We finetune the model using the Clevr dataset[15], VQAv2[40] dataset, and a custom block based task dataset (section 4.2). In total, we train on a dataset of 300K images and about 1.2M questions. A comparison of performance on Clevr and custom tabletop dataset between our fine-tuned model and model pre-trained only on the VQAv2[40] dataset is given in table 1. We train the model for 40,000 epochs, with a cosine decay learning rate schedular with warmup, a learning rate of 5e-5, and the AdamW[45] optimizer.

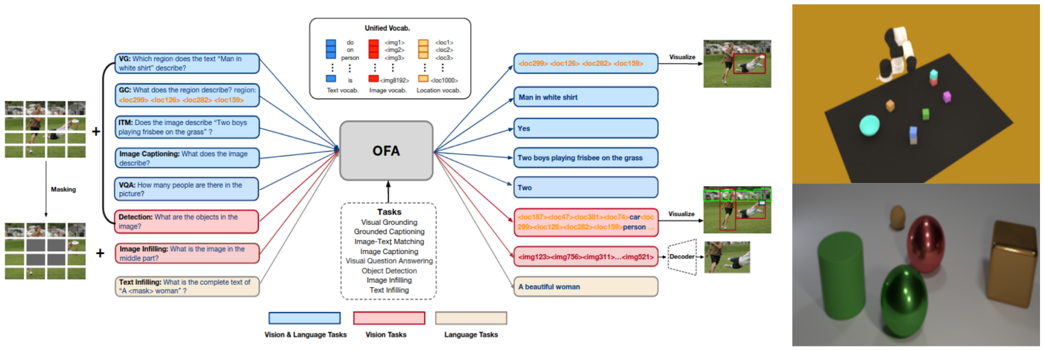

Unified Vocab.
do
on
person
is
Text vocab.
<img1>
<img2>
<img3>
<img8192>
Image vocab.
<loc1>
<loc2>
<loc3>
<loc1000>
Location vocab.
Masking
VG: Which region does the text "Man in white shirt" describe?
GC: What does the region describe? region: <loc299> <loc126> <loc282> <loc159>
ITM: Does the image describe "Two boys playing frisbee on the grass" ?
Image Captioning: What does the image describe?
VQA: How many people are there in the picture?
Detection: What are the objects in the image?
Image Infilling: What is the image in the middle part?
Text Infilling: What is the complete text of "A <mask> woman" ?
OFA
Tasks
Visual Grounding
Grounded Captioning
Image-Text Matching
Image Captioning
Visual Question Answering
Object Detection
Image Infilling
Text Infilling
<loc299> <loc126> <loc282> <loc159>
Visualize
Man in white shirt
Yes
Two boys playing frisbee on the grass
Two
<loc187><loc47><loc381><loc74>car<loc299><loc126><loc282><loc159>person ...
Visualize
<img123><img756><img311>...<img521>
Decoder
A beautiful woman
Vision & Language Tasks
Vision Tasks
Language Tasks

Figure 3: From left to right; OFA[14] unifies various multi-modal tasks in a seq2seq framework, (top) a sample from the custom block stacking and pick-place dataset, (bottom) example image from the Clevr[15] visual reasoning dataset.

4.3 Simulation and Manipulation Policy

We employ the Ravens tabletop simulation environment alongside CLIPort[8]. This combination enables us to leverage a multi-task pre-trained model that has been trained on our action set A using imitation learning. By integrating the CLIPort model into our framework, we can receive language-based goals from the LLM planner. After execution, the CLIPort model transmits the final state image to the visual reasoning model, facilitating the collection of valuable feedback. The primary focus of CLIPort is on identifying actions, thus the network's output consists of predicted pick and place affordances at each time step.

CLIPort determines these affordances by first attending to a local region to decide where to pick, then computes a placement location by finding the best match through cross-correlation of deep visual features. It uses 3, two-stream fully convolutional networks (FCNs) as Figure 4 that combine the semantic understanding of CLIP[25] with the spatial precision of TransporterNets[27]. The first FCN takes input γ_t and outputs affordance map $\mathcal{Q}_{pick}$, which is used to predict pick action $\mathcal{T}_{pick}$ (Eq 1). The second and third FCN take as input a crop centered around $\mathcal{T}_{pick}$ and the full input γ_t to output feature embeddings Φ_{query} and Φ_{key} respectively. These embeddings are used in equation 2 and 3 to obtain place location $\mathcal{T}_{place}$. Pixel coordinates to end-effector poses rely on carefully calibrated extrinsics between the robot's base frame and the RGB-D camera.

$$\mathcal{T}_{pick} = argmax_{(u,v)} \mathcal{Q}_{pick}((u,v)|\gamma_t) \tag{1}$$

Where $\mathcal{T}_{pick} \in \mathbf{SE}(2)$ is the end-effector pose for picking action, (u,v) is the pixel location, $\mathcal{Q}_{pick}$ is the pixel-wise prediction output, and γ_t is the heightmap and language inputs at timestep t.

$$\mathcal{Q}_{place}(\Delta\tau|\gamma_t, \mathcal{T}_{pick}) = \Phi_{query}(\gamma_t[\mathcal{T}_{pick}]) * \Phi_{key}(\gamma_t)[\Delta\tau] \tag{2}$$

$$\mathcal{T}_{place} = argmax_{\Delta\tau} \mathcal{Q}_{place}(\Delta\tau|\gamma_t, \mathcal{T}_{pick}) \tag{3}$$

Where $\Delta\tau$ is the potential placement pose, $\gamma_t[\mathcal{T}_{pick}]$ is a crop of heightmap centered at $\mathcal{T}_{pick}$, Φ_{query} and Φ_{key} are the outputs of the second and third Fully-Convolutional-Networks in the architecture.

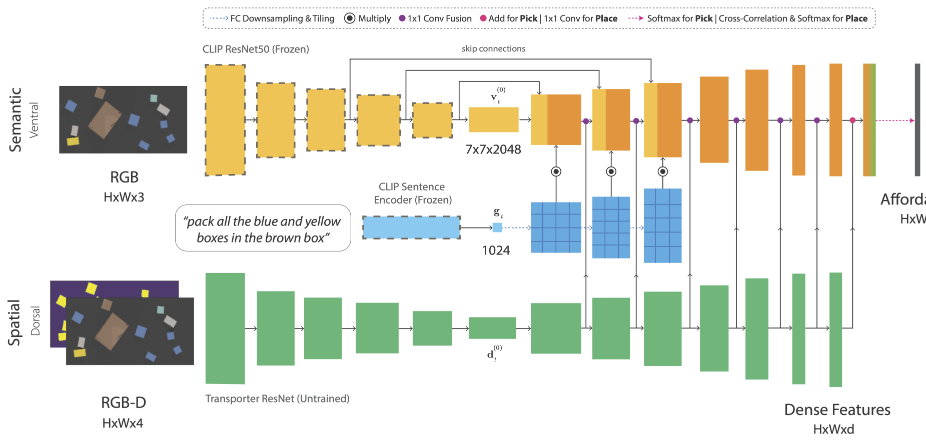

FC Downsampling & Tiling
Multiply
1x1 Conv Fusion
Add for **Pick** | 1x1 Conv for **Place**
Softmax for **Pick** | Cross-Correlation & Softmax for **Place**
CLIP ResNet50 (Frozen)
skip connections
Semantic
Ventral
RGB
HxWx3
$v_t^{(0)}$
7x7x2048
CLIP Sentence Encoder (Frozen)
"pack all the blue and yellow boxes in the brown box"
g_t
1024
Affordance
HxWx1
Spatial
Dorsal
RGB-D
HxWx4
$d_t^{(0)}$
Transporter ResNet (Untrained)
Dense Features
HxWxd

Figure 4: The two stream architecture of CLIPort

5 Experiment

5.1 Evaluating Visual Reasoning System

To evaluate the performance of our framework, we conduct tests in a tabletop environment. We modify the Ravens[27] simulation environment for tabletop settings.

CWe showcase the accuracy of OFA[14] on the Clevr[15] and custom block based stack dataset in table 1. We employ two training approaches: the "Custom" model, where we fine-tune exclusively on the custom block dataset and Clevr, and the "Mixture" model, where we fine-tune on a weighted combination of the custom block, Clevr, and VQAv2[40] datasets. We use the compositional generalization subset of the CLEVR dataset wherein trainA and valA contain objects of a particular color and texture while valB contains objects of different colors. The pre-trained model corresponds to the official weights of the base variant of OFA.The custom Block Stack dataset comprises images depicting stacked blocks, the Block Bowl dataset includes tabletop scenes involving the placement of multi-colored blocks in colored bowls, and the Global Table dataset contains questions related to global orientation and position with respect to the table and other blocks/bowls. The Unseen challenge dataset consists of variations of the block and bowl datasets, featuring unseen colors and varying levels of occlusion. We run all models five times, each on 2000 images, and calculate the average results. Similar to PaLM-E[46], we observe that training the models with a diverse mixture of robotics and general visual-language data yields a significant performance boost compared to training solely on the specific in-domain data. We also observed that both the pretrained model and our finetuned model suffer in questions having numerical answers. This is reflected in the relatively poor scores in the bowl and unseen challenge datasets, having a higher number of numerical questions compared to CLEVR and block stacking datasets.

Models	Clevr[15]		Block Stack	Block Bowl	Global Table	Unseen	VQAV2
	ValA	ValB					
Pretrained	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	52.33%
Custom	69.3%	65.9%	89.44%	56.2%	37.8%	38.3%	26.8%
Mixture	90.9%	88.5%	87.85%	55.9%	49.0%	42.2%	55.23%

Table 1: Comparing the accuracy between OFA[14] pre-trained model and 2 variations of our custom model finetuned on 2000 images each from Clevr[15], block tasks related dataset, and VQAv2[40] datasets. The performance of the "Custom" model, which was finetuned only on CLEVR and block datasets, shows poor performance on VQAv2 while finetuning on all available datasets has a considerably greater performance on VQAv2 while maintaining performance in robotics related tasks.

5.2 Closed-Loop Planning and Execution

We evaluate the effectiveness of our framework, called code-plan with full progress validation, and compare it against two baselines: code-plan without any feedback and code-plan with success detection. The first baseline executes the action sequence specified by the planner without receiving any feedback. The second baseline incorporates success detection, which checks if each action is successfully executed before proceeding to the next one. We use two complex high-level instructions: placing all blocks in bowls of the same color and placing all blocks in bowls of different colors. Our policy training includes both seen and unseen colors, resulting in comparable overall performance to the ”put-blocks-in-bowls-unseen-colors” task reported in the CILPort paper. It is worth noting that the CLIPort policy without oracle termination demonstrates significantly lower performance compared to the policy with oracle termination, as shown in Table 3.

To highlight the benefits of a visual feedback system, we compare the performance between validating all previously executed successful control actions and validating only the most recent control action. Validation refers to using our visual reasoning model to ensure that previously successful actions remain successful. The environment can be altered by more recent control actions, potentially undoing the effects of previous actions. For instance, we observed cases where the robot accidentally tipped over a bowl, causing its block to fall out. By validating all previous actions, the controller can detect any discrepancies and rectify them before continuing with the original task sequence. We conduct each experiment 15 times, randomly sampling the number and color of blocks on each occasion, and measure the number of successful attempts. A task is considered unsuccessful if the number of retries exceeds 20 or if the visual language model incorrectly predicts task completion. The results of these experiments are summarized in Table 3

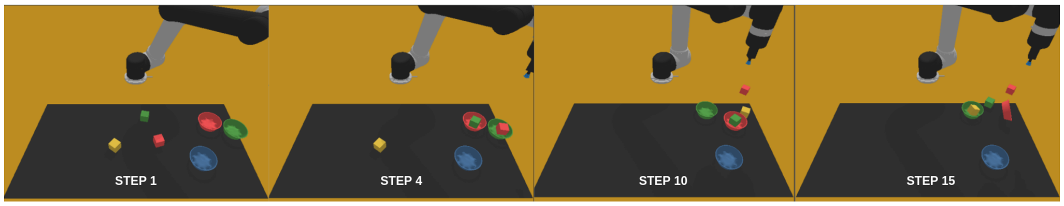


Figure 5: Environment states for the task ”placing blocks into mismatched bowls” using only most recent action validation. The robot arm fails to place the yellow block in the blue bowl and in attempting to retry this action, it displaces the red block. Since, it only retries the most recent action, i.e. placing the yellow block, it never attempts to put the red block back in the green bowl.

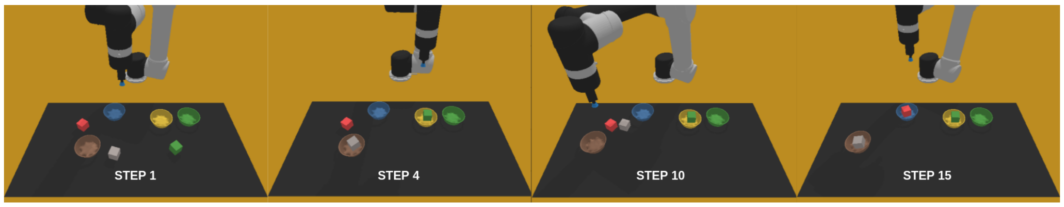


Figure 6: Environment states for the task ”placing blocks into mismatched bowls” using all previous action validation. The robot arm knocks out the gray block out of the brown bowl while executing the action place the red block in the blue bowl. Due to performing validation checks on all previous actions, the robot was able to place the gray block back into the brown bowl and continuing on with placing the red block in the blue bowl.

The performance for the task ”putting blocks in matching bowls” has lower success rate due to the VLM failing to identify the correct block in a bowl. This was due to the block and bowl color being so similar that the VLM predicted nothing was present in the bowl. Thus, we conducted the experiment with darker shades of the same color. The success percentage increased significantly as shown in Table 3. Some of the major failure cases where when the robot arm pushed the bowls/blocks to the edges of the table or when the robot arm flipped over the bowl. However, one interesting observation

Tasks	Codeplan	Codeplan+ Success	Codeplan+Progress (Ours)
Put blocks in matching bowls	0.0%	26.67%	33.33%
Put darker blocks in matching bowls	0.0%	33.3%	53.33%
Put blocks in different color bowls	6.67%	40.0%	60.0%

Table 2: Comparing the success rate between CLIPort pre-trained model without oracle termination and CLIPort pre-trained model with oracle termination using only most recent action validation and validating over all previous states. For the ”putting blocks in matching bowls task”, we conduct experiments on two types of blocks: the same color as the bowls and a darker shade of the same color.

was that the robot arm managed to flip the bowl right side up again and moved the block/bowl closer to the center when on the edge. This was majorly seen in the CLIPort + full validation experiments.

6 Conclusion

We have introduced a novel solution to tackle the issue of inadequate feedback in existing robot task and motion planning frameworks. Our approach integrates feedback and reasoning by combining large language models (LLMs) with a visual reasoning system (VRS), which mimics how humans process information in their visual cortex. By following a cyclic three-stage process of breaking tasks down into single-step sub-tasks, executing these sub-tasks while generating feedback, and dynamically re-planning when necessary, our method empowers robots to recover swiftly from failures and successfully accomplish long-term tasks. Through experiments conducted on simulated tabletop manipulation tasks, we have demonstrated that our approach surpasses baseline methods lacking a feedback system. Our proposed method represents a significant advancement in the development of more efficient and effective frameworks for robot task and motion planning.

While our proposed method represents a significant advancement, it is important to acknowledge certain limitations in our work. One limitation is that we utilized a pre-trained CLIPort model that was not specifically fine-tuned for our tasks, which could have influenced its performance. To address this, we intend to train and fine-tune our own model using a larger and more diverse dataset in future research. Another limitation lies in our visual language model (VLM), which may encounter difficulty in correctly identifying the block within a bowl when their colors are similar. To mitigate this issue, our future plans include incorporating depth information into the VLM to enhance its object discrimination capabilities.

We also aspire to implement our method in a more realistic environment, such as the iGibson 2.0 [47] simulator in the BEHAVIOR dataset[48] to showcase the advantages of our approach across a broader range of scenarios. By implementing the proposed method in a high-fidelity robotics simulator for complex household tasks, we can gain insights into how the method could be applied to a physical robot. Once validated, the method could then be adapted to work with an existing physical robot arm, such as the UR5, paired with an RGB-D camera system. Although we have not tested our pipeline on a real robot, the authors of CLIPort[8] showcased that CLIPort works in the real world without any architecture nor training data changes. Further, we train our visual reasoning model on real world data and synthetic data simultaneously, thus we expect some form of generalizability. However, to ensure optimal performance in the real world, the models used in our method would need to be fine-tuned to account for potential errors in real-world inputs and the execution process.

Name	Tasks
Sohan Anisetty	Responsible for VLM training, evaluation, and integrating CLIPort, VLM, and LLM planner. Moved OFA from Fairseq to HuggingFace, wrote training and evaluation scripts, dataloaders, converted weights, experimented with parallelization. Implemented CLIPort and VLM as callable APIs so that the plan generated by LLM can call these functions. Ran all the different variations of the experimentation tests. Wrote the VLM sections of Visual reasoning and Experiment part of the report.
Archana Kutumbaka	Responsible for prompt-engineering for LLM planner with feedback processing abilities. Task set ideation and experiment design. ViLD integration with the system. Report - Abstract, Introduction and some parts of Methods
Chunyue Xue	Responsible for Ravens simulation environment and CLIPort task setting up and modifying. Helped with framework integration. Did experiments with different splits. Wrote the parts related to CLIPort in Method, and the conclusion.
Bhushan Pawaskar	Responsible for Visual QA Model finetuning using CLEVR. Worked with Blender and CLEVR APIs to generate custom image question answers dataset for training the model on simulated images. Wrote the problem statement formulation, dataset generation pipeline, and methodology flow in the report.

Table 3: Contribution table

References

[1] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, 2014. doi:10.1109/ICRA.2014.6906922.

[2] L. P. Kaelbling and T. Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[3] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *I. J. Robotic Res.*, 28:104–126, 01 2009. doi:10.1177/0278364908097884.

[4] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[5] M. M. Botvinick, Y. Niv, and A. G. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009. ISSN 0010-0277. doi:https://doi.org/10.1016/j.cognition.2008.08.011. URL https://www.sciencedirect.com/science/article/pii/S0010027708002059. Reinforcement learning and higher cognition.

[6] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models, 2022. URL https://arxiv.org/abs/2209.11302.

[7] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

[8] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. *CoRR*, abs/2109.12098, 2021. URL https://arxiv.org/abs/2109.12098.

[9] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation, 2022. URL https://arxiv.org/abs/2209.05451.

[10] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu. Coca: Contrastive captioners are image-text foundation models, 2022. URL https://arxiv.org/abs/2205.01917.

[11] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning, 2022. URL https://arxiv.org/abs/2204.14198.

[12] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, C. Liu, M. Liu, Z. Liu, Y. Lu, Y. Shi, L. Wang, J. Wang, B. Xiao, Z. Xiao, J. Yang, M. Zeng, L. Zhou, and P. Zhang. Florence: A new foundation model for computer vision, 2021. URL https://arxiv.org/abs/2111.11432.

[13] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui. Open-vocabulary object detection via vision and language knowledge distillation. 2021. doi:10.48550/ARXIV.2104.13921. URL https://arxiv.org/abs/2104.13921.

[14] P. Wang, A. Yang, R. Men, J. Lin, S. Bai, Z. Li, J. Ma, C. Zhou, J. Zhou, and H. Yang. Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. *CoRR*, abs/2202.03052, 2022. URL https://arxiv.org/abs/2202.03052.

[15] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, 2016. URL https://arxiv.org/abs/1612.06890.

[16] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive science*, 6(2):101–155, 1982.

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

[18] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[19] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. URL https://arxiv.org/abs/2201.07207.

[20] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policis: Language model programs for embodied control. In *arXiv preprint arXiv:2209.07753*, 2022.

[21] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

[22] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. 2022. doi:10.48550/ARXIV.2202.02005. URL https://arxiv.org/abs/2202.02005.

[23] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[24] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. Hénaff, M. M. Botvinick, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver io: A general architecture for structured inputs amp; outputs, 2021. URL https://arxiv.org/abs/2107.14795.

[25] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[26] A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi. Simple but effective: Clip embeddings for embodied ai, 2021. URL https://arxiv.org/abs/2111.09888.

[27] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *CoRR*, abs/2010.14406, 2020. URL https://arxiv.org/abs/2010.14406.

[28] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence. Socratic models: Composing zero-shot multimodal reasoning with language, 2022. URL https://arxiv.org/abs/2204.00598.

[29] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[30] H. Bao, L. Dong, S. Piao, and F. Wei. Beit: Bert pre-training of image transformers, 2021. URL https://arxiv.org/abs/2106.08254.

[31] Z. Peng, L. Dong, H. Bao, Q. Ye, and F. Wei. Beit v2: Masked image modeling with vector-quantized visual tokenizers, 2022. URL https://arxiv.org/abs/2208.06366.

[32] W. Wang, H. Bao, L. Dong, J. Bjorck, Z. Peng, Q. Liu, K. Aggarwal, O. K. Mohammed, S. Singhal, S. Som, and F. Wei. Image as a foreign language: Beit pretraining for all vision and vision-language tasks, 2022. URL https://arxiv.org/abs/2208.10442.

[33] H. Bao, W. Wang, L. Dong, Q. Liu, O. K. Mohammed, K. Aggarwal, S. Som, and F. Wei. Vlmo: Unified vision-language pre-training with mixture-of-modality-experts, 2021. URL https://arxiv.org/abs/2111.02358.

[34] C. Li, H. Xu, J. Tian, W. Wang, M. Yan, B. Bi, J. Ye, H. Chen, G. Xu, Z. Cao, J. Zhang, S. Huang, F. Huang, J. Zhou, and L. Si. mplug: Effective and efficient vision-language learning by cross-modal skip-connections, 2022. URL https://arxiv.org/abs/2205.12005.

[35] F. Liu, G. Emerson, and N. Collier. Visual spatial reasoning. *arXiv preprint arXiv:2205.00363*, 2022.

[36] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Inferring and executing programs for visual reasoning, 2017. URL https://arxiv.org/abs/1705.03633.

[37] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[38] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation, 2021. URL https://arxiv.org/abs/2109.03891.

[39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. URL https://arxiv.org/abs/2010.11929.

[40] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. Bernstein, and L. Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016. URL https://arxiv.org/abs/1602.07332.

[42] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[43] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2022. URL https://arxiv.org/abs/2212.04088.

[44] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units, 2016.

[45] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.

[46] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. Palm-e: An embodied multimodal language model, 2023.

[47] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. E. Vainio, C. Gokmen, G. Dharan, T. Jain, A. Kurenkov, K. Liu, H. Gweon, J. Wu, L. Fei-Fei, and S. Savarese. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 455–465. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/li22b.html.

[48] S. Srivastava, C. Li, M. Lingelbach, R. Martín-Martín, F. Xia, K. Vainio, Z. Lian, C. Gokmen, S. Buch, C. K. Liu, S. Savarese, H. Gweon, J. Wu, and L. Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments, 2021. URL https://arxiv.org/abs/2108.03332.

7 Appendix

7.1 Problem definition

Action $a_k \in A$: A is the set of all possible actions the agent can take which is equivalent to the skill repertoire an agent possesses. Each action a_k can for our table top manipulation problem be formulated as a simple pick and place action and represented as a tuple of end-effector poses for pick and place: $a_k = (\mathcal{T}_{pick}, \mathcal{T}_{place})$.

State $s_k \in S$: S is the set of all possible environment states. s_0 is the start state and s_n is the goal state. The information contained in a state is not directly accessible to the agent. The agent can only access the observation o_k of a state s_k at any given step k

Observation $o_k \in O$: O is the set of all possible observations of the environment state. These observations are raw RGB camera images of the scene. $o_k \in R^{H \times W \times C}$. We assume that the agent is able to access the raw camera data without any noise or interference in the process.

System here refers to the dynamic model of the system given by f. This function determines what change will take place in the system state s_k at any given step when an action a_k is taken by the agent. $f(s_k, a_k) = s_{k+1}$

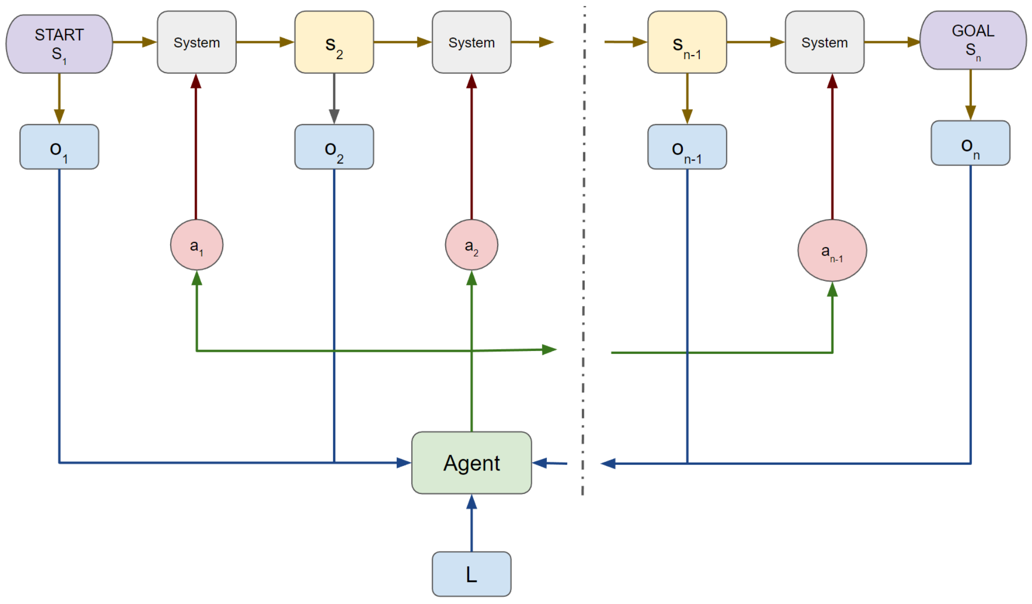


Figure 7: A high level overview of all the system interactions made by the agent. At every k^{th} step, the agent receives input of observations o_k. Based on the original natural language instruction L, the agent outputs an action a_k for that step. This continues for a total of n steps till the agent reaches the goal state s_n

Codeplan c_k :
Chain of tasks for a structured plan for successfully executing the long-horizon task
Example: putFirstOnSecond(yellow block, yellow bowl)

Instruction i_k :
Conversion of the codeplan tasks into a natural language instruction for the low level actor
Example: Put yellow block in the yellow bowl.

Query q_k :

A natural language question that validates the success of the previously passed instruction.
Example: “Is the yellow block in the yellow bowl?”

Response Feedback r_k :
Facilitates verification of success from the visual feedback model based on a given query.
Example: A list of objects in the scene [Red cube, Green bowl, Blue cube]; A boolean success response to the query [True] [False]

7.2 Example Data

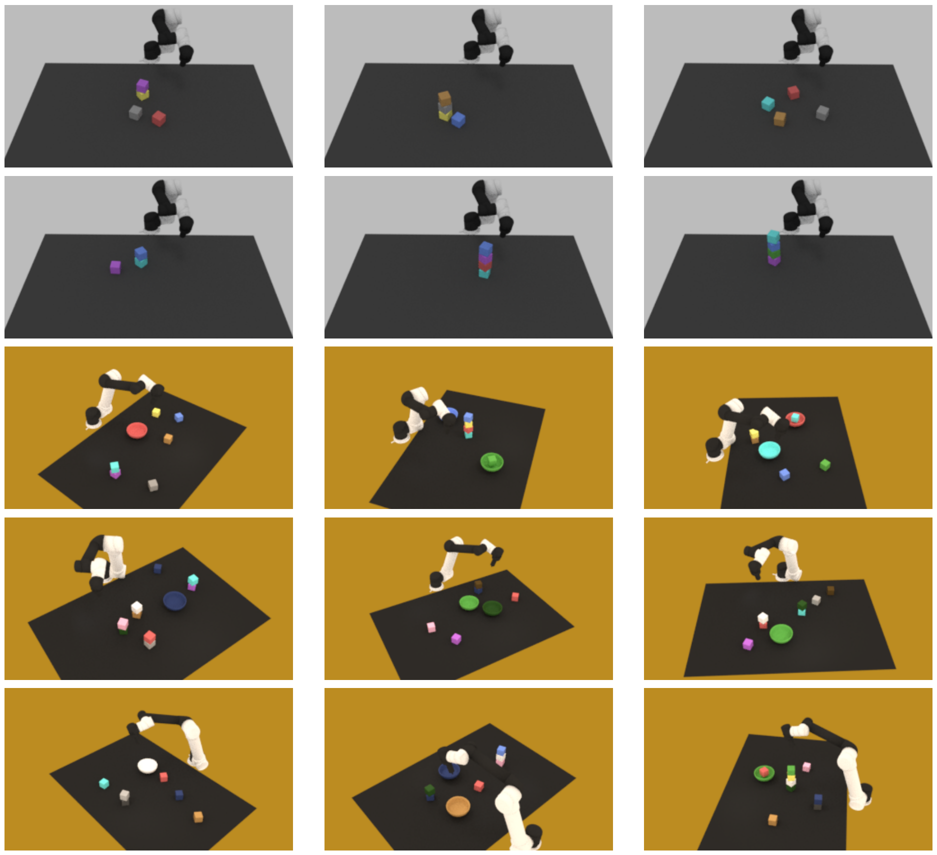


Figure 8: Example images from our custom dataset

7.3 Complete code plans

7.4 More examples of putting blocks in mismatched bowls

```
#Write a code to stack all blocks on the red block
from control_actions import put_first_on_second
from plan_actions import get_object_names, tell_planner
from reasoning_actions import find_object_related_to_obj

# List to track executed control actions
executed_actions = []

def validate_previous_actions():
    invalid_actions = []
    for index, action in enumerate(executed_actions):
        if action['type'] == 'put_first_on_second':
            obj_a = action['obj_a']
            obj_b = action['obj_b']
            current_obj_below = find_object_related_to_obj('below', obj_a)
            if current_obj_below != obj_b:
                invalid_actions.append((index, action))
    return invalid_actions

# Get all objects on table
objects = get_object_names()

# Filter out the blocks
blocks = [obj for obj in objects if 'block' in obj]

# Find the red block
red_block = None
for block in blocks:
    if 'red block' in block:
        red_block = block
        break

# Check if red block is present
if red_block is None:
    tell_planner("No red block found.")
else:
    # Remove red block from blocks list
    blocks.remove(red_block)
    # Start stacking on red block
    current_block = red_block

    while len(blocks) > 0:
        # Stack the next block on the current_block
        next_block = blocks.pop(0)

        while True:
            # Check if previous actions are still valid
            invalid_actions = validate_previous_actions()

            if not invalid_actions:
                # Stack the next block on the current_block
                put_first_on_second(next_block, current_block)
                executed_actions.append({
                    'type': 'put_first_on_second',
                    'obj_a': next_block,
                    'obj_b': current_block
                })
                print(f"Stacked the {next_block} on the {current_block}.")

                # Update current_block
                current_block = next_block
                break
            else:
                for index, invalid_action in invalid_actions:
                    tell_planner(f"Validation failed. Action {index}:
{invalid_action['type']}({invalid_action['obj_a']}, {invalid_action['obj_b']}) is no
longer valid. Redoing the action.")
                    # Redo the invalid action
                    put_first_on_second(invalid_action['obj_a'], invalid_action['obj_b'])

    tell_planner("Plan Completed.)
```

Figure 9: An example generated code plan from an input prompt: ”Write code to stack all blocks on red block” example code snippet. Here, sensor is a call to the VLM model, and gripper/end effector are calls to Cliport[8]. The prompts are shown in Fig 2.

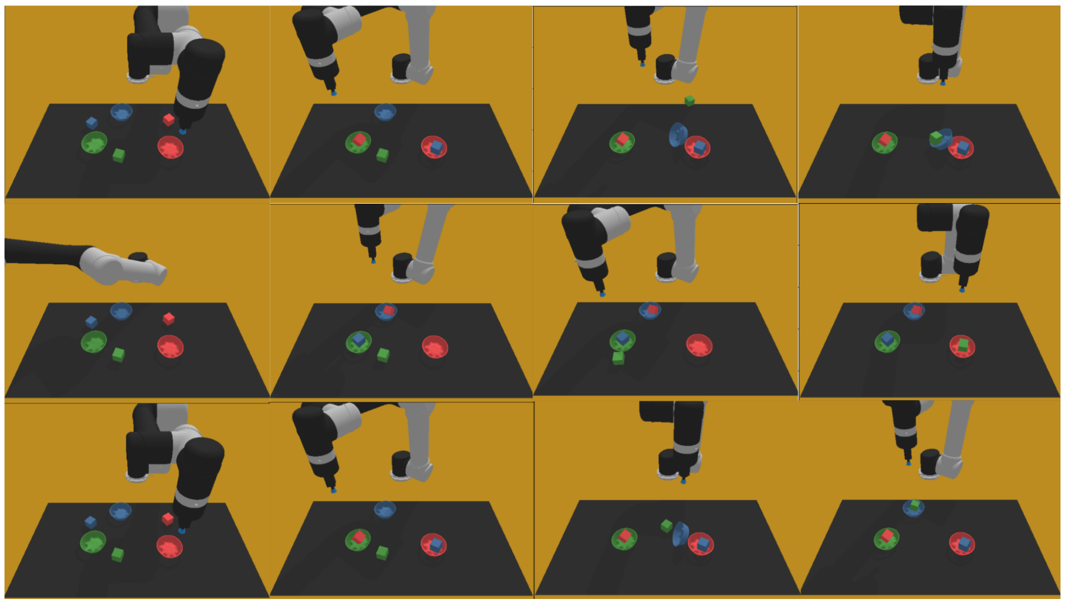


Figure 10: More examples of complete validation feedback mechanism in play recovering from unfavourable situations (Column 3).