1 APPENDIX

1.1 OVERVIEW

The Appendix contains the following content.

- **Failure Taxonomy** (Appendix 1.2): more thorough definition and figure to discussions about the different failure modes.
- **FailGen Data Generation Pipeline** (Appendix 1.3): more discussion of FailGen implementation with example configurations files.
- **AHA Datasets** (Appendix 1.4): more details on the instruction-tuning dataset and evaluation datasets.
- **Additional Experimental Results** (Appendix 1.5): more details and experiments with instruction finetuning.
- **Downstream Robotic Application: VLM Reward Generation** (Appendix 1.6): more policy rollouts, generated reward function examples, and prompts.
- **Downstream Robotic Application: VLM Task-plan Generation**(Appendix 1.7): more policy rollouts, generated task-plan examples, and prompts.
- **Downstream Robotic Application: VLM Sub-task Verification**(Appendix 1.8): more policy rollouts.

1.2 FAILURE TAXONOMY

We conducted an in-depth study of recent real-world, diverse robot datasets (such as Open-X (Padalkar et al., 2023), DROID (Khazatsky et al., 2024), and EGO4D (Grauman et al., 2022)) and the policies trained using these datasets. Through this analysis, we identified several common modes of failure, which can be categorized into seven types: incomplete grasp, inadequate grip retention, misaligned keyframe, incorrect rotation, missing rotation, wrong action sequence, and wrong target object.

Incomplete Grasp (`No_Grasp`) Failure: `No_Grasp` is an object-centric failure that occurs when the gripper reaches the desired grasp pose but fails to close before proceeding to the next keyframe.

Inadequate Grip Retention (`Slip`) Failure: `Slip` is an object-centric failure that occurs after the object has been successfully grasped. As the gripper moves the object toward the next task-specific keyframe, the grip weakens, causing the object to slip from the gripper. For generating the AHA dataset for training and evaluation, we configured a 5-timestep activation for the `Slip` failure mode, triggering the object to drop from the gripper.

Misaligned keyframe (`Translation`) Failure: This action-centric failure occurs when the gripper moves toward a task keyframe, but a translation offset along the X, Y, or Z axis causes the task to fail. For the AHA training and evaluation dataset, we introduced a translation offset of [-0.5, 0.5] meters. In the ManiSkill-Fail dataset, we applied a translation noise of [0, 0.1] meters along either the X, Y, or Z axis from the original waypoint. The translation coordinate system is depicted in Figure 3 (Left).

Incorrect Rotation (`Rotation`) Failure: `Rotation` is an action-centric failure that occurs when the gripper reaches the desired translation pose for the sub-task keyframe, but there is an offset in roll, yaw, or pitch, leading to task failure. For the AHA dataset, we set a rotation offset of [-3.14, 3.14] in radians along roll, yaw, or pitch. The rotation coordinate system is depicted in Figure 3 (Right).

Missing Rotation (`No_Rotation`) Failure: `No_Rotation` is an action-centric failure that happens when the gripper reaches the desired translation pose but fails to achieve the necessary rotation (roll, yaw, or pitch) for the sub-task, resulting in task failure.

Wrong Action Sequence (`Wrong_action`) Failure: `Wrong_action` is an action-centric failure that occurs when the robot executes actions out of order, performing an action keyframe before the correct one. For example, in the task `put_cube_in_drawer`, the robot moves the cube toward the drawer before opening it, leading to task failure.

Wrong Target Object (`Wrong_object`) Failure: `Wrong_object` is an object-centric failure that occurs when the robot acts on the wrong target object, not matching the language instruction. For

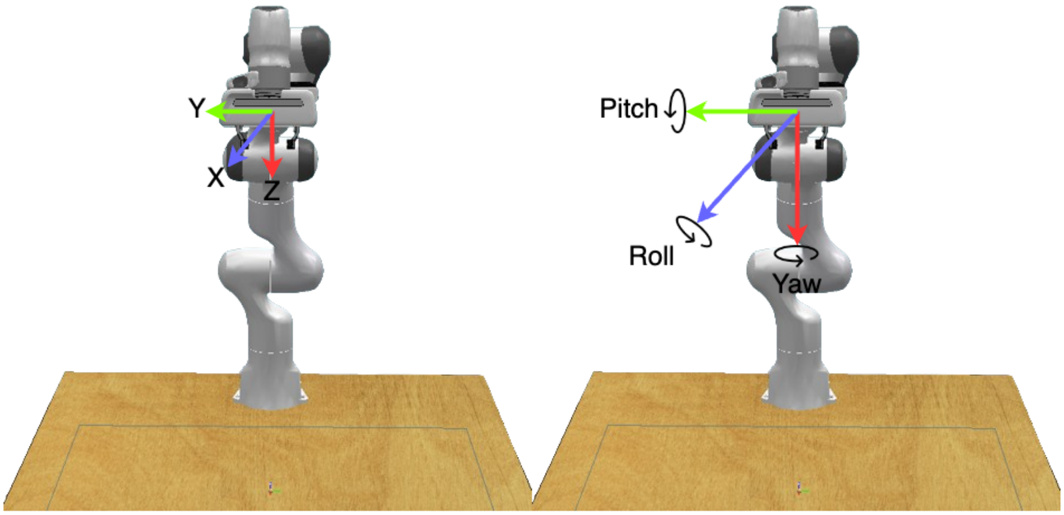


Figure 1: **Failure mode reference coordinate systems.** (Left) Translation coordinate system, and (Right) rotation coordinate system.

example, in the task `pick_the_red_cup`, the gripper picks up the green cup, causing failure. We perform a sweep through all manipulable objects, swapping them with the target object in the scene.

1.3 FailGen Data Generation Pipeline

We developed `FailGen`, an environment wrapper that can be easily integrated into any simulator. It leverages pre-defined or hand-crafted robot demonstrations for imitation learning, where each trajectory is represented as a waypoint-based system. Two consecutive waypoints form a sub-task, with each sub-task linked to a predefined set of language descriptions. `FailGen` allows for modifications to environment parameters, such as gripper end-effector translation, rotation, and open/close state. By altering these parameters, we systematically generate failures at every waypoint. However, for the 79 tasks collected from RLBench, we do not initially know which sub-task will fail due to specific failure modes. To address this, we perform a systematic sweep, using RLBench's built-in success conditions to explore all possible combinations. This generates a configuration of failures for each task, which we then use to procedurally generate all failure training data. Additionally, we manually annotate each sub-task with natural language instructions describing the task, and pair this with failure mode explanations to serve as language input for instruction-tuning. Example of the configuration files are depicted at Figure 5.

1.4 AHA Dataset

Using `FailGen`, we curated two datasets from RLBench (James et al., 2020). The first is the training dataset, AHA dataset (train), which is used for instruction-tuning AHA alongside the co-train dataset. The second is the testing dataset, AHA dataset (test), used for evaluation. AHA dataset (train) contains approximately 49k image-query pairs of failures derived from 79 tasks, while AHA dataset (test) consists of around 11k image-query pairs from 10 hold-out tasks.

1.5 Additional Experimental Results

We conducted additional experiments to better understand and visualize AHA's predictions. We trained two versions of the AHA model with 13B parameters, using different language models for fine-tuning: Llama-2-13B and Vicuna-1.5-13B. The results showed less than a 2.5% performance difference between the two models, indicating that our fine-tuning data is effective regardless of the base language model. These results are presented in Table 3. Additionally, we visualized the output

Figure 2: (Left) **Example of config file of one task for Maniskill-Fal**. (Right) **Example of config file for AHA task**

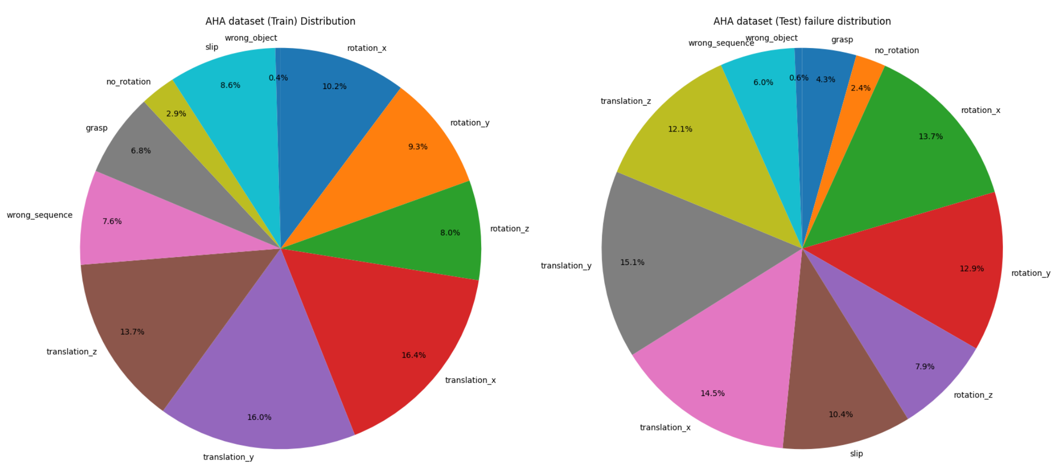


Figure 3: **Data distribution of AHA dataset for both train and test.**

predictions from various baselines compared to our model and evaluated performance across all three datasets, with the results shown in Figure 4.

AHA model performance uncertainty estimation. To evaluate the relationship between uncertainty estimation and model performance, we conducted additional experiments across various evaluation datasets. Specifically, we compared the sentence token prediction probabilities of AHA-13B with those of LLaVA v1.5-13B. AHA-13B exhibited significantly higher average prediction probabilities, reflecting its superior accuracy. These findings underscore the positive impact of fine-tuning with the AHA failure dataset on model confidence and performance as depicted in Table 2.

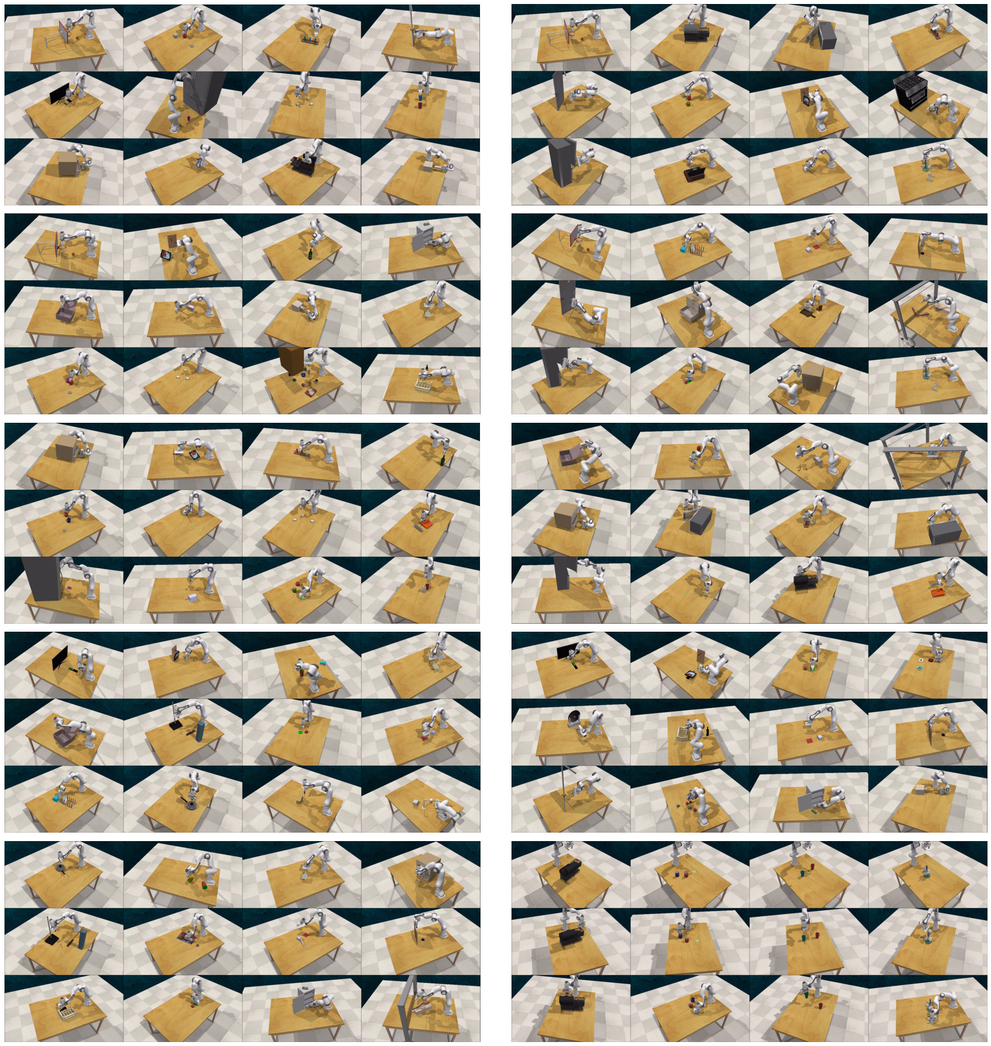


Figure 4: **Examples of different failure modes.** Row 1: `No_grasp` and `Rotation_x`. Row 2: `Rotation_y` and `Rotation_z`. Row 3: `Slip` and `Wrong_sequence`. Row 4: `Translation_x` and `Translation_y`. Row 5: `Translation_z` and `Wrong_object`.

Effects of Viewpoints on Evaluation. We evaluated the reasoning capabilities of the AHA model on the ManiSkill-Failure dataset across three different viewpoint configurations (one, two, and three viewpoints). Interestingly, we observed a slight performance advantage when using single-viewpoint images. We attribute this to the resolution limitations of the LLaVA-1.5 visual encoder (256x256), where single-viewpoint inputs offer clearer and more focused visual information for failure reasoning, as summarized in Table 1.

Model: AHA-13B (Viewpoints)	Binary Success	ROUGE-L	LLM Fuzzy Match	Cosine Similarity
One viewpoint	1.000	0.673	0.587	0.712
Two viewpoints	1.000	0.615	0.587	0.671
Three viewpoints	1.000	0.600	0.633	0.681

Table 1: Performance comparison across different numbers of viewpoints for AHA-13B

Dataset	AHA-13B (Output Probabilities / Cosine Similarity)	LLaVA-13B-v1.5 (Output Probabilities / Cosine Similarity)
AHA Dataset (Test)	0.670 / 0.583	0.066 / 0.208
Maniskill Fail	0.457 / 0.681	0.024 / 0.208
RoboFail	0.292 / 0.471	0.000 / 0.203

Table 2: Performance against model prediction sentence probabilities likelihood evaluated across datasets.

Table 3: **Ablation on Different Base LLMs for Fine-Tuning.** We fine-tuned AHA-13B using both LLaMA-2-13B and Vicuna-1.5-13B as base LLM models. The quantitative results show that the average performance difference between the two models is less than 2.5%, indicating that our failure formulation and the AHA dataset are effective regardless of the base model selection.

	AHA dataset (Test)				ManiSkill-Fail				RoboFail			
Models	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑	$\text{ROUGE}_L \uparrow$	Cos Sim ↑	BinSucc(%) ↑	Fuzzy Match ↑
AHA-13B (Llama-2)	**0.446**	0.583	0.702	**0.768**	**0.600**	**0.681**	**1.000**	0.633	0.280	**0.471**	**0.643**	0.465
AHA-13B (Vicuna-1.5)	0.458	**0.591**	**0.709**	0.695	0.574	0.657	**1.000**	**0.851**	**0.290**	0.468	**0.661**	**0.605**

1.6 VLM Reward Generation

In this section, we present reward functions generated by GPT-4o and AHA for comparison, as shown in Figure 5. Additionally, we demonstrate RL policy rollouts improved through AHA 's failure feedback across all five tasks along with all the final dense reward function modified by AHA shown in Figure 6 and 7. For all tasks, except **put_sphere_on_holder** (trained with PPO for 10M steps), PPO was trained for 25M steps prior to reflection and evaluation.

Simulation task Details We describe each of the 4 tasks in detail, along with their Maniskill variations and success condition.

1.6.1 PICKUP YCB

Filename: `pick_single_ycb.py`

Task: Pick up the single YCB object and lift it up to target height.

Success Metric: The object position is within goal_thresh (default 0.025) euclidean distance of the goal position.

1.6.2 PUSH T

Filename: `push_T.py`

Task: Push the T into the T shaped area.

Success Metric: The 3D T block covers at least 90

1.6.3 PLACE SPHERE

Filename: `place_sphere_v1.py`

Task: Pick up the sphere and place it into the bin.

Success Metric: the sphere is on top of the bin. That is, the sphere's xy-distance to the bin goes near 0, and its z-distance to the bin goes near the sphere radius + the bottom bin block's side length the object is static. That is, its linear and angular velocities are bounded with a small value the gripper is not grasping the object.

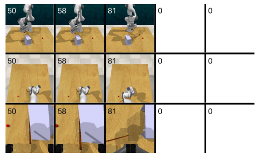
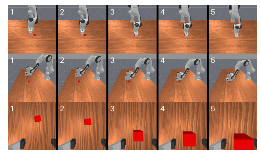
1.6.4 STACK CUBE

Filename: `stack_cube_v1.py`

Task: Pick up the red cube and stack it onto the green cube.

Success Metric: the red cube is on top of the green cube (to within half of the cube size), the red cube is static, the red cube is not being grasped by the robot (robot must let go of the cube).

Table 4: **Examples of detection and reasoning over failures by various VLMs.** We sampled one evaluation data point from each of the three evaluation datasets and compared the reasoning predictions generated by different state-of-the-art VLMs alongside our AHA-13B. Our model consistently provides the most semantically similar reasoning to the ground-truth answers for the given queries.

Evaluation Datasets	AHA dataset (Test)	Maniskill-Fail	RoboFail (Liu et al., 2023)
			
Question	At the current sub-task, the robot is picking up the hockey stick. First, determine whether it has succeeded by choosing from ["yes", "no"]. If not, explain why it failed?	At the current sub-task, the robot is picking up the red cube. First, determine whether it has succeeded by choosing from ["yes", "no"]. If not, explain why it failed?	At the current sub-task, the robot is placing the apple into the bowl. First, determine whether it has succeeded by choosing from ["yes", "no"]. If not, explain why it failed?
LLaVA 1.5-13B	The image displays a time-lapse sequence of a robotic arm grabbing a hockey stick.	Based on the image, it depicts a different sub-task of a robot grasping a red block.	For the pick-up task, i would say the robot arm has succeeded in picking.
GPT4o	No, the robot arm fails to hold the hockey stick securely.	Yes, the robot arm has successfully picked up the red block.	No, the robot gripper failed to pick up the bowl in the second frame.
Gemini 1.5 Flash	Yes.	Yes, Yes, Yes.	Frame 1, no the gripper is not grasping the bowl and move it.
AHA-13B (Ours)	No, the robot slip the object out of its gripper.	No, the robot gripper fails to close the gripper.	No, this is not the right action sequence for the task.
Ground-truth Answer	No, the robot slip the object out of its gripper.	No,the robot gripper fails to close.	No, this is not the right action sequence for the task.

1.6.5 OPEN DRAWER

Filename: `open_cabinet_drawer_v1.py`

Task: Pull open the drawer.

Success Metric: The drawer is open at least 90% of the way, and the angular/linear velocities of the drawer link are small.

Figure 5: **(Left)** Example of improved dense reward function using GPT-4o for reflection. **(Right)** Example of improved dense reward function using AHA for reflection

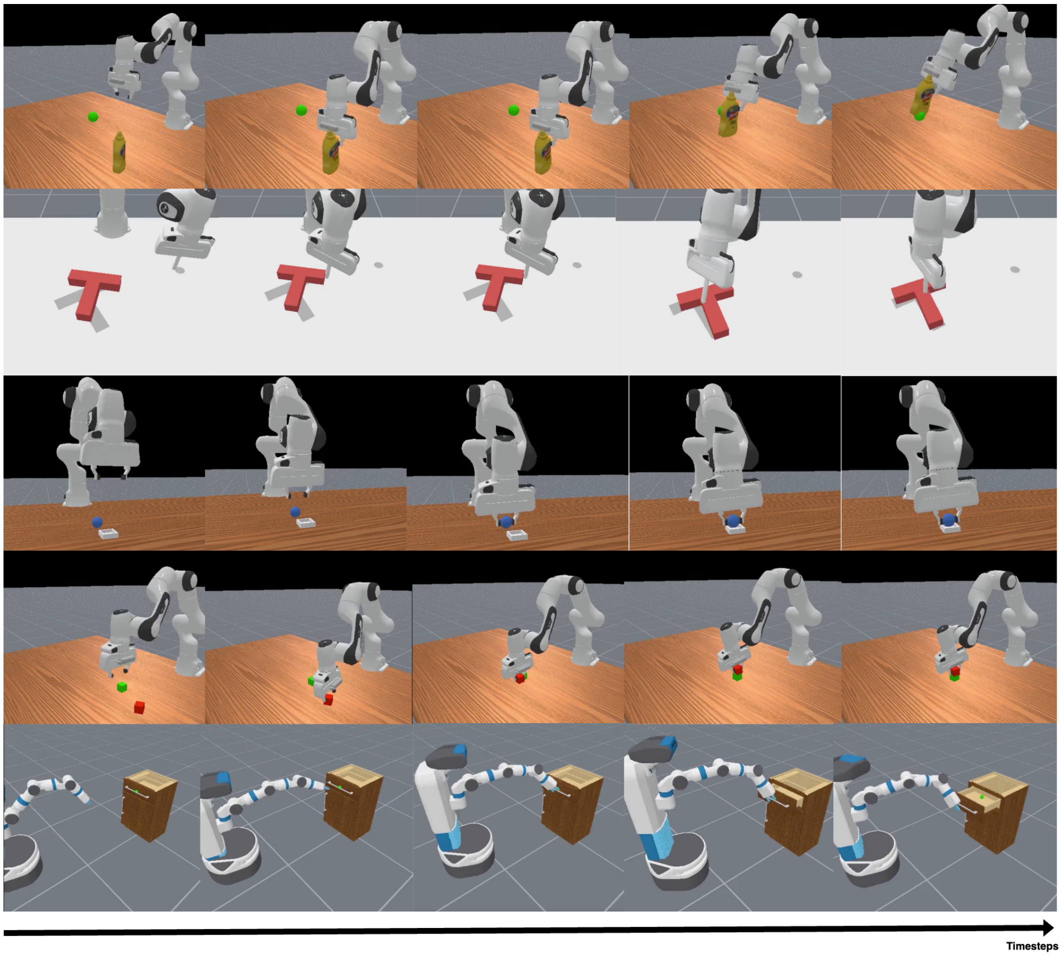


Figure 6: **RL policy roll-outs via improved with AHA.** Row 1: `pickup_YCB`. Row 2: `push_T`. Row 3: `Place_sphere`. Row 4: `stack_cube`. Row 5: `open_drawer`

1.7 VLM Task-plan Generation

In this section, we present the policy rollouts improved by AHA in Figure 8, along with the modified task plans in Figure 9.

Simulation task Details We describe each of the 3 tasks in detail, along with their PyBullet variations and success condition.

1.7.1 PUT BANANA CENTRE

Filename: `ours_raven_ycb_pick.py`

Task: Pick up the banana and place it onto the centre of the table.

Success Metric: The success condition on the final location of the banana with respect to the table area.

1.7.2 STACK BANANA

Filename: `ours_ycb_banana_spam_stack.py`

Task: Pick up the banana and place it onto the spam can.

Success Metric: The position of the banana should be on the spam can, and rest stably.

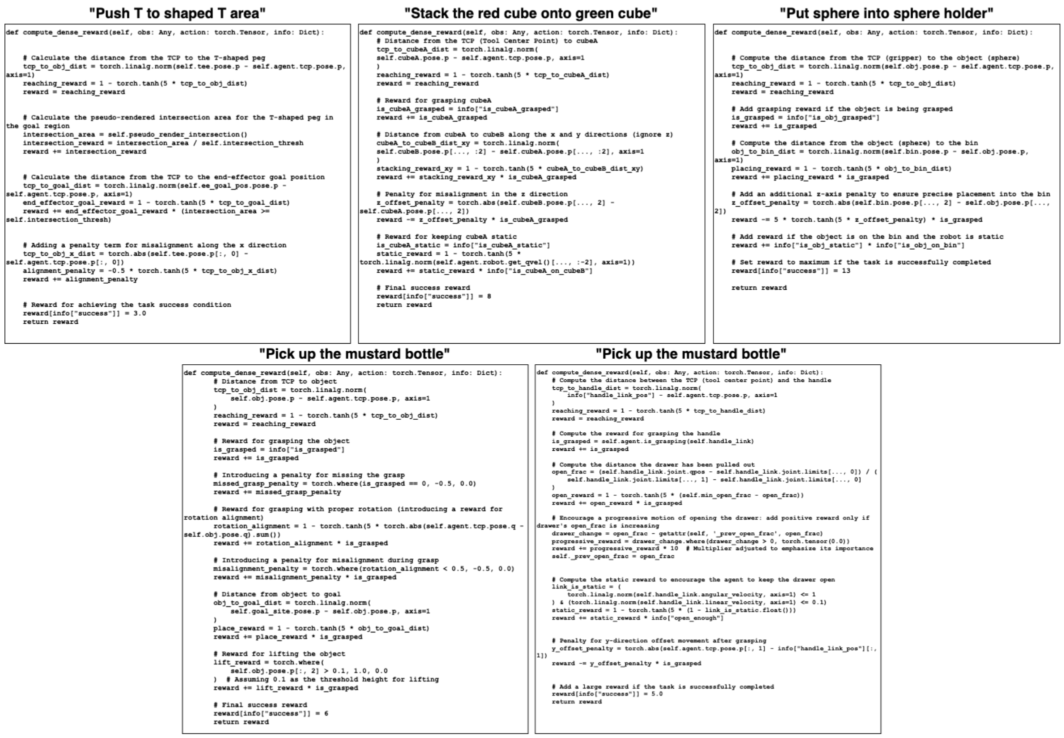


Figure 7: **Examples of modified reward function via AHA**

1.7.3 STACKS CUBES

Filename: `ours_raven_bowl_stack.py`

Task: Pick up the green cube and place into the green bowl, and then take the yellow cube and stack it on top of the green.

Success Metric: When the yellow cube is stably stack on top of the green in the green bowl.

1.8 VLM SUB-TASK VERIFICATION

In this section, we leverage `Manipulate-Anything` (Duan et al., 2024) as the main policy framework, integrating it with AHA. AHA functions as a sub-task verifier VLM, playing a crucial role in ensuring task success when using `Manipulate-Anything`. Examples of the roll-outs are shown in Figure 10.

Simulation task Details We describe each of the 4 tasks in detail, along with their RLBench variations and success condition.

1.8.1 PUT BLOCK

Filename: `put_block.py`

Task: Pick up the green block and place it on the red mat.

Success Metric: The success condition on the red mat detects the target green block.

1.8.2 PICKUP CUP

Filename: `pickup_cup.py`

Task: Pick up the red cup.

Success Metric: Lift up the red cup above the pre-defined location.

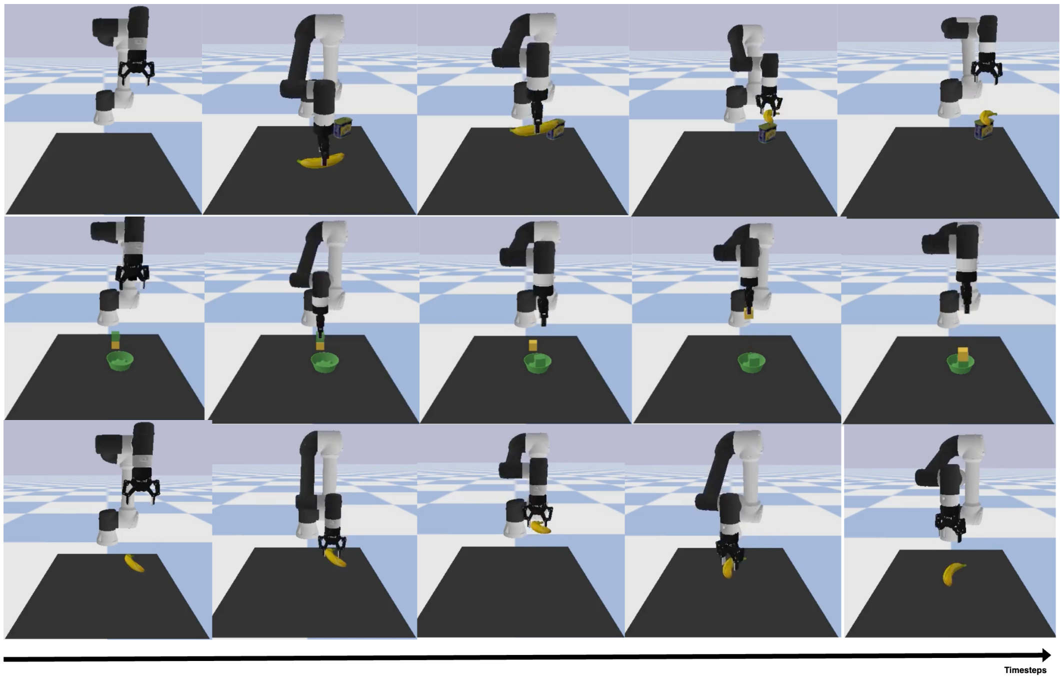


Figure 8: **TAMP policy roll-outs via improved with AHA.** Row 1: `put_banana_centre`. Row 2: `stack_banana`. Row 3: `stack_cubes`

1.8.3 SORT MUSTARD

Filename: `sort_mustard.py`

Task: Pick up the yellow mustard bottle, and place it into the red container.

Success Metric: The yellow mustard bottle inside red container.

1.8.4 PICK & LIFT

Filename: `pick_and_lift.py`

Task: Pick up the red cube.

Success Metric: The red cube is lifted up.

1.9 LIMITATIONS AND OPPORTUNITIES

While AHA is quite capable, it is not without limitations. In the following sections, we discuss some of these limitations and potential risks for real-world deployment.

Does not cover all failure cases. While AHA is instruction-tuned to address the seven most common failure modes in robotic tabletop manipulation, and provides fine-grained failure identification down to axis-level translation or orientation offsets of the gripper, it is inevitable that certain edge cases remain uncovered. These edge cases fall outside the distribution of failure modes considered in our approach and may occur in real-world scenarios. To address this limitation, future work could focus on distilling policies already trained with action-based "artifacts" to generate naturally occurring failures. These could then be labeled using human annotations or a hybrid approach combining AHA and a GPT-4o-based labeling system. Such efforts could enhance the robustness and applicability of failure detection across a broader range of scenarios.

Improving policies directly with AHA Currently, we have demonstrated AHA's effectiveness in enhancing existing robotics systems that utilize off-the-shelf VLMs/LLMs to automate various pipeline components. AHA has shown significant impact in improving code generation, task-plan

"Stack banana onto spam can"

````text
Thank you for the clarification. Given that the robot gripper moves to the desired position with an offset along the z direction,
we need to account for this offset when placing the banana on top of the potted meat can.
Let's assume the offset is a known constant, `GRIPPER_OFFSET_Z`. We'll use this offset to adjust the placement pose accordingly.
Here's the updated `gen_plan` and `gen_domain` functions:


```python
GRIPPER_OFFSET_Z = 0.02  # Example offset value, adjust as necessary


def gen_plan(initial: RavenBelief, grasp: RavenGrasp):
    plan = []

    # Pick up the banana
    plan.append(Action("pick", ["object_1", grasp]))

    # Get the pose of the potted meat can
    spam_pose = initial.objects["object_0"].pose

    # Calculate the placement pose for the banana on top of the potted meat can
    banana_place_pose = RavenPose(
        x=spam_pose.x,
        y=spam_pose.y,
        z=spam_pose.z + BLOCK_SIZE + GRIPPER_OFFSET_Z,  # Adjust for the gripper offset
        roll=spam_pose.roll,
        pitch=spam_pose.pitch,
        yaw=spam_pose.yaw
    )

    # Place the banana on top of the potted meat can
    plan.append(Action("place", ["object_1", grasp, banana_place_pose]))

    return plan


def gen_domain(initial: RavenBelief):
    return {
        "grasp": GraspSampler()
    }
```


### Explanation


1. **Picking the Banana**: The plan starts by picking up the banana (`object_1`) using a grasp sampled from the `GraspSampler`.
2. **Calculating Placement Pose**: The pose of the potted meat can (`object_0`) is retrieved from the initial state. The placement
pose for the banana is calculated by adding the height of the banana (approximated as `BLOCK_SIZE`) and the gripper offset
(`GRIPPER_OFFSET_Z`) to the z-coordinate of the potted meat can's pose. The roll, pitch, and yaw are kept the same to ensure proper
alignment.
3. **Placing the Banana**: The plan then places the banana at the calculated pose on top of the potted meat can.


### Why This Will Always Satisfy the Goal


- **Grasp Sampling**: The grasp is sampled from a `GraspSampler`, ensuring that a valid grasp is used for picking up the banana.
- **Placement Calculation**: The placement pose is calculated based on the current pose of the potted meat can and adjusted for the
gripper offset, ensuring that the banana is placed directly on top of it.
- **Generalization**: The plan is general enough to handle different initial positions and orientations of the objects, as it
dynamically calculates the placement pose based on the initial state.


This approach ensures that the banana will always be stacked on top of the potted meat can, accounting for the gripper offset along
the z direction.
````

"Stack two cubes into a bowl"

````text
Thank you for pointing that out. Let's correct the plan to account for the z-offset when placing the second cube.
```python
def gen_plan(initial:RavenBelief, x_offset:float, y_offset:float, z_offset:float):
    plan = []


    # Pick up the green block
    green_block = initial.objects["object_6"]
    plan.append(Action("pick", green_block.pose.point))


    # Place the green block into the green bowl
    green_bowl = initial.objects["object_5"]
    x, y, z = green_bowl.pose.point
    plan.append(Action("place", [x + x_offset, y + y_offset, z + z_offset]))


    # Pick up the yellow block
    yellow_block = initial.objects["object_4"]
    plan.append(Action("pick", yellow_block.pose.point))


    # Stack the yellow block onto the green block
    green_block_new_pose = [x + x_offset, y + y_offset, z + z_offset + BLOCK_SIZE]
    plan.append(Action("place", [green_block_new_pose[0], green_block_new_pose[1], green_block_new_pose[2] + BLOCK_SIZE]))


    return plan

def gen_domain(initial:RavenBelief):
    return {
        "x_offset": ContinuousSampler(-0.02, 0.02),
        "y_offset": ContinuousSampler(-0.02, 0.02),
        "z_offset": ContinuousSampler(0.01, 0.03),
    }
```
### Explanation

1. **Pick up the green block**: The plan starts by picking up the green block (`object_6`).
2. **Place the green block into the green bowl**: The green block is then placed into the green bowl (`object_5`). The offsets
(`x_offset`, `y_offset`, `z_offset`) allow for some variability in the exact placement within the bowl.
3. **Pick up the yellow block**: The plan continues by picking up the yellow block (`object_4`).
4. **Stack the yellow block onto the green block**: Finally, the yellow block is placed on top of the green block, which is now in
the bowl. The height is adjusted by adding the `BLOCK_SIZE` to the z-coordinate of the green block's new position, ensuring the
yellow block is stacked correctly.


### Why the Function Will Always Satisfy the Goal

- The `x_offset`, `y_offset`, and `z_offset` parameters allow for slight variations in the placement of the green block within the
bowl, ensuring it fits regardless of minor positional differences.
- The yellow block is always placed directly on top of the green block, ensuring a stack is formed.
- The domain for the offsets is chosen to be small enough to ensure the blocks are placed correctly but large enough to allow for
some variability.


### Constraints and Potential Failures

- If the offsets are too large, the green block might not be placed correctly within the bowl, or the yellow block might not stack
properly on the green block.
- The domain for the offsets is chosen to minimize these risks, but if a failure occurs, the offsets can be adjusted to smaller
ranges.

This plan should achieve the goal of placing the green block into the bowl and then stacking the yellow block onto the green block,
given the initial state and the defined domain.
````

"Move banana to centre"

````text
To account for the slight offset along the y direction when grasping the object, we can adjust the placement position accordingly.
We will add an additional parameter to the `gen_plan` function to specify the y offset and use it to correct the placement
position.


Here's the updated implementation:


```python
def gen_plan(initial: RavenBelief, grasp: RavenGrasp, y_offset: float):
    plan = []
    object_name = "object_0"  # The banana is identified as "object_0"
    plan.append(Action("pick", [object_name, grasp]))
    # Adjust the y position by the offset when placing the object
    adjusted_y = TABLE_CENTER[1] + y_offset
    plan.append(Action("place", [object_name, grasp, RavenPose(x=TABLE_CENTER[0], y=adjusted_y, z=0.02)]))
    return plan


def gen_domain(initial: RavenBelief):
    return {
        "grasp": GraspSampler(),
        "y_offset": ContinuousSampler(-0.05, 0.05)  # Allowing for a small offset range
    }
```


### Explanation:
1. **gen_plan Function**:
   - The function now takes an additional parameter `y_offset` to account for the slight offset along the y direction.
   - When placing the object, the y position is adjusted by adding the `y_offset` to the table center's y coordinate.


2. **gen_domain Function**:
   - The function returns a dictionary with two entries:
     - `grasp`: Sampled using the `GraspSampler`.
     - `y_offset`: Sampled using a `ContinuousSampler` with a range of -0.05 to 0.05, allowing for a small adjustment in the y
direction.


### Why This Will Always Satisfy the Goal:
- The plan still explicitly picks up the banana and places it at the center of the table, but now with an adjustment for the y
offset.
- The `y_offset` parameter allows for correcting the placement position to account for the slight offset during grasping.
- The table center coordinates are fixed and known, ensuring the banana is always moved to the correct location with the necessary
adjustment.


This approach ensures that the goal is achieved regardless of the specific grasp sampled and the slight offset during grasping, as
long as the offset is within the specified range.
````

Figure 9: **Examples of modified task-plan via AHA**

generation, and sub-task success detection. However, AHA is not yet capable of directly influencing low-level trained policies through failure language reasoning. A potential next step would involve training low-level policies with language-conditioned demonstrations covering a diverse range of corrective actions tied to various failure modes. This would enable low-level policies to interpret failure reasoning in a counterfactual manner and generate corrective actions directly.

References

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*, 2024.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18995–19012, 2022.

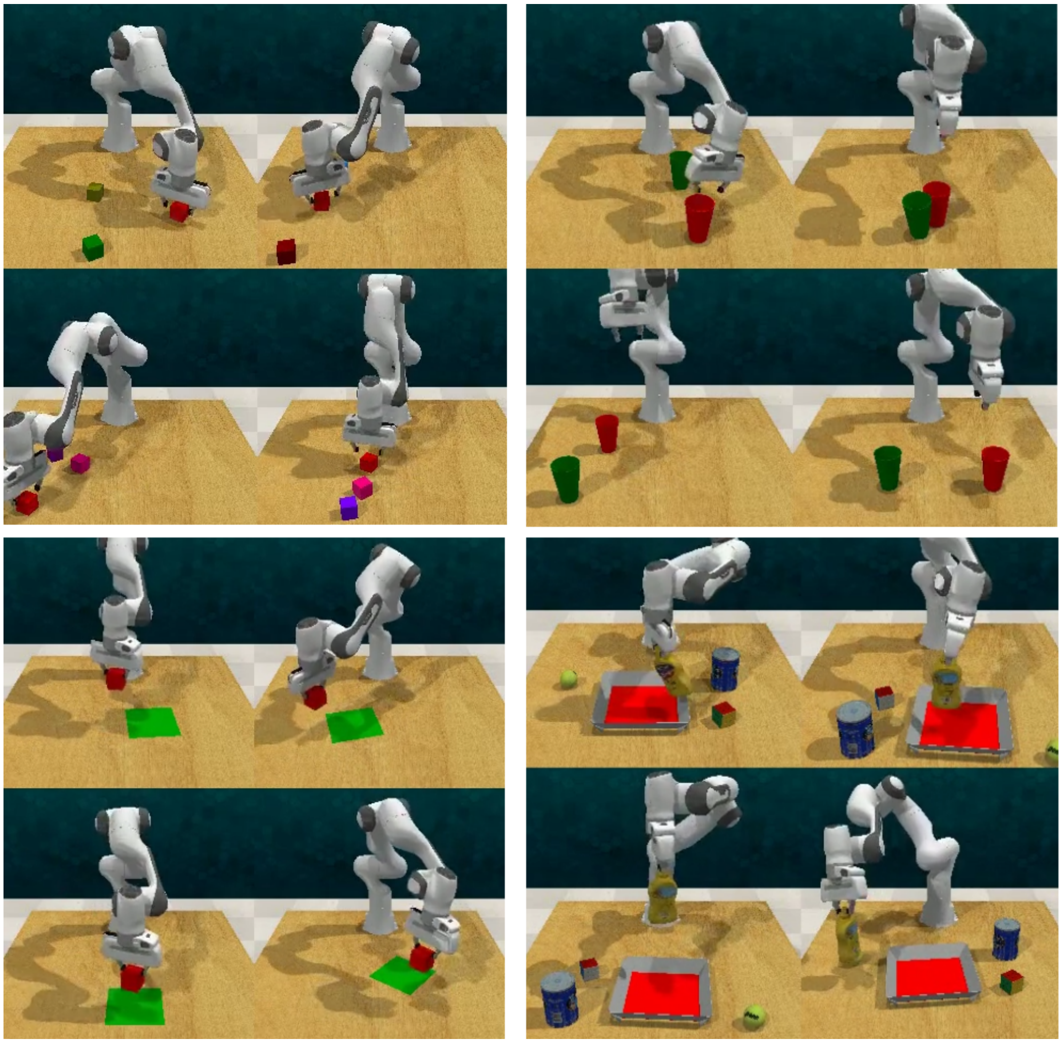


Figure 10: **Examples of zero-shot data generator trajectories with AHA as sub-tasks verifier.** Row 1: `pickup_cube`, `pickup_cup`. Row 2: `put_block`, `sort_mustard`

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023.

Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.