Think Twice, Act Once: Verifier-Guided Action Selection For Embodied Agents

Supplementary Material

First, we provide additional details about the benchmarks (App 8) and the training setup (App 9). Then, we provide the prompts used for synthetic data generation in App 10. Further, qualitative examples of synthetic mistakes and verifications are available in App 11. Finally, we provide some qualitative examples of the outputs generated by our verifiers at test time in App 12.

8. Details about Benchmarks

8.1. LangR [43]

The LangR benchmark [43], built on the Habitat 2.0 [42] simulator, is designed to evaluate the generalization capability of embodied agents in household rearrangement scenarios. Agents receive high-level instructions and must execute tasks that involve manipulating objects (pick, open, place), searching for target items, and performing simple forms of logical reasoning such as conditional operations. The allowed actions are: `navigate(receptacle)`, `pick(object)`, `place(object)`, `open(receptacle)`, `close(receptacle)`.

The actions are executed only if certain preconditions are satisfied. For example, an object can be picked only if it is within reach. The object categories used are: ball, clamp, hammer, screwdriver, padlock, scissors, block, drill, spatula, knife, spoon, plate, sponge, cleanser, plum, pear, peach, apple, lemon, can, box, banana, strawberry, lego, rubriks cube, book, bowl, cup, fork. The maximum number of steps for an episode is 32.

The benchmark provides a suite of training tasks along with a separate set of held-out test tasks. The test split is constructed to examine multiple aspects of generalization, including previously unseen environments and novel instruction formulations.

Two main forms of generalization are emphasized. The first is **paraphrastic robustness**, where the agent must interpret varied rephrasings of an instruction that shares the same underlying goal. The second is **behavioral generalization**, which requires the agent to handle new types of reasoning that do not appear in the training distribution. For example, during training the agent may learn to locate a specified number of object instances, while the multiple rearrangements task in the test set requires discovering all instances of a category without being told the exact count. We describe the tasks below.

Paraphrastic Robustness

- **Instruction Rephrasing:** Same underlying goal expressed using a different wording than in training.
- **Referring Expressions:** Objects are mentioned through descriptive or visual attributes rather than their canonical names (e.g., a banana described as a curved yellow fruit).
- **Context:** Objects are referred to within a situational or contextual description (e.g., a ball described as a sports object).
- **Irrelevant Instruction Text:** Additional text is included that does not affect the task but may distract the agent.

Behavioral Generalization

- **Multiple Rearrangements:** Requires rearranging three objects, although training tasks involve only two.
- **Novel Objects:** Introduces new combinations of instructions and object categories that never co-occur in training.
- **Multiple Objects:** Requires manipulating all instances of an object category. The agent must search for and detect every instance, a concept not present in training.
- **Conditional Instructions:** Task outcome depends on whether a specified condition holds (e.g., if the fridge is open, move the apple to it; otherwise move the orange, and only the required object).

The benchmark also includes a spatial reasoning task that uses instructions such as "place the object to the right of the black table." The action space available to the agent does not provide primitives for lateral (left, right, forward, back) movement, which prevents the agent from acquiring meaningful knowledge of the scene layout. As a result, the task is not compatible with the defined action space, so we exclude it from our evaluation.

8.2. ALFRED [33]

The ALFRED [33] benchmark is built on top of the AI2THOR simulator [16]. In this work, we use the EB-ALFRED implementation from [49], which restructures the original tasks into categories designed to probe different aspects of out-of-distribution generalization. The benchmark includes seven task types: *Pick & Place*, *Stack & Place*, *Pick Two & Place*, *Clean & Place*, *Heat & Place*, *Cool & Place*, and *Examine in Light*. Agents operate using eight possible actions: `pick up`, `open`, `close`, `turn on`, `turn off`, `slice`, `put down`, and `find`.

EB-ALFRED tasks are grouped into six subsets, each targeting a distinct skill or reasoning capability:

- **Base:** Evaluates core task solving abilities needed to plan and execute low to medium complexity action sequences.
- **Common Sense:** Measures the use of indirect object references grounded in everyday knowledge (for example, describing a refrigerator as "a receptacle that can keep food fresh for several days") and tests the agent's ability to apply such knowledge during instruction following.

- **Complex Instruction:** Contains longer contexts with both relevant and irrelevant details, assessing an agent's ability to extract the intended instruction.
- **Spatial Awareness:** Refers to objects through spatial relations with other items, testing spatial grounding and relational reasoning.
- **Visual Appearance:** Requires identifying objects based on visual attributes such as color or shape.
- **Long Horizon:** Includes tasks requiring extended action sequences, typically more than 15 steps in EB-ALFRED.

To construct the benchmark, [49] used the *valid seen* split of ALFRED. A set of 50 tasks with fewer than 15 steps was first selected, from which the common sense and complex instruction subsets were derived. Another 50 tasks with more than 15 steps formed the long horizon subset. Instances for the visual appearance and spatial awareness subsets were chosen directly from ALFRED based on language references to color, shape, or spatial relations. In total, EB-ALFRED contains 300 test instances, uniformly distributed across the six subsets (50 per subset).

9. Training Details

We train both the policy and verifier using the `LLaMAFactory` framework [55]. Full finetuning is applied while keeping the vision encoder and projection module fixed. All training runs are conducted on 8×NVIDIA L40 GPUs. Training data is formatted as multi turn dialogues using the `sharegpt` format provided by `LLaMAFactory`.

The inputs to the policy and verifier consist of prior images and actions rather than earlier chains of thought. To generate data compatible with this interface, each trajectory from the dataset is decomposed into a set of sub conversations, one for each action step. Consider an original trajectory of the form

$$I,\ o_1,\ (c_1, a_1),\ o_2,\ (c_2, a_2),\ \ldots$$

where I is the instruction, o_i are observations, c_i are chains of thought, and a_i are actions. The ith sub conversation contains the instruction along with all observations and executed actions up to step i:

$$I,\ o_1,\ a_1,\ o_2,\ a_2,\ \ldots,\ o_i,\ (c_i, a_i).$$

All chains of thought except the final one, c_i, are removed. During training we compute loss only on the last assistant message of each sub conversation, implemented by setting `mask_history` to `True`. Hyperparameters are provided in Table 6.

Table 6. Hyperparameters used for training

Hyperparameter	Value
Number of GPUs	8
per_device_train_batch_size	2
gradient_accumulation_steps	4
effective batch size	8 * 2 * 4 = 64
learning_rate	1e-5
num_train_epochs	3
warmup_ratio	0.1
bf16	True
lr_scheduler_type	cosine

10. Prompts

10.1. Prompt for CoT data generation

Prompt to Generate Synthetic Chain-of-Thought (adapted and modified from [52])

```
You're an expert reinforcement learning researcher. You've trained a policy for controlling a robot with an arm to move around and perform tasks in a household environment.
The robot successfully completed a task specified by the instruction: "{instruction}". For that purpose, the
robot executed a sequence of actions. Consecutive moves that were executed are the following:

insert original trajectory with instruction, images, actions


The attached images show what the robot was seeing at a given time step (corresponding to every <image> tag in the trajectory). You can use the images to understand what the robot was seeing, where it was, what it was holding, whether its actions were successful or not, etc.


## General Information and Instructions

1. possible actions are: pick(object), place_on_recep(receptacle), navigate(receptacle), open_fridge(), close_fridge(), open_cabinet(cabinet).
2. Possible objects are: ball, clamp, hammer, screwdriver, padlock, scissors, block, drill, spatula, knife, spoon, plate, sponse, cleanser, plum, pear, peach, apple, lemon, can, box, banana, strawberry, lego, rubrik's cube, book, bowl, cup, mug, orange, lid, toy airplane, wrench.
3. When exploring, select from possible receptacles: [cabinet, drawer 7, cabinet drawer 6, fridge, chair, black table, brown table, TV stand, sink, right counter, left counter]

## Your objective

I want you to annotate the given trajectory with reasoning. That is, for each step, I need to know not only which action should be chosen,
but importantly what reasoning justifies that action choice. I want you to be descriptive and include all the relevant information available.
The reasoning should include the task to complete, the remaining high-level steps, the high-level movements that should be executed and why they are required and any other relevant justification.


### Begin by describing the task

Start by giving an overview of the task. Make it more comprehensive than the simple instruction. Include the activity,
the objects the robot interacts with, and their relative locations in the environment. Then, describe
the high-level movements that were most likely executed, based on the task that was completed and the
primitive movements that were executed. Also, for each high-level movement write a justification for why it should be executed.
```

```
Write an answer for this part using markdown and natural language. Be descriptive and highlight all the relevant details, but ensure that your description is consistent with the trajectory that was  executed, specified above.

### List the reasonings for each step

Finally, for each step describe the reasoning that allows to determine the correct action. For each step describe the remaining part of the objective,
the current progress, the objects that are still relevant for determining the plan, and the plan for the next steps, based on the available features. Start the reasoning from a high level and gradually add finer features. I need you to be descriptive and very precise. Ensure that the reasoning is consistent with the task and the executed trajectory.
Write the answer for this part as a Python-executable dictionary. For every step in the initial trajectory there should be exactly one separate item of the form <step id>:<reasoning>.
Do not group the answers. The final dictionary should have exactly the same set of integer keys as the assistant messages in the original_trajectory above.
The reasoning should be a single string that describes the reasoning in natural language and includes all the required features.

Each reasoning string should have the following form:
- Describe the full task that remains to be completed (but only describe what remains), and place it inside a tag <task>.
- Describe the complete high-level plan for completing the remaining task (the list of remaining high-level steps), and place it inside a tag <plan>.
- Describe why the chosen high-level step should be executed now, which features of the current environment influence that decision, and how it should be done. Place it within a tag <subtask_reason>. First provide appropriate reasoning, and then the name of the action chosen (and not, "I choose action XYZ because ..."; rather, "I need to do XYZ, hence I should choose this action")
- Describe the high-level step that should be executed now (chosen from the list of high-level steps), and place it inside a tag <subtask>.



## Reminders

Keep in mind that the robot might not know where a certain object is, and it may have to move around the environment to search for it. Searching might be one of your subtasks.

Also, it's possible that the actions chosen by the robot are sometimes suboptimal, you should take that into account.
Your output reasoning chains should be from the perspective of the assistant, about how it should reason before picking the next action.
Note that the small pebble like thing near the gripper is part of the gripper itself, not a separate item.
The reasoning should be such that first it states why a certain action should be chosen, and then specify that action ("I need to find XYZ, hence, I should do action A" and not "I should do action A. It will help me find XYZ").
Sometimes, the robot would select non-sensical actions, you don't necessarily have to justify such actions (for example by hallucinating objects that don't exist).
You can use information from all time steps to develop your plan, but your final output should be such that the output at time t doesn't assume information about the future
```

```
     (because the robot will think and act one step at a time).
Also, in the original trajectory, after every assistant action, there is a flag telling whether that action was successful or not. You can use this to inform your reasoning chain output. However, note that during deployment, the agent will not have access to this information directly. Instead, it must use its visual input to decide whether the action succeeded. Hence, in your output reasoning chains, when you say that a previous action was successful/failed, you must say that this is based on the image (you can use the ground truth success status, but don't say it in the output).
In the output reasoning trajectory, it is okay if the robot changes the original plan/subtasks as it obtains more information about the environment, or after failures, similar to how an intelligent agent would adapt as it works toward solving a task. The robot should also be capable of recovering from previous errors. Such changes should be clearly stated in the reasoning string.

## Task summary

Here is a breakdown of what needs to be done:

- Describe the task.
- Describe the high-level movements that were executed, based on the completed task and the listed features.
- Describe the plan for the solution that allowed the robot to complete the task successfully.
- For each step on the trajectory, describe the reasoning that leads to determining the correct action. The reasoning should be descriptive and precise. You should provide exactly one reasoning
string for each ASSISTANT step on the trajectory specified by original_trajectory.
- Just return the dictionary directly, like:
'''python
{{
  ...
}}
'''
- At the very end of the response, write a single label FINISHED to indicate that the answer is complete.
```

10.2. Prompts for Synthetic Failed Trajectory Generation

Prompt to Generate Synthetic Mistakes and Verifications

```
Given the above conversation, your job is to create a new trajectory where the agent makes a mistake. Don't change the instruction, just imagine some mistakes and insert them into the new trajectory.

Your mistakes should be realistic and plausible, i.e., something that the agent is actually likely to do, and not a random mistake. For example:
- The agent misunderstands the instruction. it interacts with the incorrect object, or with incorrect receptacles, or doesn't perform the expected operations on the objects/receptacles. Here, try to make the "incorrect" actions still somewhat close to the correct actions, e.g., replacing apple with orange instead of remote, or replacing turn on with turn another related action instead of put down the object.
- The agent doesn't complete a precondition for the next action, e.g., it does not 'find
```

```
    ' the object before interacting with it, or doesn't 'open' the microwave or cabinet
    before putting the object inside. this one is quite important -- basically, think of
     ways in which the agent might be incompetent: even if it understood the task
    correctly, it might fail to perform it properly

- The agent messes up the order in which actions should be done. for example, if it has
    to clean something, it puts the object in the sink, takes it out, and then turns on
    the faucet

- The agent gets confused between different instances of the same object

- The agent completes only one part of the instruction and not the full instruction

- The agent doesn't do all the actions required for a subgoal, for example, it cleans an
     object in the sink, but does not pick it up before putting it somewhere else.


Keep in mind that the original trajectory is successful, so if you insert a mistake
    there, your action should be different from the action given in the original
    trajectory, because otherwise that action would be correct.

The output should also be in a similar json format. additionally, for every action in
    the output, add a judgement on whether the action is correct or incorrect. first,
    provide detailed reasoning, and then the verdict ("action_is_correct: yes or no").
    the verdict and judgement should be a single string and should go in a separate key
    in the dict called 'verification'. also, it's possible that some actions are correct
     and others are incorrect in the same conversation.

Also, per conversation, add a mistake_description field, giving a summary of what the
    mistake was.
Make sure that in the output, before every assistant message there is a user message.
for example:

<new_trajectory>
mistake_summary: <insert mistake summary>

{
"content": "...",
"role": "user",
},
{
"content": "...",
"role": "assistant",
"is_action_successful": true,
"verification": "<insert reasoning here>\n
is_action_correct: X" where X is yes or no,
},


**IMPORTANT**:
- Judge every action on whether it advances the task or not. so it is possible that the
    reasoning for the action is incorrect, but it still advances the task, in that case,
     the action is correct.
- The mistakes should start from the task interpretation, planning, and subtask
    reasoning itself, and not like it understands the task but the subtask/action are
```

```
    not aligned with the reasoning.
- Keep in mind that the output conversation is from the perspective of the agent, and
    the agent doesn't know it's making a mistake. Make sure that in the output the agent
     should not seem to know that its plans and actions are not aligned with the
    instruction. The agent should think its doing the right thing, so it doesn't even
    know that it misunderstood the instruction. so there should not be anything like "
    doing X even though the instruction said Y" or "mistakenly doing X".
- Try to make a 'realistic' mistake - go for an object that is somewhat related to the
    actual object mentioned in the instruction.
- In the verification, try to explain your reasoning in as much detail as possible,
    writing out which object was targeted and which should have been targeted instead.
- Try to generate diverse types of mistakes. You don't have to restrict yourself to the
    mistakes mentioned above, feel free to come up with other realistic failure modes.
- Keep in mind that the original trajectory is successful, so if you insert a mistake
    there, your action should be different from the action given in the original
    trajectory, because otherwise that action would be correct.
- Simply using a different object/receptacle/action might not be considered a mistake.
    For example, if a lamp is required, both floorlamp and desklamp are correct.
- Make sure that every assistant message has all the relevant tags (<task>, <plan>, <
    subtask>, etc., and especially the <action> tag)
- In the verification, first try to reason about what the instruction specifies and the
    action taken by the agent, how the two are related, what the agent should have done,
     and describe in detail why the action is incorrect.
- Every verification will be used independently of each other, so every verification
    should be complete in itself (shouldn't assume existence of previous verifications).

Double check for these important points before writing out the output conversation. It
    is crucial that these instructions are followed. So check rigorously and strictly
    your output, and revise them to follow these instructions if necessary.
```

11. Examples of Synthetic Incorrect Actions

Here, we show some examples of trajectories with synthetic mistakes generated by the LLM to train the verifier.

11.1. LangR

Example 1 (LangR): The agent erroneously picks up the bowl even though the instruction specifies a cleaner.

```
Instruction: Shift a cleanser and place it into the specified black table.


action 1: "<task>Move the bowl over to the black table</task>\n<plan>1. Go to right
    kitchen counter where the bowl usually sits.\n2. Pick up the bowl.\n3. Navigate to
    the black table.\n4. Place the bowl on the black table.</plan>\n<subtask>Navigate to
     the right kitchen counter.</subtask>\n<subtask_reason>The bowl is often left on the
     right side of the kitchen counter, so I will head there first.</subtask_reason>\n<
    action>('nav(receptacle_aabb_counter_right_kitchen_counter)', 10)</action>"

verification: "The instruction asks for relocating a cleanser, but the assistant plans
    to move a bowl instead. Nevertheless, navigating to a kitchen counter could
    logically be a first step for finding a cleanser as well, so this motion does not
    directly violate task progress.\naction_is_correct: yes",
```

```
=====

action 2: "<task>Move the bowl over to the black table</task>\n<plan>[ Pick bowl --> Go
    to black table --> Place bowl ]</plan>\n<subtask>Pick up the bowl.</subtask>\n<
    subtask_reason>I can see the bowl on the counter and will grab it.</subtask_reason>\
    n<action>('pick_bowl(robot_0)', 42)</action>",

verification: "The picked object is a bowl, not the requested cleanser, so this action
    does not advance the actual task.\naction_is_correct: no"

<following actions omitted for clarity>
```

Example 2 (LangR): The agent erroneously navigates to Table 1 instead of the TV stand after picking up the object

```
Instruction: Place both a plum and a plate onto the TV stand.

action 1: "<task>Place a plum and a plate on the TV stand.</task> <plan>['navigate to
    sink counter', 'pick plum', 'navigate toward TV-stand area', 'place plum', 'search
    for plate', 'place plate']</plan> <subtask>Navigate to sink counter</subtask> <
    subtask_reason>The plum is normally kept on the kitchen counter near the sink, so
    the first step is to move there.</subtask_reason> <action>('nav(
    receptacle_aabb_sink_kitchen_counter)', 9)</action>",

verification: "The instruction requires obtaining the plum first. Moving from the start
    position to the sink counter is the expected way to reach the plum, so this movement
     advances the task.\naction_is_correct: yes",

=========

action 2: "<task>Place a plum and a plate on the TV stand.</task> <plan>['pick plum', '
    navigate toward TV-stand area', 'place plum', 'search for plate', 'place plate']</
    plan> <subtask>Pick plum</subtask> <subtask_reason>The plum is visible on the
    counter and now within reach, so it should be picked up.</subtask_reason> <action>('
    pick_plum(robot_0)', 30)</action>",

verification: "Grasping the plum is necessary before transporting it anywhere. This
    action clearly moves the task forward.\naction_is_correct: yes"

==========

action 3: "<task>Deliver the plum.</task> <plan>['navigate to TV-stand area', 'place
    plum', 'search for plate', 'place plate']</plan> <subtask>Navigate to TV-stand area
    </subtask> <subtask_reason>The low wooden table in front of the sofa is identified
    as the surface directly in front of the television, so I should move there to drop
    the plum.</subtask_reason><action>('nav(
    receptacle_aabb_Tbl1_Top1_frl_apartment_table_01)', 6)</action>",

verification: "The instruction specifies the TV stand, not Table 1. Navigating to Table
    1 therefore heads to the wrong receptacle and does not directly advance the true
    goal.\naction_is_correct: no",

<following actions omited for clarity>
```

11.2. ALFRED

Example 1 (ALFRED): The agent erroneously picks up the baseball instead of the tennis racket, and turns the lamp off instead of on.

```
Instruction: Grab a tennis racket between beds, turn on the lamp

action 1: "<task>I need to: (1) collect the elongated sporting object I can see between
    the two beds, and (2) operate the nearby lamp.</task>\n<plan>1) Focus on the bat-
    shaped item between the beds and secure it. 2) Locate the desk lamp and toggle its
    power.</plan>\n<subtask>Look for the sporting item I believe to be a BaseballBat.</
    subtask>\n<subtask_reason>The long, thin handle and rounded head suggest a bat; I
    will therefore issue a search for a BaseballBat.</subtask_reason>\n<action>find a
    BaseballBat</action>",

verification: "The instruction requires the agent to locate a TennisRacket, but the
    agent mistakenly assumes the object is a BaseballBat and executes a search for that
    instead.  Although the 'find' command itself executes, it advances the wrong goal.\n
     is_action_correct: no",


============


action 2: "<task>The sporting item has been detected.  I will now secure it.</task>\n<
    plan>Approach the object and grasp it.</plan>\n<subtask>Pick up the BaseballBat.</
    subtask>\n<subtask_reason>With the bat in view, gripping it by the handle completes
    the first half of the instruction.</subtask_reason>\n<action>pick up the BaseballBat
    </action>",

verification: "No BaseballBat is present; the elongated object is actually a
    TennisRacket.  Because the target object does not exist, the simulator would return
    failure.  Attempting this pick-up neither fulfils nor advances the stated task.\n
    is_action_correct: no",


============


action 3: "<task>With the sporting item handled, I will now take care of the lamp.</task
    >\n<plan>Find the DeskLamp, then toggle its power.</plan>\n<subtask>Locate the
    DeskLamp.</subtask>\n<subtask_reason>The lamp is required next; I must bring it into
     view before interacting.</subtask_reason>\n<action>find a DeskLamp</action>",

verification: "Searching for the lamp is sensible even though the racket hasn't been
    obtained yet.  The action does move the agent toward one component of the goal, so
    the step is reasonable.\n is_action_correct: yes"


============


action 4: "<task>The lamp is now in reach.  I will toggle it.</task>\n<plan>Interact
    with the lamp's switch.</plan>\n<subtask>Switch the DeskLamp off.</subtask>\n<
    subtask_reason>The lamp appears lit in my view; turning it off will leave it in the
    desired off state.</subtask_reason>\n<action>turn off the DeskLamp</action>",
```

```
verification: "The instruction explicitly says to turn the lamp ON, not OFF.  The agent
    does the opposite, so even though the environment executes the switch action this
    contradicts the task requirement.\n is_action_correct: no",
```

Example 2 (ALFRED): The agent erroneously places the wine bottle instead of the vase on the table.

```
Instruction: Put a glass vase on a table


action 1: "<task>Instruction outstanding: place the vase on a table.</task>\n<plan>1)
    Locate a suitable decorative vessel; 2) Pick it up; 3) Bring it to a table; 4) Set
    it down.</plan>\n<subtask>Begin by locating a nice glass vessel (going to look for a
     WineBottle that will serve as the vase).</subtask>\n<subtask_reason>The shelves
    often hold tall glass containers; a wine bottle can double as a vase.</
    subtask_reason>\n<action>find a WineBottle</action>",

verification: "The instruction explicitly says to put a glass vase (object class
    Glassbottle/Vase) on a table. The assistant chooses to search for a WineBottle
    instead. Although both are made of glass, a wine bottle is not the requested vase,
    so this action does not directly progress toward satisfying the instruction.\
    naction_is_correct: no",


=============


action 2: "<task>Vessel found; next step is to secure it.</task>\n<plan>Pick the bottle
    up, then head to a table.</plan>\n<subtask>Grasp the WineBottle.</subtask>\n<
    subtask_reason>The object is within reach and I need to carry it to the table.</
    subtask_reason>\n<action>pick up the WineBottle</action>",

verification: "Again, the agent manipulates a WineBottle instead of the required glass
    vase. Picking it up succeeds mechanically, but the chosen object is incorrect for
    the task.\naction_is_correct: no",


=============


action 3: "<task>Object secured; now need a table.</task>\n<plan>Navigate to a dining
    table in the room.</plan>\n<subtask>Find a DiningTable.</subtask>\n<subtask_reason>
    Any table surface will work as a display spot.</subtask_reason>\n<action>find a
    DiningTable</action>",

verification: "Navigating toward a table is generally required, but because the agent is
     holding the wrong object, this search does not really bring it closer to completing
     the instructed task. Therefore the action, while plausible, does not advance the
    true goal.\naction_is_correct: no",


=============

```

```
27
28 action 4: "<task>Table located; time to place the decorative bottle.</task>\n<plan>
       Release the bottle onto the table surface.</plan>\n<subtask>Put the carried bottle
       down on the table.</subtask>\n<subtask_reason>The receptacle is in reach and I no
       longer need to transport the item.</subtask_reason>\n<action>put down the object in
       hand</action>",
29
30 verification: "The assistant places the WineBottle on the table instead of a glass vase.
        The receptacle is correct (a table), but the object does not match the instruction.
        Consequently, the high-level task remains unsatisfied.\naction_is_correct: no",
```

12. Additional Qualitative Examples of Verification During Inference

12.1. LangR

Instruction: Procure a purple fruit and deliver it to the left counter.

Sampled Action: <task>Find and pick up the purple fruit and then deliver it to the left counter.</task> <plan>[Search TV stand, search sofa, search table2, pick fruit, navigate to left counter, place fruit]</plan> <subtask_reason>At the TV stand, I should try a pick to see if the fruit is here, hence pick(object).</subtask_reason> <subtask>Attempt to pick the fruit at the TV stand.</subtask> <action>('pick_strawberry(robot_0)', 38)</action>

Verification CoT: The assistant tries to pick a strawberry instead of the requested purple fruit. This does not help fulfil the instruction and therefore does not advance the task. **action_is_correct: no**

Figure 7. Verification example on the LangR benchmark. Here, the task is to move a "purple fruit", but the agent tries to pick up a strawberry. The verifier correctly identifies that the strawberry is not the right object, classifying the action as incorrect.

Instruction: Move the rubriks cube to the brown table, the lego to the brown table, and the bowl to the brown table.

Sampled Action: <task>Remaining: locate and pick the bowl (it was not found at the TV stand).</task> <plan>[Return to table1 to search for bowl, then cabinet area, pick bowl, navigate to brown table, place bowl]</plan> <subtask_reason>I am at the brown table and need to check if the bowl is here, hence I should attempt a place action to see if it is present</subtask_reason> <subtask>attempt to place bowl on brown table</subtask> <action>('place(receptacle_aabb_Tbl2_Top1_frl_apartment_table_02)', 51)</action>

Verification CoT: The robot is not holding any object (the previous pick attempt failed and no new object was picked). Attempting to place an item while not holding anything cannot succeed and does not progress the task. **action_is_correct: no**

Figure 8. Verification example on the LangR benchmark. Here, the agent erroneously attempts to place the bowl without successfully picking it up first. The verifier identifies this mistake, classifying the action as incorrect.

12.2. ALFRED

Instruction: Place washed leafy green vegetable in a receptacle that can keep it fresh for several days

Sampled Action: <task>The fridge is open and I (mistakenly) see another lettuce inside; I still intend to store the held lettuce.</task> <plan>[Optional] Decide whether to store this extra lettuce or switch back to my original intent, then ...</plan> <subtask_reason>I am uncertain of needing both wastebulbs; closing keeps the environment tidy while I reconsider.</subtask_reason> <subtask>Close the fridge door.</subtask> <action>close the Fridge</action>

Verification CoT: Closing the fridge at this moment is counter-productive because the lettuce needs to remain inside to stay fresh. Although syntactically valid, it moves the state away from the goal. **action_is_correct: no**

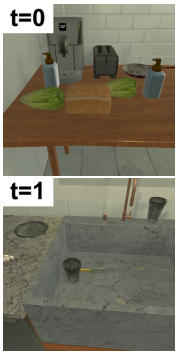
Figure 9. Verification example on the ALFRED benchmark. Here, the agent incorrectly attempts to close the fridge before putting the lettuce inside because it mistakenly detects a second lettuce in the fridge. The verifier identifies this mistake, classifying the action as incorrect.

Instruction: As you prepare for dinner, arrange an eating surface that will hold your food with a spoon in it on the kitchen table.

Sampled Action: <task>Entire task still outstanding: create a state where the dining-table has a container holding a spoon resting on it.</task> <plan>[1] locate a spoon-like utensil, [2] pick it up, [3] find a bowl or cup, [4] place utensil inside it, [5] carry the filled utensil to the dining-table, [6] set it down on the table.</plan> <subtask_reason>Before anything else I need a spoon; without knowing its position I cannot grasp or use it. Therefore I must visually search for an appropriate utensil first.</subtask_reason> <subtask>Locate a spoon-like utensil.</subtask> <action>find a Ladle</action>

Verification CoT: The user's instruction explicitly calls for a "spoon." The agent instead decides to search for a Ladle, which is a different utensil. Although both are eating utensils, a ladle is not the requested object. Thus the chosen action does not advance the stated goal directly. **action_is_correct: no**

Figure 10. Verification example on the ALFRED benchmark. Here, the task is to move a spoon, but the agent proposes to move a ladle, which is not in the scene. The verifier correctly identifies that the correct object is a spoon, classifying the action as incorrect.



Instruction: During a cooking session, keep a clean flipping tool on a wooden table.

Sampled Action: <task>Entire task still outstanding: locate a suitable flipping utensil, make sure it is clean, and finally leave it resting on the wooden dining table.</task> <plan>1) Find a knife. 2) Pick it up. 3) Go to a sink. 4) Wash the knife by turning water on then off. 5) Bring the clean knife to the wooden dining table and set it down there.</plan> <subtask_reason>The very first requirement is to know where a knife is; without visual contact I cannot interact with it or progress toward washing or placing. Therefore I need to perform a search action that will move my sensors through the scene until a knife is detected.</subtask_reason> <subtask>Find a knife</subtask> <action>find a Knife</action>

Verification CoT: The instruction asks for any kind of cleaning flipping tool (i.e., a knife). By performing the action "find a Knife" the agent begins the search for such an object, which is a reasonable first step toward accomplishing the goal. **action_is_correct: yes**

Figure 11. Verification example on the ALFRED benchmark. Here, the task is to move a spatula, but the agent proposes to move a knife. The verifier fails to identify this mistake and incorrectly classifies the action as correct.