

Skill-Acquisition in LLM Agents

Isai Garcia-Baza, Scott Merrill, Titus Spielvogel

Problem

Problem Statement:

- Given an agent that interacts with an environment to solve a set of problems using a set of primitive actions
- Learn new actions which are a composition of primitive actions that are helpful in solving the set of problems

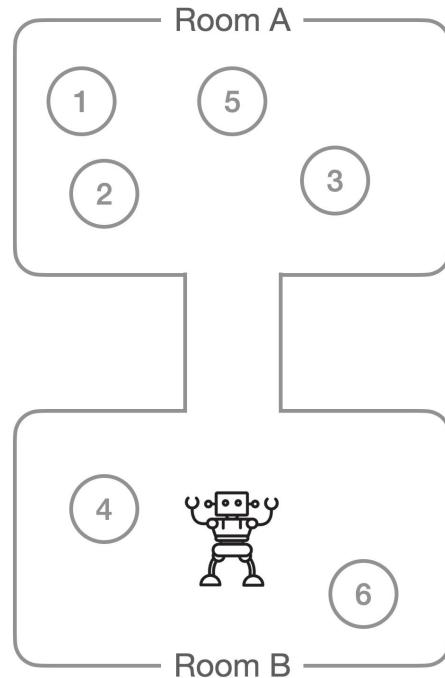
Key Idea: Equip the agent with the ability to prompt an LLM to learn new actions and decide which actions to take next

Input: Environment description, environment start state and goal state, set of problems, primitive actions, few shot examples for LLM Prompts

Output: A set of learned actions (their python code and their description)

Gripper Environment

- 2 rooms, 6 balls, and a robot with 2 grippers
- 3 possible actions:
 - move(roomA, roomB),
 - pickup(obj, room, gripper)
 - drop(obj, room, gripper)
- Dataset contains 20 problems each problem is described by a initial state and goal state of the environment (e.g. Ball 1 is in Room A...)
- 3 problems for training, 17 problems for testing



Input/Output Example - Gripper

Input

Actions: move(roomA, roomB), pickup(obj, room, gripper), drop(obj, room, gripper)

Initial State: Ball 1, 2, 3, 5 are in Room A, Ball 4, 6 are in Room B, Robot is in Room A, Right and Left Gripper are free

Goal State: Ball 4, 6 are in Room A

Output

```
def move_objects(room1, room2, objects, gripper):  
    for obj in objects:  
        pickup(obj, room1, gripper)  
        move(room1, room2)  
        drop(obj, room2, gripper)  
        move(room2, room1)
```

Description: This action moves all objects from one room to another using a gripper.

Input: room1 (string), room2 (string), objects (list): list of objects to move from room1 to room2, gripper (string)

Required World State: all objects must be in room1, the robot must be in room1, gripper must be free

Resulting World State: all objects are now in room2, the robot is in room1

Solution: Modifications to LearnAct

- Modification motivated by the Voyager [4] and LearnAct Paper [1]

Hypothesis 1: While GPT-3.5 can handle 16k-tokens, shorter prompts may be better

- Shorten prompts and made them more precise (docstrings, understand state of the world, etc.)

Hypothesis 2: LLM models fine tuned on python code are better at coding, while GPT-3.5 is better for reasoning

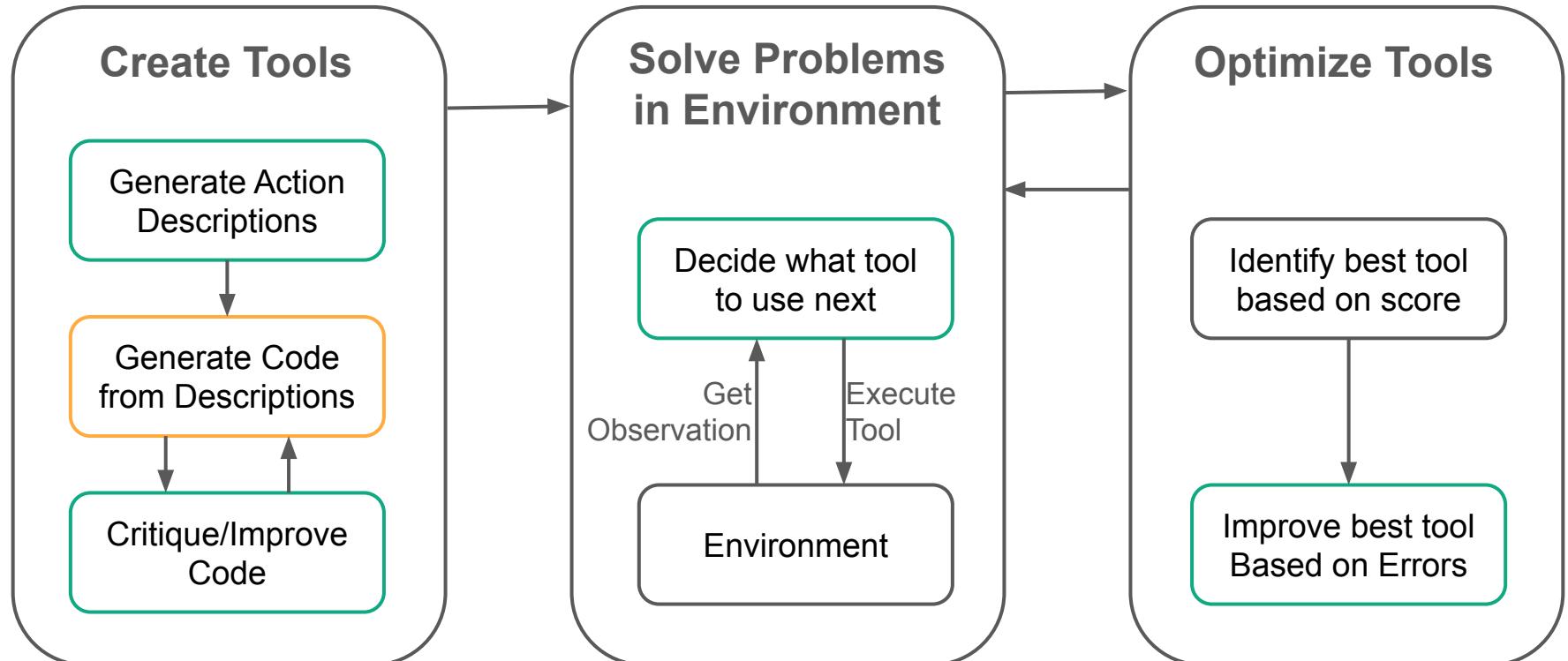
- Differentiated between code generation and reasoning tasks

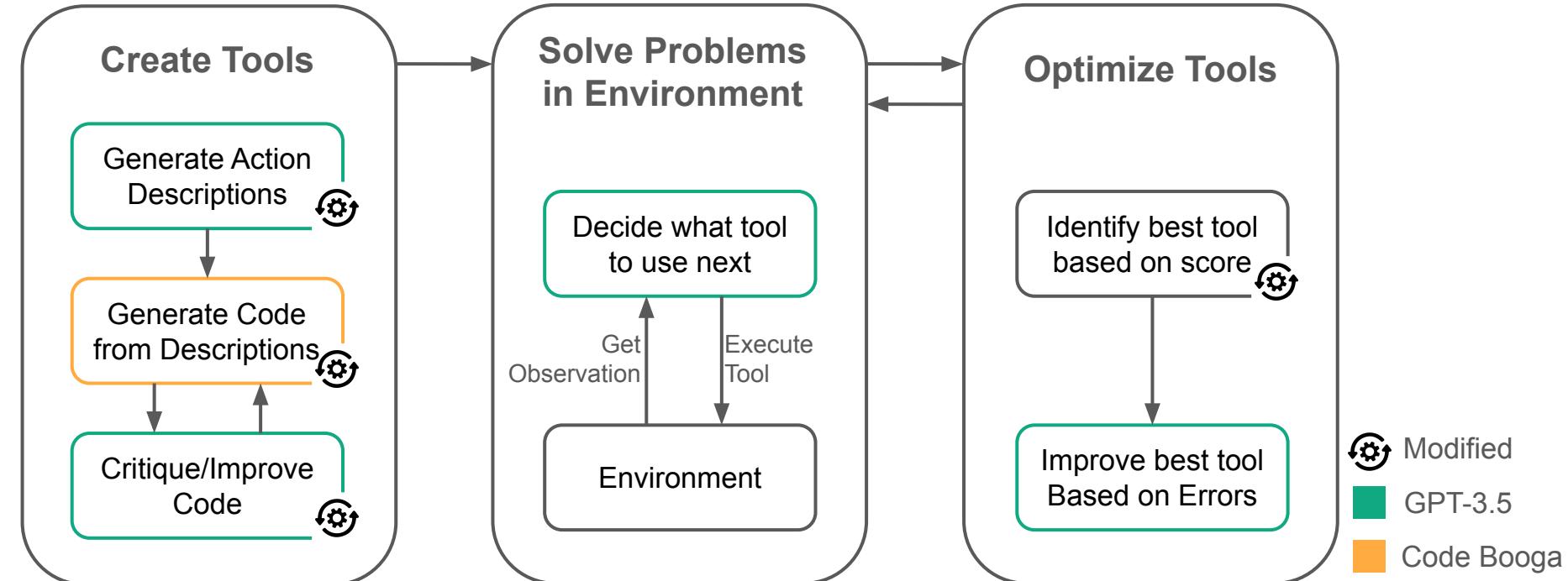
Hypothesis 3: Using critique methods from LLM literature, we can prompt LLM to iteratively improve it's code

- Add a critic LLM prompt to correct and improve original code generation result

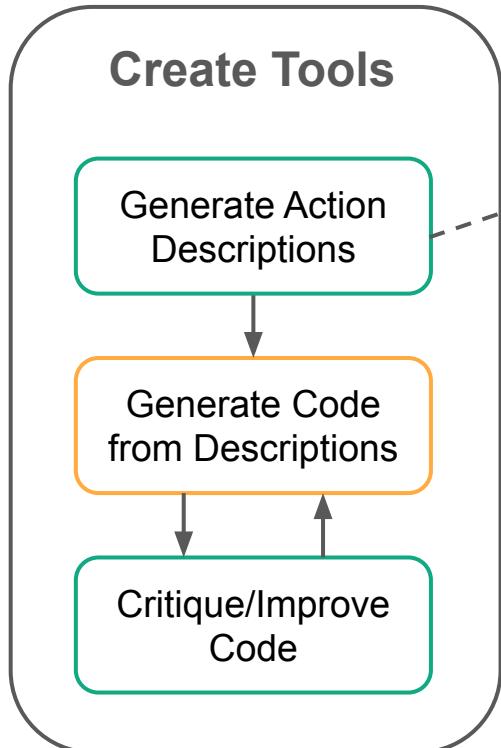
Methods

█ GPT-3.5
█ Code Llama





Methods



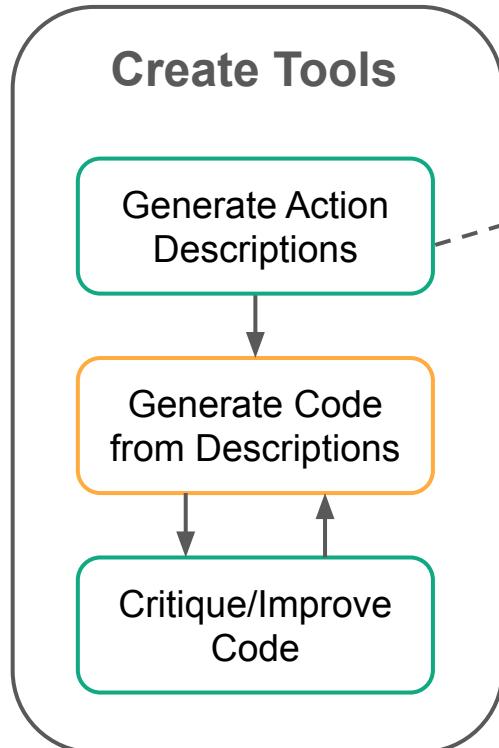
Prompt

You are a robot with a gripper that can pickup and move objects between different rooms.

Here are the actions in the domain:
<List of Tools and their Docstring>

Please **combine at least two of the above actions** into functions composite helper functions.
Please list a description, the inputs, the required and resulting world state.

Methods



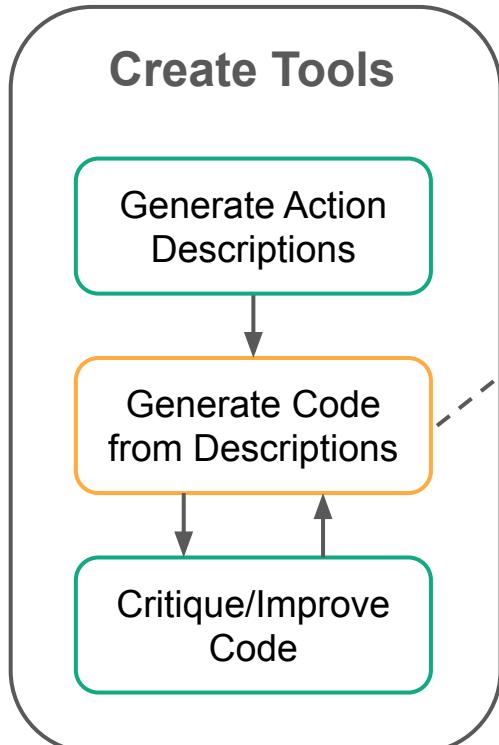
Response

```
carry_to(robot_loc, object_name, object_loc,  
destination_loc, gripper_tool):
```

<Description>
<Inputs>
Required World State,
Resulting World State:
Basic Functions Used

```
move_object(...):  
...
```

Methods



Prompt

Assume you have access to the the following functions:
<List of Tools and their Docstring>

You will be given a function description and your task is to write that function.

Example 1:

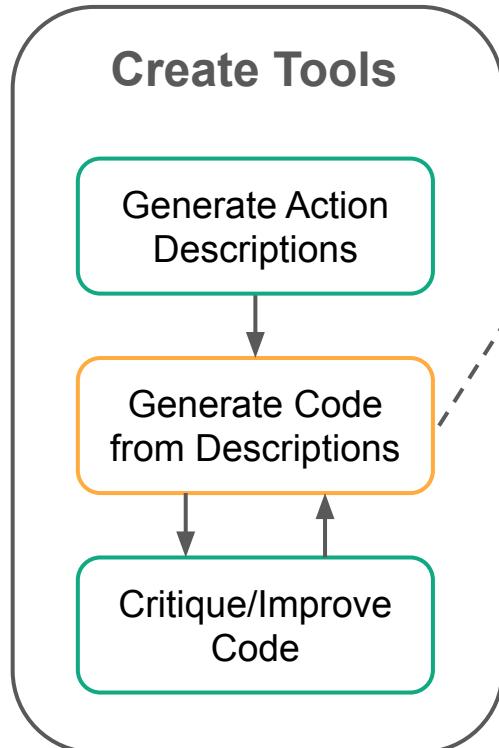
<one shot example>

Now please **implement** the function:

```
carry_to(robot_loc, object_name, object_loc,  
        destination_loc, gripper_tool)
```

<Docstring of carry_to>

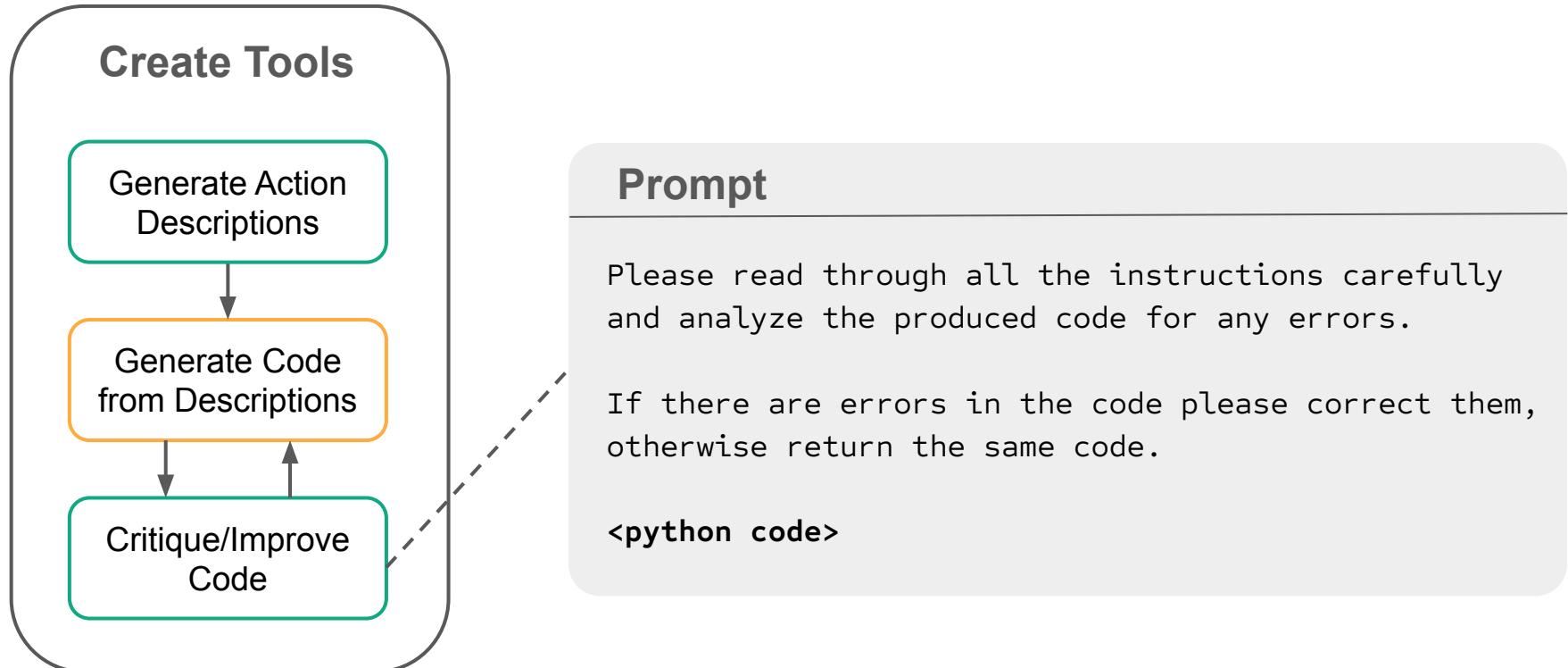
Methods



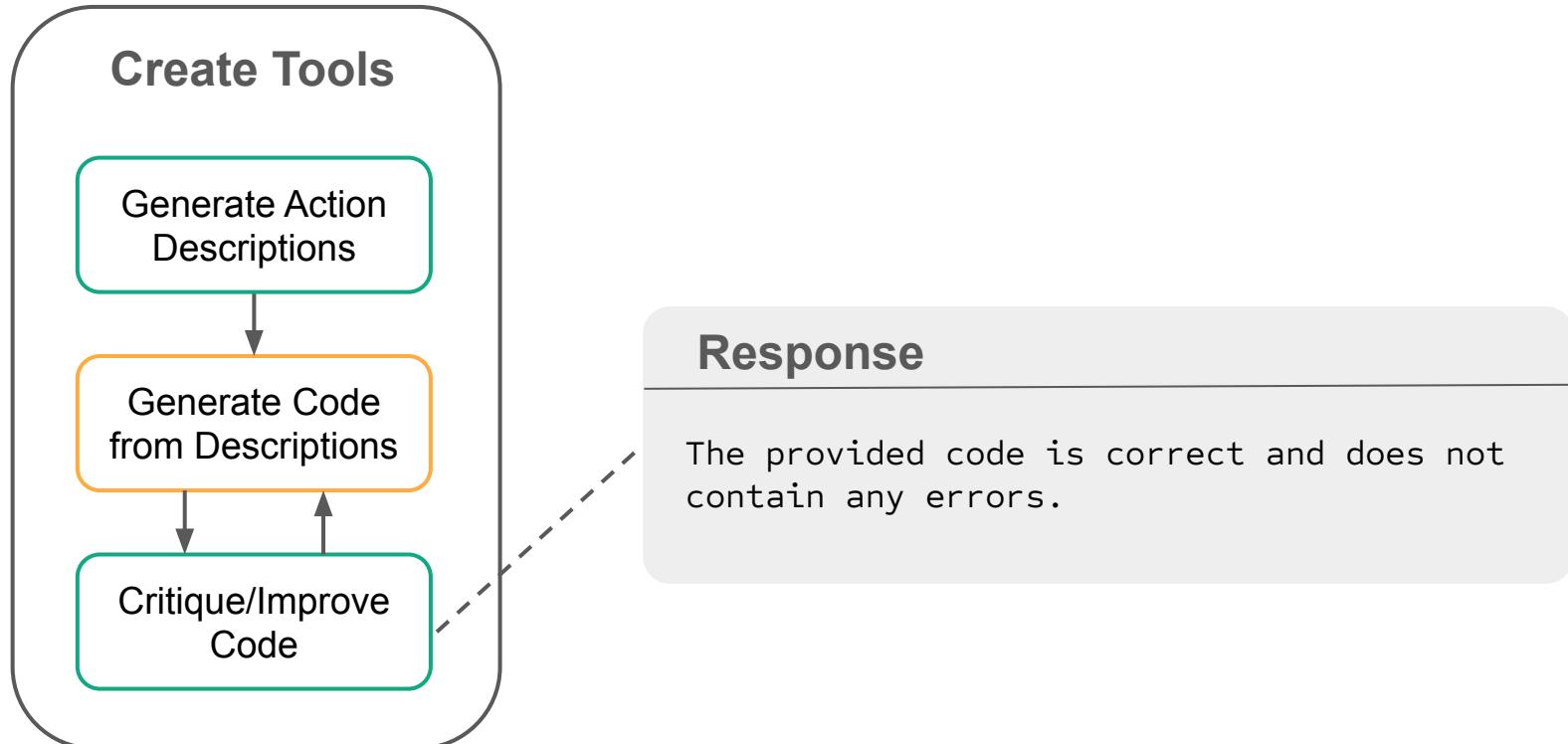
Response

```
def carry_to(robot_loc, object_name, object_loc,  
            destination_loc, gripper_tool):  
  
    move(robot_loc, object_loc)  
    pick(object_name, object_loc, gripper_tool)  
    move(object_loc, destination_loc)  
    drop(object_name, destination_loc, gripper_tool)
```

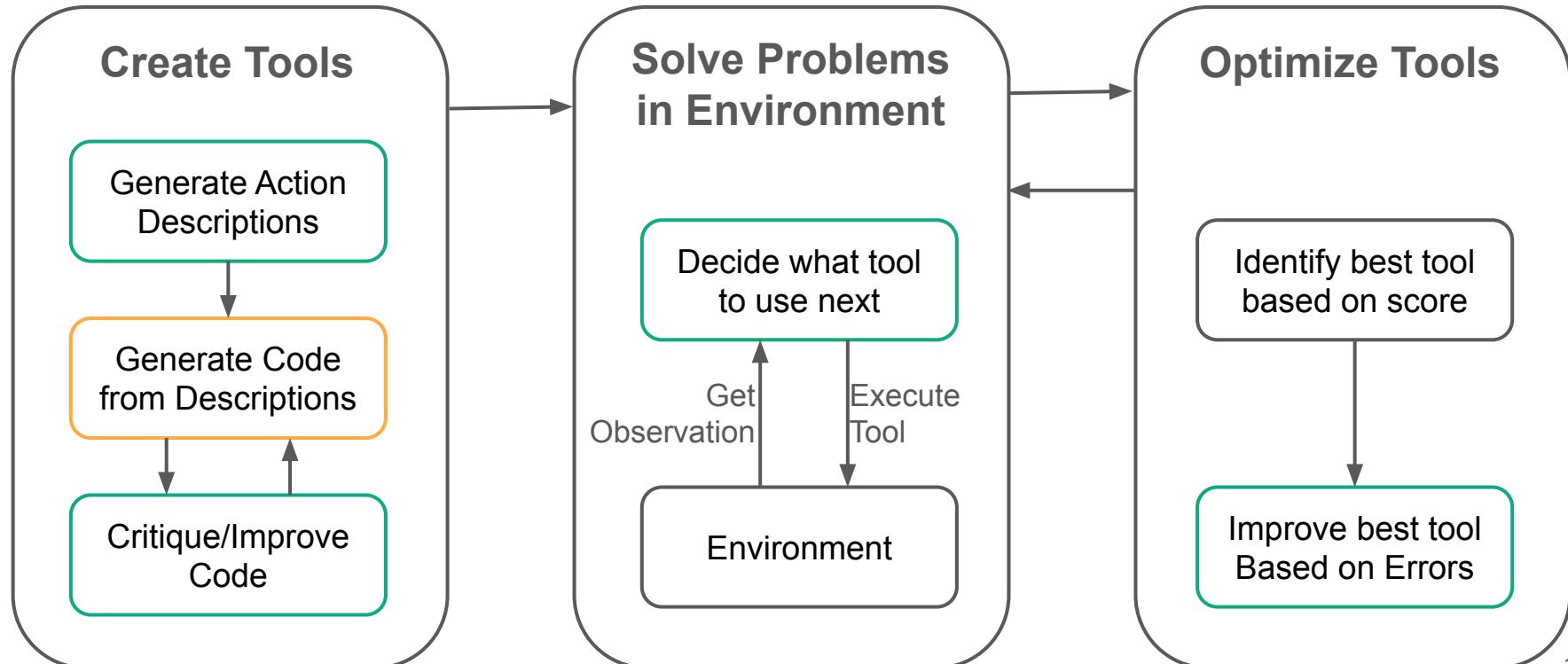
Methods



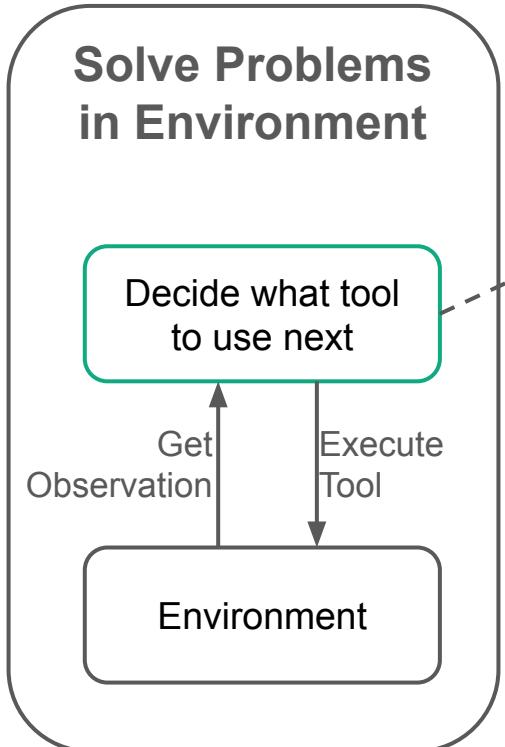
Methods



Methods



Methods



Prompt

You are a robot with a gripper that can move objects between different rooms.

There are actions defined in this domain:

<List of **primitive Tools** and their Docstring>
carry_to(...)

<Docstring for carry_to>

move_object(...)

<Docstring for carry_to>

Here are examples:

<**few shot examples**>

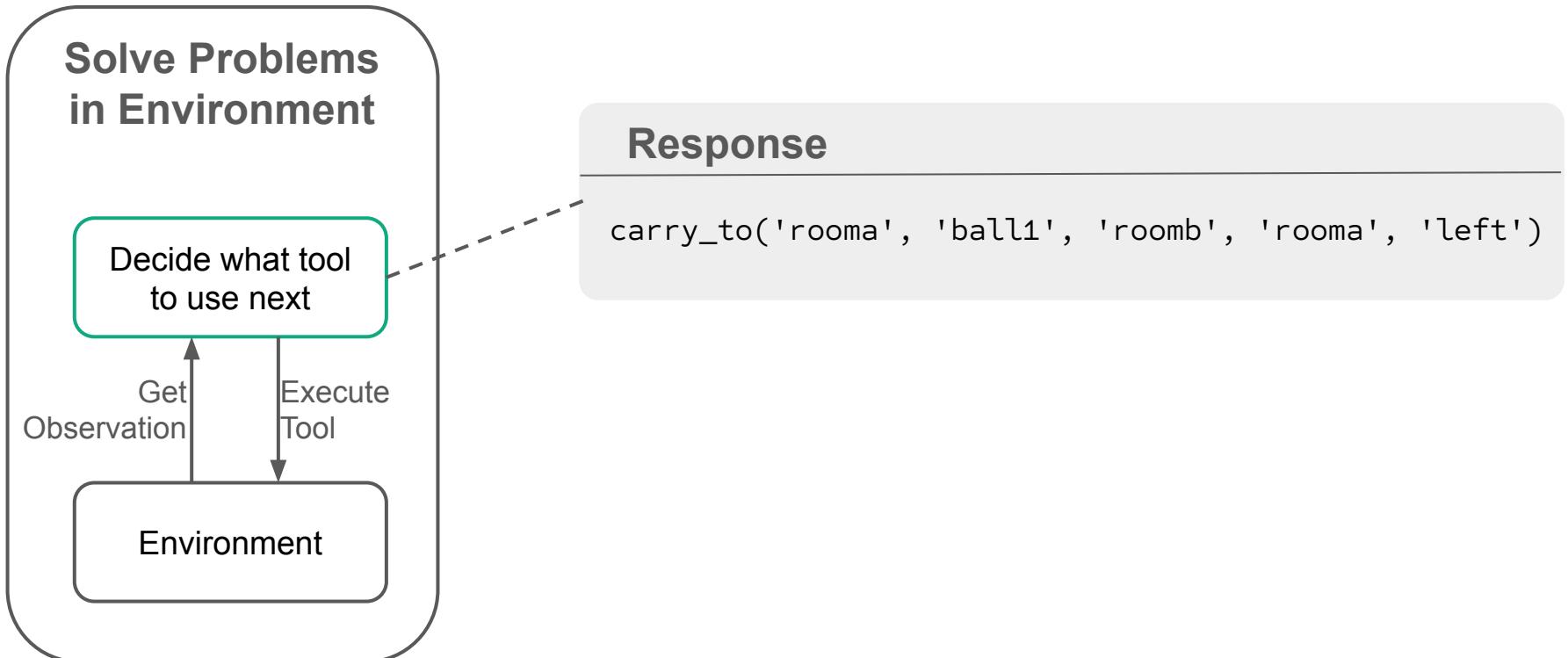
You should perform actions to accomplish the goal:

The goal is to satisfy the following conditions:

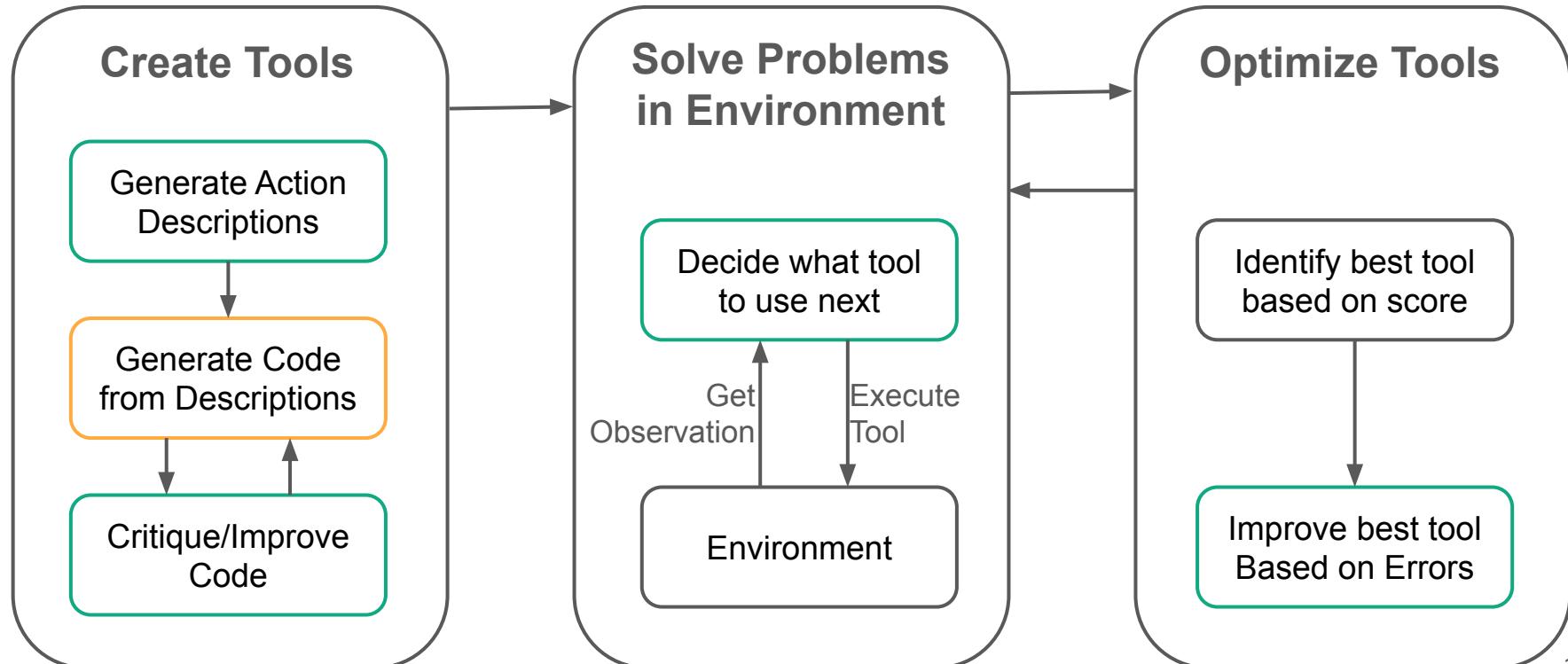
Observation:

Action:

Methods



Methods



Experiments & Results

	Step Accuracy higher is better	Task Success Rate (GPT-3.5) higher is better	Task Success Rate (GPT- 4) higher is better	Avg. # of steps to solve task (GPT-3.5), lower is better
Learn Act (with our modifications)	64.37%	70.0	N/A	4.78
Baseline 1: LearnAct	27.70%	45.0	82.5	9.11
Baseline 2: Voyager	NA	2	76.5	NA

Examples of
tools created

```
def move_object(robot_loc, object_name, object_loc, destination_loc,  
gripper_tool):  
    move(robot_loc, object_loc)  
    pick(object_name, object_loc, gripper_tool)  
    move(object_loc, destination_loc)  
    drop(object_name, destination_loc, gripper_tool)  
  
def carry_to(robot_loc, object_name, object_loc, destination_loc, gripper_tool):  
    move(robot_loc, object_loc)  
    pick(object_name, object_loc, gripper_tool)  
    move(object_loc, destination_loc)  
    drop(object_name, destination_loc, gripper_tool)
```

Conclusion

- Decomposition of the whole process into multiple steps helped (e.g. first generating a list of tools, then code)
- Reducing the number of token per LLM Prompt increased the quality of the LLM response
- Use GPT 3.5 for reasoning tasks and Code Llama [6] for generating code
- Being as precise as possible in the prompt helped (e.g., prompting the LLM for a structured docstring instead of just a method description)
- Shorter avg. steps to solve shows there is space for domain-specific LLM agents to work together rather than relying on one generalist model

Q&A

References

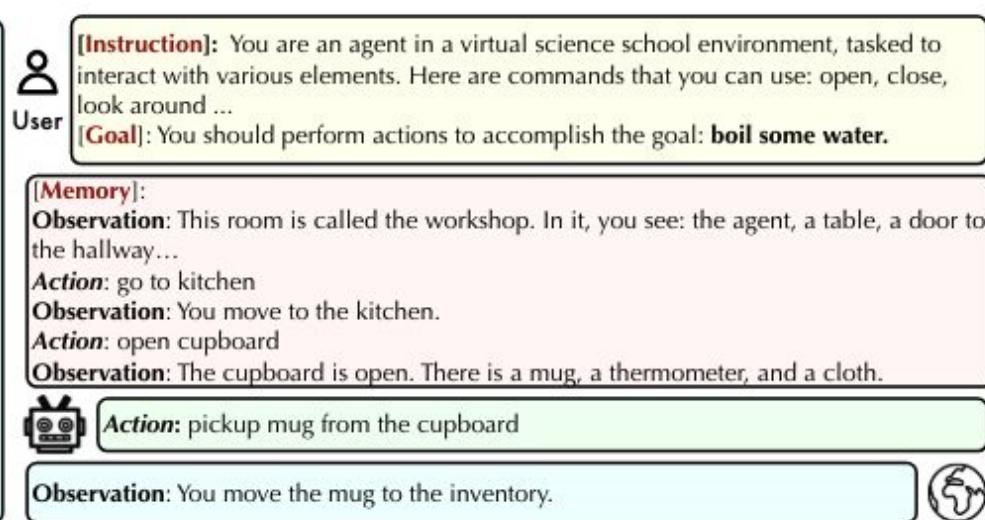
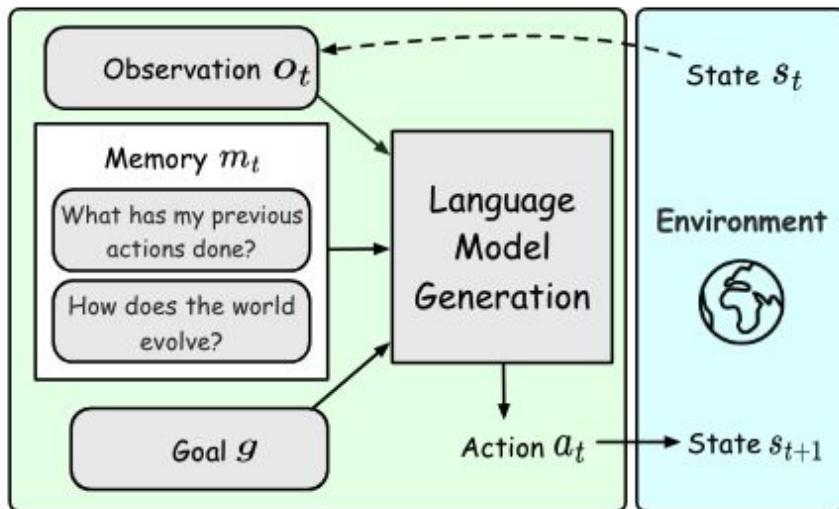
- [1] Zhao, H., Ma, C., Wang, G., Su, J., Kong, L., Xu, J., ... & Yang, H. (2024). Empowering Large Language Model Agents through Action Learning. *arXiv preprint arXiv:2402.15809*.
- [2] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- [3] Ma, C., Zhang, J., Zhu, Z., Yang, C., Yang, Y., Jin, Y., ... & He, J. (2024). AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *arXiv preprint arXiv:2401.13178*.
- [4] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., ... & Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- [5] Silver, T., & Chitnis, R. (2020). PDDLGym: Gym environments from PDDL problems. *arXiv preprint arXiv:2002.06432*.
- [6] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., ... & Synnaeve, G. (2023). Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Backup Slides

Related Work

- Many attempts have been made to solve RL control problems by exploiting the code generation and reasoning abilities of LLMs
- Two notable ones that inspired our work are:
 1. Voyager (2021):
 2. LearnAct (2024):

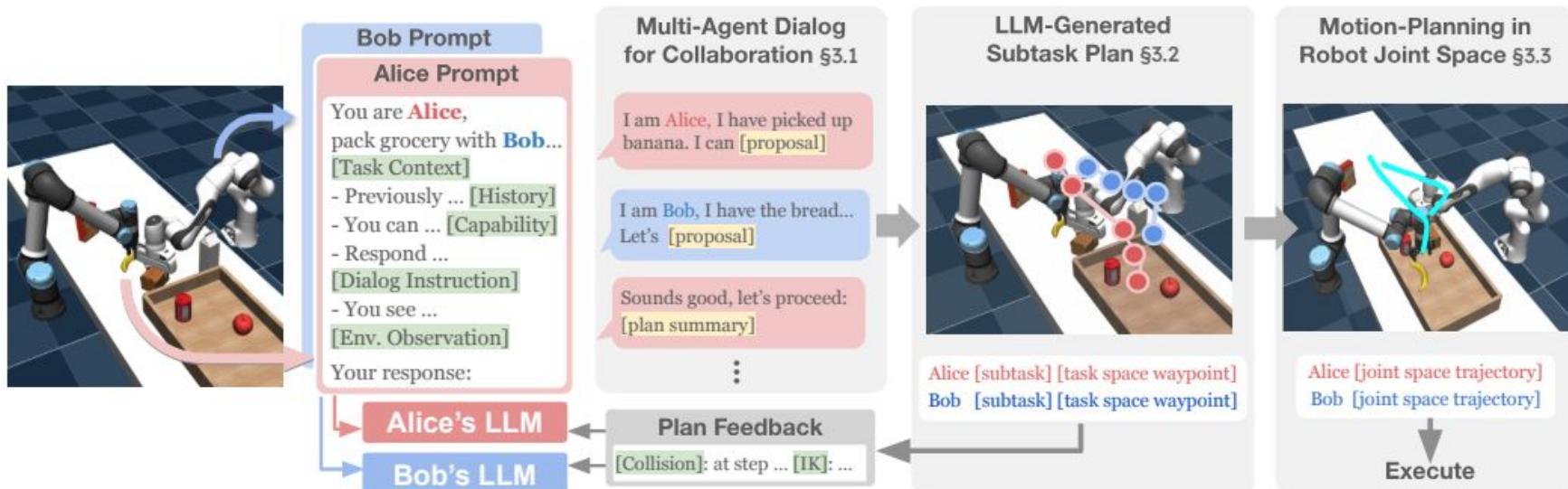
LLM Agents Overview



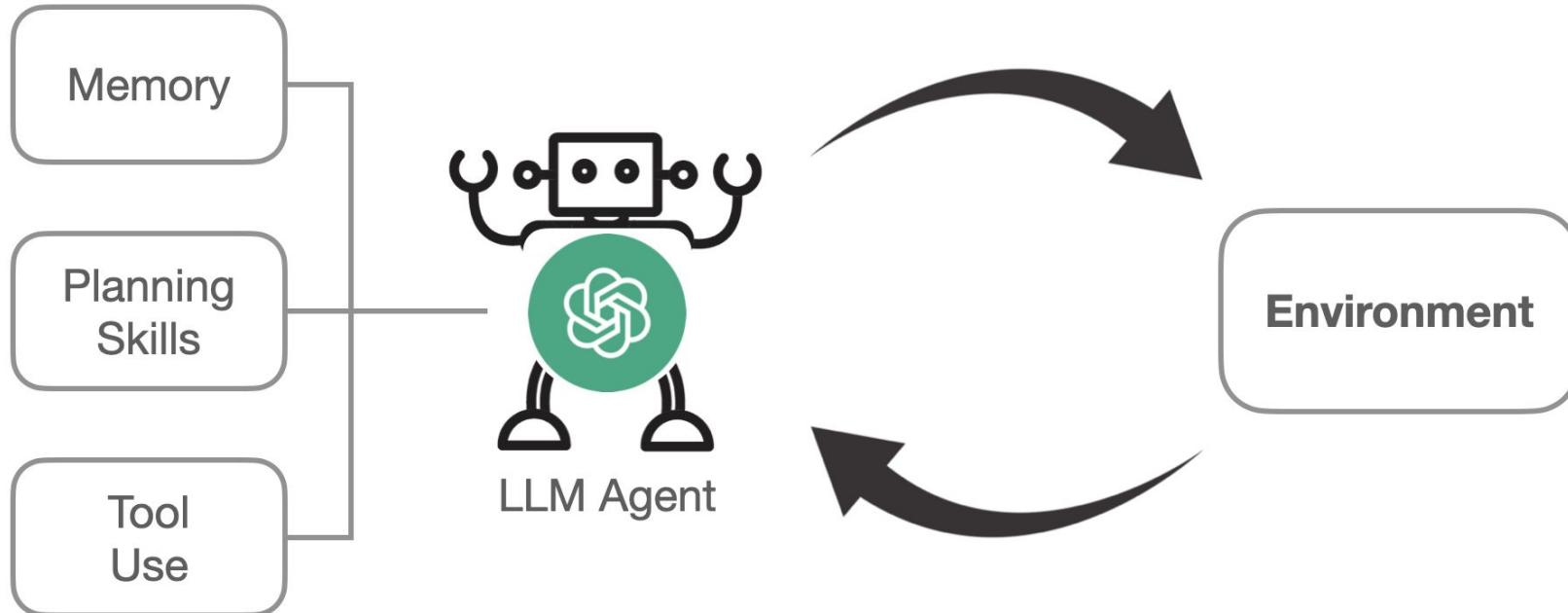
References

- [1] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *CoRR*, abs/2010.03768, 2020.
- [2] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [3] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.
- [4] Lilian Weng. LLM powered autonomous agents. [https://lilianweng.github.io/ posts/2023-06-23-agent/](https://lilianweng.github.io/posts/2023-06-23-agent/). Accessed on: 02/04/2023.
- [5] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Problem: Gripper Environment



LLM Agent

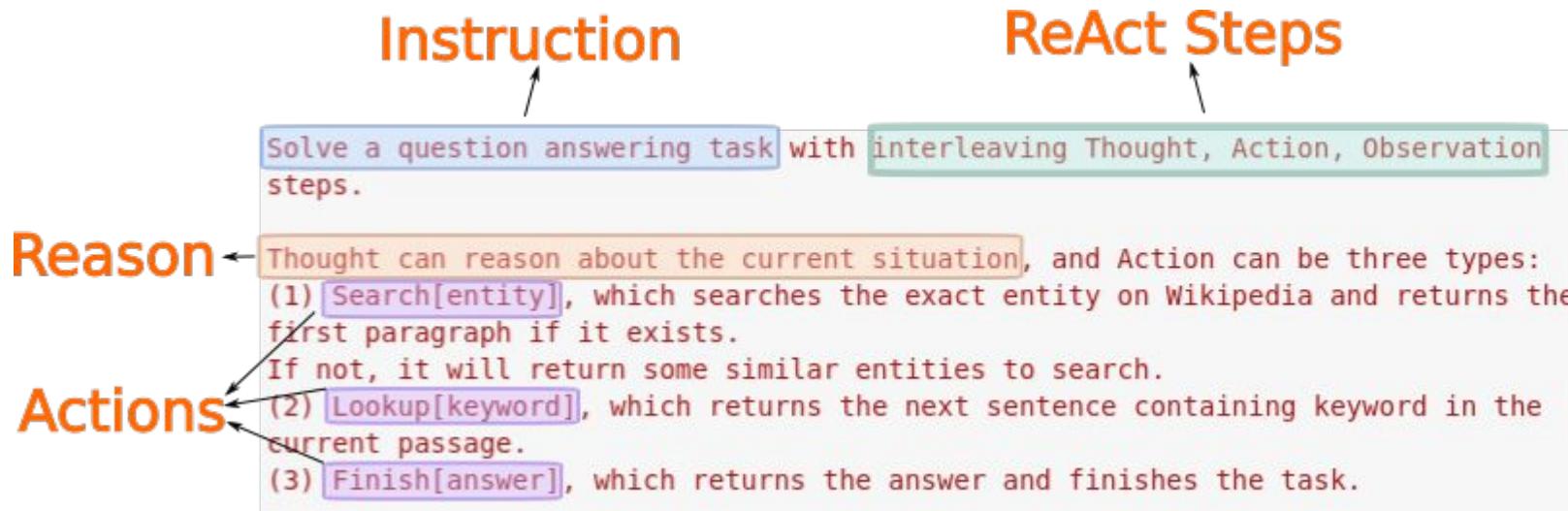


Problem Statement

- How can LLM agents acquire novel skill by combining primitive skills?
 - How their “tools” can be broken down into more elementary primitives?
 - How these primitives can be combined?
 - How to guide an LLM Agent to acquire the appropriate skills for solving a final task?
- Data Source
 - GPT3.5 and GPT4.0 from the Azure OpenAI API as backbone LLMs
 - custom reinforcement learning environments such as a grid world
 - If successful, more complex environments, such as ALFWorld

ReAct Prompting Example

- React prompt to LLM has 4 key components
- Example: <https://github.com/smerrillunc/COMP590/blob/main/SimpleGridWorld.ipynb>



LearnAct Skill Learning

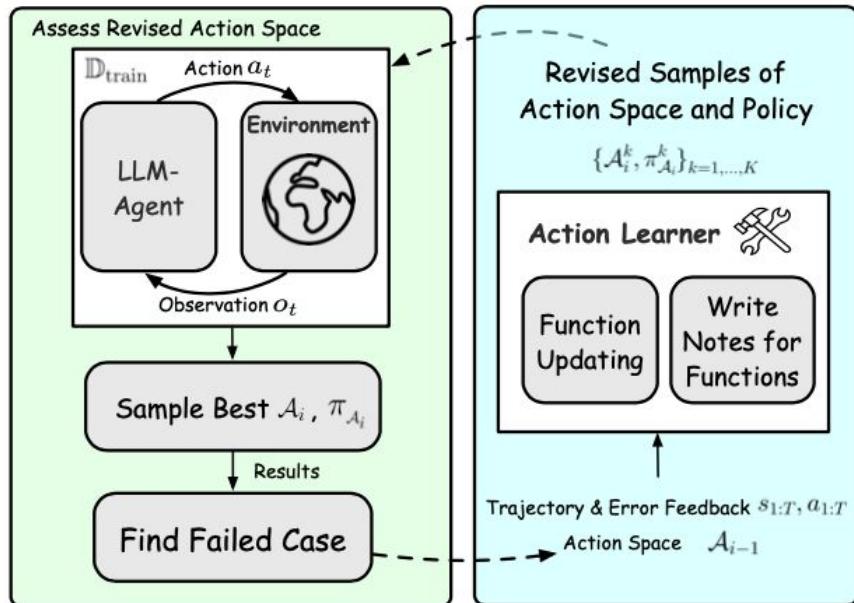
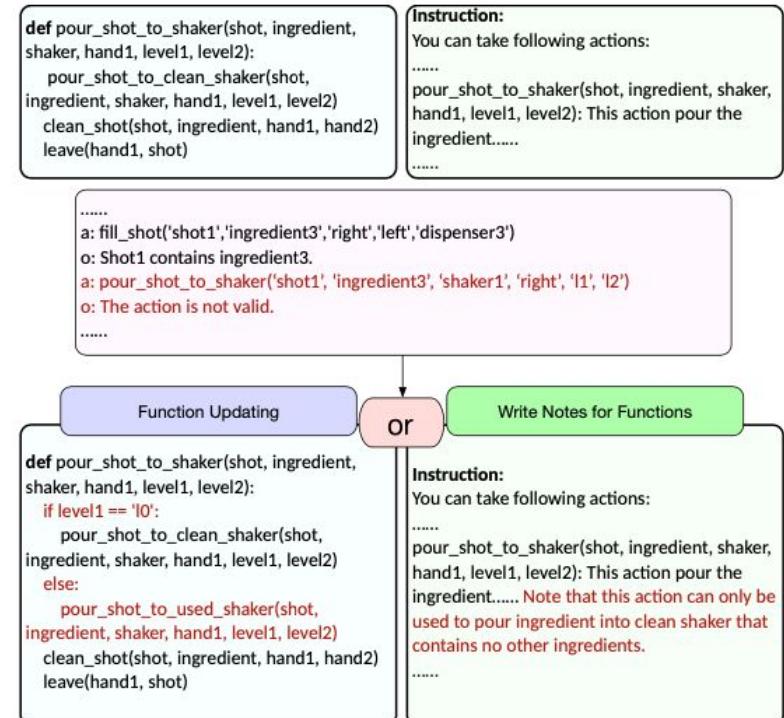
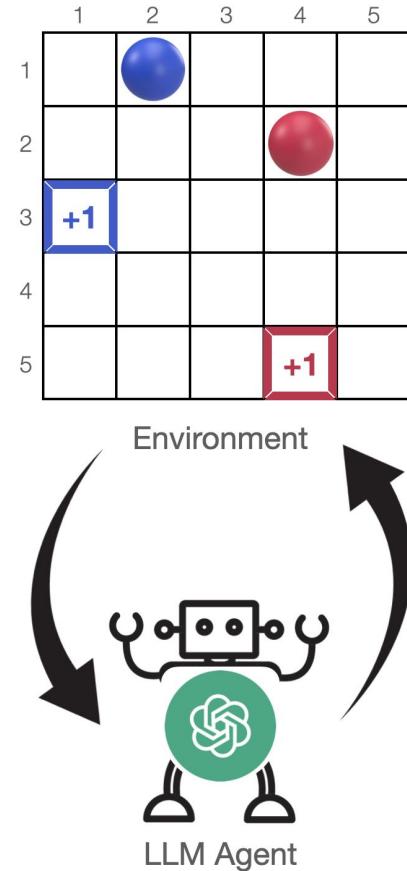


Figure 2: During the learning stage, action usage by agent and action optimization are repeatedly executed. The im-



Evaluation

- Goal: Place blue ball on blue cell and red ball on red cell
- Equip LLM agent with custom skills that are
 - Only useful for a part of task
 - Need to be further broken down
- For example: “pickup red ball” and “put down blue ball”
 - “locate ball”, “go to ball”, “pick up ball”, “go to cell” and “put down ball”
- May be able to observe log of substeps
- Some actions may be purposefully corrupted requiring reformatting of substeps
- Measure Success Rate and reward agent using total discounted rewards to optimize for solutions in fewer steps.



Summary Prompt

You are a robot with a gripper that can pickup and move objects between different rooms. You will be told the locations of the objects, the location of the robot, and whether the gripper is free or occupied. You will also be told which objects need to be in which room.

Here are the actions in the domain.

move(<robot_loc>,<room1>):

- Description: This action moves the robot from robot_loc to room1.
- Inputs: robot_loc (string): robot's current location, room1 (string): room to move to
- Required World State: Robot is at robot_loc
- Resulting World State: Robot has moved to room1

pick(<obj_name>,<obj_room>,<gripper>):

- Description: This action picks up obj_name in obj_room using it's specified gripper.
- Inputs: obj_name (string): object to pickup, obj_room (string): the current room of obj_name, gripper (string): the gripper to use to pickup obj_name
- Required World State: Robot is in obj_room. obj_name is in obj_room. gripper is free
- Resulting World State: gripper is not free, obj_name is in gripper

drop(<obj_name>,<current_room>,<gripper>):"

- Description: This action drops obj_name currently held in gripper in current_room.
- Inputs: obj_name (string): object to drop, current_room (string): the current room the robot is in, gripper (string): the gripper holding obj_name
- Required World State: Robot is in current_room. obj_name is in gripper. gripper is not free
- Resulting World State: gripper is free, obj_name is in current_room, robot is in current room

Please combine at least two of the above actions into functions composite helper functions. These functions should be useful to move objects between rooms. Please list a description of the function, the inputs to the function, the required world state to use the function, the resulting world state after the function is used and the basic functions used.

LLM Summary Prompt Response

1. carry_to(robot_loc, object_name, object_loc, destination_loc, gripper_tool):

- Description: This function moves the robot from robot_loc to object_loc. Then it picks up object_name using gripper_tool, moves from object_loc to destination_loc, and drops the object_name using gripper_tool.
- Inputs: robot_loc (string): current location of the robot, object_name (string): name of the object to be carried, object_loc (string): current location of object_name, destination_loc (string): destination to move object_name to, gripper_tool (string): gripper to use to pick up and drop object_name
- Required World State: gripper_tool is free
- Resulting World State: gripper_tool is free, robot is at destination_loc, object_name is at destination_loc
- Basic Functions Used: move, pick, drop

2. move_object(robot_loc, object_name, object_loc, destination_loc, gripper_tool):

- Description: This function moves the robot from robot_loc to object_loc, picks up object_name using gripper_tool, moves from object_loc to destination_loc while carrying object_name, and drops object_name at the destination_loc from gripper_tool.
- Inputs: robot_loc (string): current location of the robot, object_name (string): name of the object to be moved, object_loc (string): current location of object_name, destination_loc (string): destination to move object_name to, gripper_tool (string): the left or right gripper to use
- Required World State: gripper_tool is free
- Resulting World State: gripper_tool is free, robot is at destination_loc, object_name is at destination_loc
- Basic Functions Used: move, pick, drop

Code Prompt

Assume you have access to the the following functions.

move(<robot_loc>,<room1>):- Description: This action moves the robot from robot_loc to room1.

- Inputs: robot_loc (string): robot's current location, room1 (string): room to move to

- Required World State: Robot is at robot_loc

- Resulting World State: Robot has moved to room1

pick(<obj_name>,<obj_room>,<gripper>):- Description: This action picks up obj_name in obj_room using it's specified gripper.

- Inputs: obj_name (string): object to pickup, obj_room (string): the current room of obj_name, gripper (string): the gripper to use to pickup obj_name

- Required World State: Robot is in obj_room. obj_name is in obj_room. gripper is free

- Resulting World State: gripper is not free, obj_name is in gripper

drop(<obj_name>,<current_room>,<gripper>):- Description: This action drops obj_name currently held in gripper in current_room.

- Inputs: obj_name (string): object to drop, current_room (string): the current room the robot is in, gripper (string): the gripper holding obj_name

- Required World State: Robot is in current_room. obj_name is in gripper. gripper is not free

- Resulting World State: gripper is free, obj_name is in current_room, robot is in current room

You will be given a function description and your task is to write that function. And wrap all executable code with ``python ``. Do not write any unnecessary functions.

Example 1:

move_pickup_drop(robot_loc, object_loc, object_name, destination_loc, gripper):

- Description: This function moves the robot from robot_loc to object_loc, picks up object_name using gripper, moves from object_loc to destination_loc, and drops object_name using its gripper.

- Inputs: robot_loc (string): current location of robot, object_loc ('string'): current location of object, object_name (string): object to be moved, destination_loc (string): destination to move object to, gripper (string): gripper to use to pickup and drop object

- Required World State: Robot is at robot_loc, object_name is in object_loc, gripper is free

- Resulting World State: Robot is at destination_loc, object_name is in destination_loc, gripper is free

- Basic Functions Used: move, pick, drop

You will respond:

```python

def move\_pickup\_drop(robot\_loc, object\_loc, object\_name, destination\_loc, gripper):

move(robot\_loc, object\_loc)

pick(object\_name, object\_loc, gripper)

move(object\_loc, destination\_loc)

drop(object\_name, destination\_loc, gripper)

...

Now please implement the function carry\_to(robot\_loc, object\_name, object\_loc, destination\_loc, gripper\_tool), wrap all executable code with ``python ``.

- Description: carry\_to(robot\_loc, object\_name, object\_loc, destination\_loc, gripper\_tool) :- Description: This function moves the robot from robot\_loc to object\_loc. Then it picks up object\_name using gripper\_tool, moves from object\_loc to destination\_loc, and drops the object\_name using gripper\_tool.

- Inputs: robot\_loc (string): current location of the robot, object\_name (string): name of the object to be carried, object\_loc (string): current location of object\_name, destination\_loc (string): destination to move object\_name to, gripper\_tool (string): gripper to use to pick up and drop object\_name

- Required World State: gripper\_tool is free

- Resulting World State: gripper\_tool is free, robot is at destination\_loc, object\_name is at destination\_loc

- Basic Functions Used: move, pick, drop.

# Error Correction Prompt

Please read through all the instructions carefully and analyze the produced code for any errors.

You are not allowed to ask any questions or make assumptions on global variables.

If there are errors in the code please correct them, otherwise return the same code.

If the required task is not solvable you should say 'Impossible.'

If there are errors in the code you should say 'ERROR' then provide a correction.



# LLM Action Query

You are a robot with a gripper that can move objects between different rooms. Your name is Robby.  
There are actions defined in this domain:

move(<robot\_loc>,<room>):

- Description: This action moves the robot from `robot_loc` to `room1`.
- Inputs: `robot_loc` (string): robot's current location, `room1` (string): room to move to
- Required World State: Robot is at `robot_loc`
- Resulting World State: Robot has moved to `room1`

pickup(<obj\_name>,<obj\_room>,<gripper>):

- Description: This action picks up `obj_name` in `obj_room` using it's specified gripper.
- Inputs: `obj_name` (string): object to pickup, `obj_room` (string): the current room of `obj_name`, `gripper` (string): the gripper to use to pickup `obj_name`
- Required World State: Robot is in `obj_room`, `obj_name` is in `obj_room`, gripper is free
- Resulting World State: gripper is not free, `obj_name` is in gripper

drop(<obj\_name>,<current\_room>,<gripper>):

- Description: This action drops `obj_name` currently held in gripper in `current_room`.
- Inputs: `obj_name` (string): object to drop, `current_room` (string): the current room the robot is in, `gripper` (string): the gripper holding `obj_name`
- Required World State: Robot is in `current_room`, `obj_name` is in gripper, gripper is not free
- Resulting World State: gripper is free, `obj_name` is in `current_room`, robot is in `current_room`

carry\_to(<robot\_loc>,<object\_name>,<object\_loc>,<destination\_loc>,<gripper\_tool>):

- Description: This function moves the robot from `robot_loc` to `object_loc`. Then it picks up `object_name` using `gripper_tool`, moves from `object_loc` to `destination_loc`, and drops the `object_name` using `gripper_tool`.
- Inputs: `robot_loc` (string): current location of the robot, `object_name` (string): name of the object to be carried, `object_loc` (string): current location of `object_name`, `destination_loc` (string): destination to move `object_name` to, `gripper_tool` (string): gripper to use to pick up and drop `object_name`
- Required World State: `gripper_tool` is free
- Resulting World State: `gripper_tool` is free, robot is at `destination_loc`, `object_name` is at `destination_loc`
- Basic Functions Used: `move`, `pick`, `drop`

move\_object(<robot\_loc>,<object\_name>,<object\_loc>,<destination\_loc>,<gripper\_tool>):

- Description: This function moves the robot from `robot_loc` to `object_loc`, picks up `object_name` using `gripper_tool`, moves from `object_loc` to `destination_loc` while carrying `object_name`, and drops `object_name` at the `destination_loc` from `gripper_tool`.
- Inputs: `robot_loc` (string): current location of the robot, `object_name` (string): name of the object to be moved, `object_loc` (string): current location of `object_name`, `destination_loc` (string): destination to move `object_name` to, `gripper_tool` (string): the left or right gripper to use
- Required World State: `gripper_tool` is free
- Resulting World State: `gripper_tool` is free, robot is at `destination_loc`, `object_name` is at `destination_loc`
- Basic Functions Used: `move`, `pick`, `drop`

Here are examples:

The goal is to satisfy the following conditions: ball1 is at rooma, ball2 is at roomb.

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at rooma. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at rooma. Room rooma Room roomb.

Action: move\_object(rooma, Ball2, rooma, roomb, 'left')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is at rooma. Room rooma Room roomb

Action: pickup(ball1,'rooma','right')

Observation: Ball1 is a ball. Ball1 is carrying right. Ball2 is at rooma. Ball2 is a ball. Ball2 is at rooma. Left is a gripper. Left is free. Right is a gripper. Robby is at rooma. Room rooma Room roomb

Action: drop(ball1,'rooma','left')

Observation: Ball1 is a ball. Ball1 is carrying right. Ball2 is a ball. Ball2 is carrying left. Ball3 is a ball. Ball3 is at rooma. Left is a gripper. Right is a gripper. Robby is at rooma. Room rooma Room roomb

Action: move('rooma','roomb')

Observation: Ball1 is a ball. Ball1 is carrying right. Ball2 is a ball. Ball2 is carrying left. Ball3 is a ball. Ball3 is at rooma. Left is a gripper. Right is a gripper. Robby is at roomb. Room rooma Room roomb

Action: drop(ball1,'rooma','right')

Observation: The action is not valid and therefore takes no effect. Please remember to satisfy the restriction of actions.

Action: drop(ball1,'roomb','right')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is carrying left. Ball3 is a ball. Ball3 is at rooma. Left is a gripper. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb

Action: drop(ball2,'roomb','left')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at rooma. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb

Action: move('roomb','rooma')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at rooma. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at rooma. Room rooma Room roomb

Action: pickup(ball3,'rooma','right')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is carrying right. Left is a gripper. Left is free. Right is a gripper. Robby is at rooma. Room rooma Room roomb

Action: move('rooma','roomb')

Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is carrying right. Left is a gripper. Left is free. Right is a gripper. Robby is at roomb. Room rooma Room roomb

Action: drop(ball3,'roomb','right')

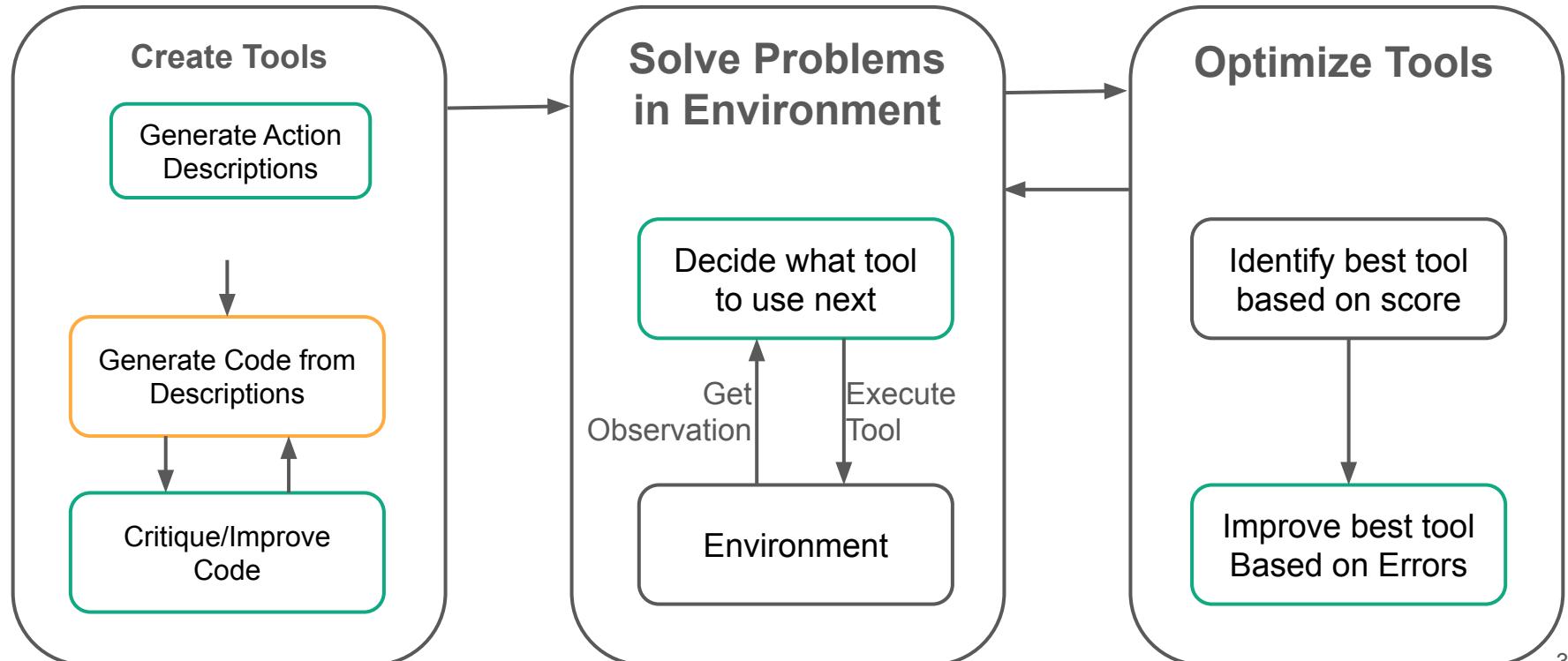
Observation: Ball1 is a ball. Ball1 is at roomb. Ball2 is a ball. Ball2 is at roomb. Ball3 is a ball. Ball3 is at roomb. Left is a gripper. Left is free. Right is a gripper. Right is free. Robby is at roomb. Room rooma Room roomb

Action:

| Model                       | Accuracy, higher is better |               |                                  |
|-----------------------------|----------------------------|---------------|----------------------------------|
|                             | HumanEval (pass@1)         | MBPP (pass@1) | Multilingual Human Eval (pass@1) |
| <b>Codex</b>                | 33.5                       | 45.9          | 26.1                             |
| <b>GPT 3.5</b>              | 48.1                       | 52.2          | -                                |
| <b>GPT 4 (reported)</b>     | 67.0                       | -             | -                                |
| <b>Palm-Coder</b>           | 36.0                       | 47.0          | -                                |
| <b>StarCoder Python</b>     | 33.6                       | 52.7          | 25.3                             |
| <b>StarCoder (prompted)</b> | 40.8                       | 49.5          | -                                |
| <b>Llama 2 (70B)</b>        | 30.5                       | 45.4          | 24.4                             |
| Code Llama                  | 7B                         | 33.5          | 41.4                             |
|                             | 13B                        | 36.0          | 47.0                             |
|                             | 34B                        | 48.8          | 55.0                             |
|                             | 70B                        | 53.0          | 62.4                             |
| Code Llama - Instruct       | 7B                         | 34.8          | 44.4                             |
|                             | 13B                        | 42.7          | 49.4                             |
|                             | 34B                        | 41.5          | 57.0                             |
|                             | 70B                        | 67.8          | 62.2                             |
| Code Llama - Python         | 7B                         | 38.4          | 47.6                             |
|                             | 13B                        | 43.3          | 49.0                             |
|                             | 34B                        | 53.7          | 56.2                             |
|                             | 70B                        | 57.3          | 65.6                             |

# Methods

█ GPT-3.5  
█ Code Llama



## Input

**Actions:** move(roomA, roomB), pickup(obj, room, gripper), drop(obj, room, gripper)

**Initial State:** Ball 1, 2, 3, 5 are in Room A, Ball 4, 6 are in Room B, Robot is in Room A, Right & Left Gripper are free

**Goal State:** Ball 4, 6 are in Room A

## Output

```
def move_objects(room1, room2, objects, gripper):
 for obj in objects:
 pickup(obj, room1, gripper)
 move(room1, room2)
 drop(obj, room2, gripper)
 move(room2, room1)
```

**Description:** This action moves all objects from one room to another using a gripper.

**Input:** room1 (string), room2 (string), objects (list): list of objects to move from room1 to room2, gripper (string)

**Required World State:** all objects must be in room1, the robot must be in room1, gripper must be free

**Resulting World State:** all objects are now in room2, the robot is in room1