
VOYAGER: An Open-Ended Embodied Agent with Large Language Models

**Guanzhi Wang^{1 2}, Yuqi Xie³, Yunfan Jiang^{4*}, Ajay Mandlekar^{1*},
Chaowei Xiao^{1 5}, Yuke Zhu^{1 3}, Linxi Fan^{1†}, Anima Anandkumar^{1 2†}**
¹NVIDIA, ²Caltech, ³UT Austin, ⁴Stanford, ⁵ASU

*Equal contribution †Equal advising
<https://voyager.minedojo.org>

Interpreted by:

John Tan Chong Min

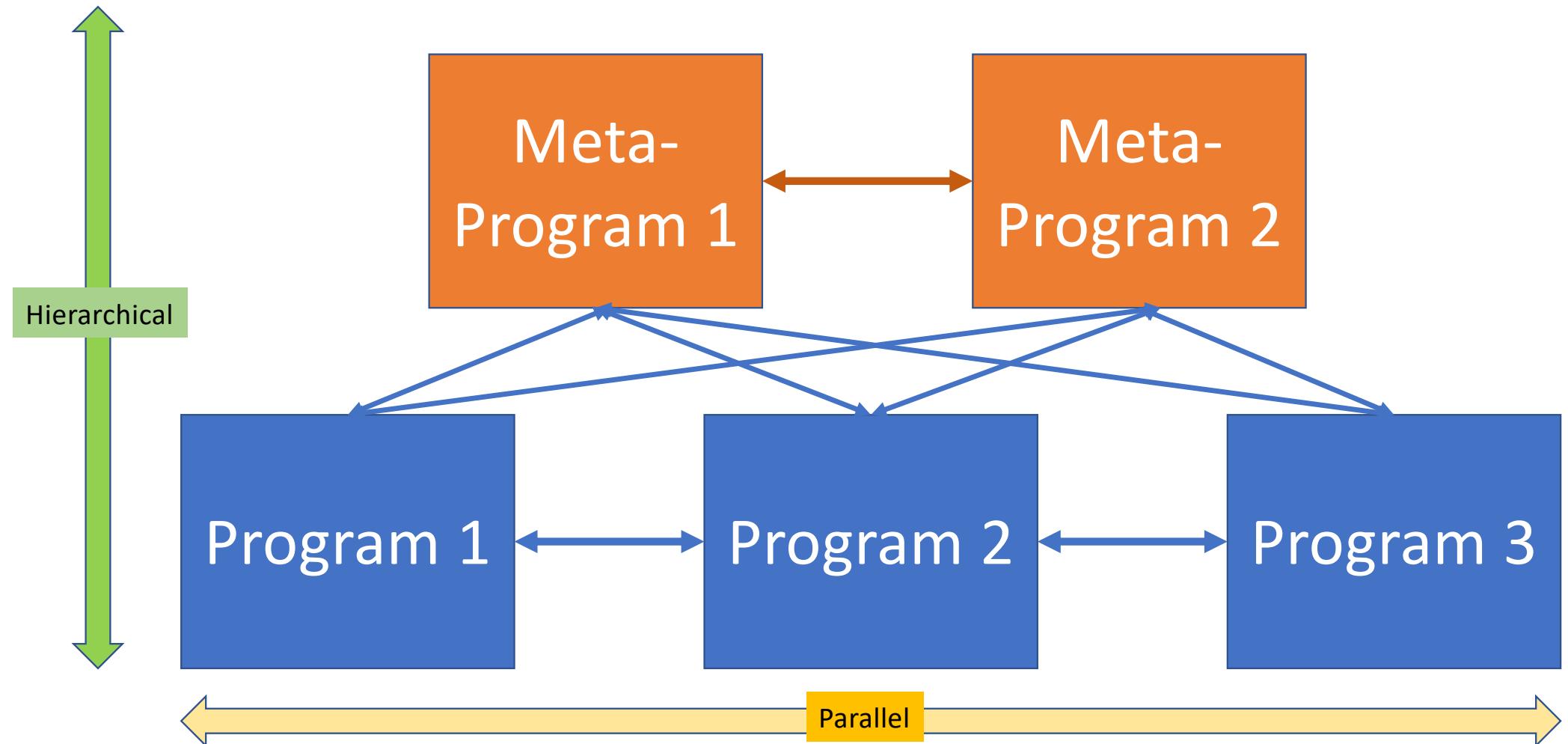
A New Way of doing RL

- Use LLM to decide actions (here given as skills via code) based on context
- Build upon a framework based on LLMOps which require just external database tuning without changing model weights

LLMOps

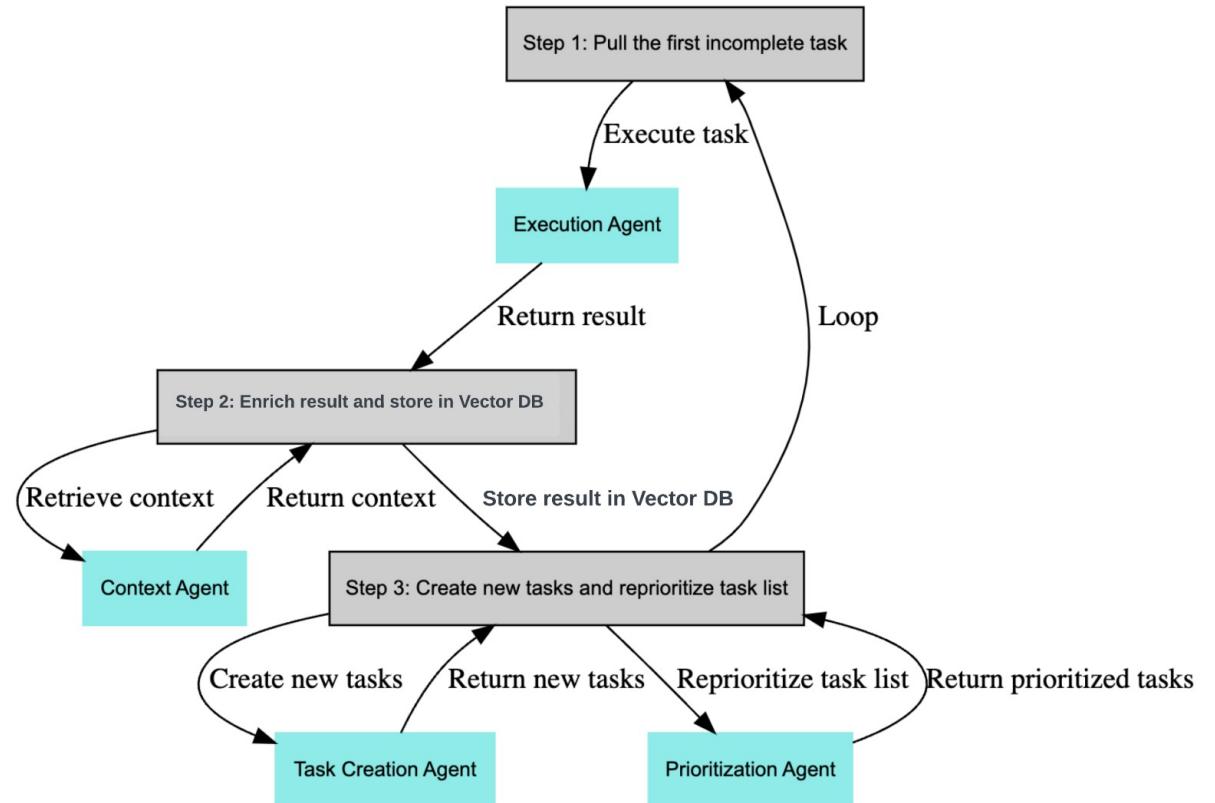
Integrating LLMs in a framework

Systems-level approach to LLMs



Example: BabyAGI / AutoGPT / AgentGPT

- Uses GPT4 to in a larger ecosystem to:
 - Create Tasks
 - Execute Tasks
 - Prioritize Tasks
- Uses memory to store and retrieve task/result pairs
 - Stored by vector embeddings



<https://github.com/yoheinakajima/babyagi>

Let's see it in action!

The screenshot shows the AgentGPT interface in Beta mode. At the top, it says "Assemble, configure, and deploy autonomous AI Agents in your browser." Below the header, there's a section titled "AgentGPT-3.5 (3 / 5 runs)" which lists the following tasks:

- ★ Embarking on a new goal: Create a comprehensive report of the Nike company
- Thinking... (Restart if this takes more than 30 seconds)
- ★ Added task: Gather financial statements and annual reports of Nike from the past few years
- ★ Added task: Conduct a SWOT analysis of the company
- ★ Added task: Research and analyze Nike's market share and competition
- ★ Added task: Write a comprehensive report summarizing all findings
- Thinking... (Restart if this takes more than 30 seconds)

At the bottom of this section is a "Pause Mode" toggle switch. To the right, there's a "Current tasks" sidebar with four items:

- Gather financial statements and annual reports of Nike from the past few years
- Conduct a SWOT analysis of the company
- Research and analyze Nike's market share and competition
- Write a comprehensive report summarizing all findings

Below the sidebar are "Custom task" and "Add" buttons. At the very bottom, there are fields for "Name" (set to "ResearchGPT") and "Goal" (set to "Create a comprehensive report of the Nike company").

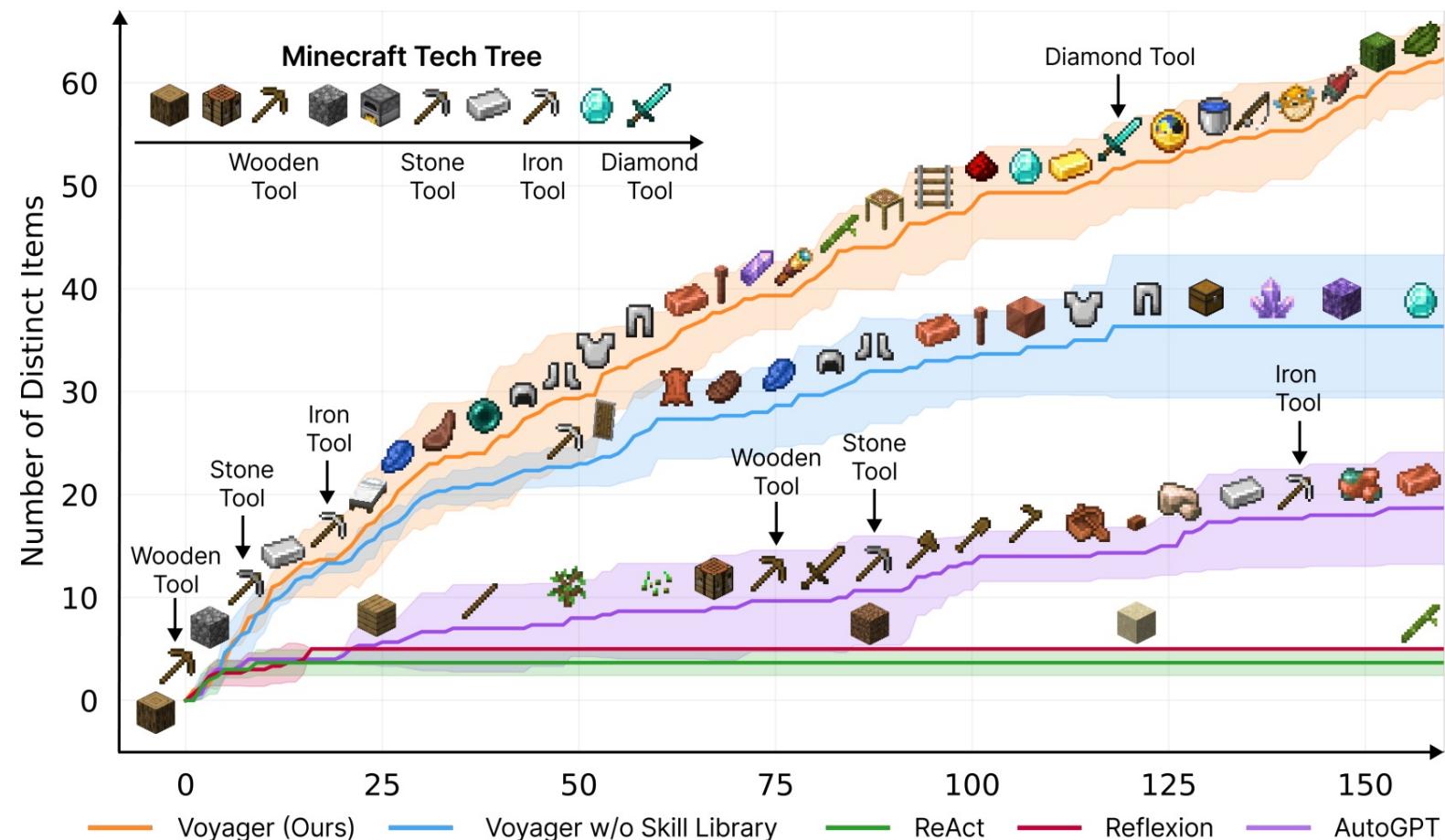
<https://agentgpt.reworkd.ai/>

VOYAGER: An Open-Ended Embodied Agent with Large Language Models

Better than AutoGPT...

Wait, what??!

Voyager has the best performance out of LLM-based methods



Components of Voyager

Teacher comes up with something manageable for student

Automatic Curriculum

Student learns (multiple tries)

Iterative Prompting Mechanism

Student remembers (only good stuff)

Skill Library

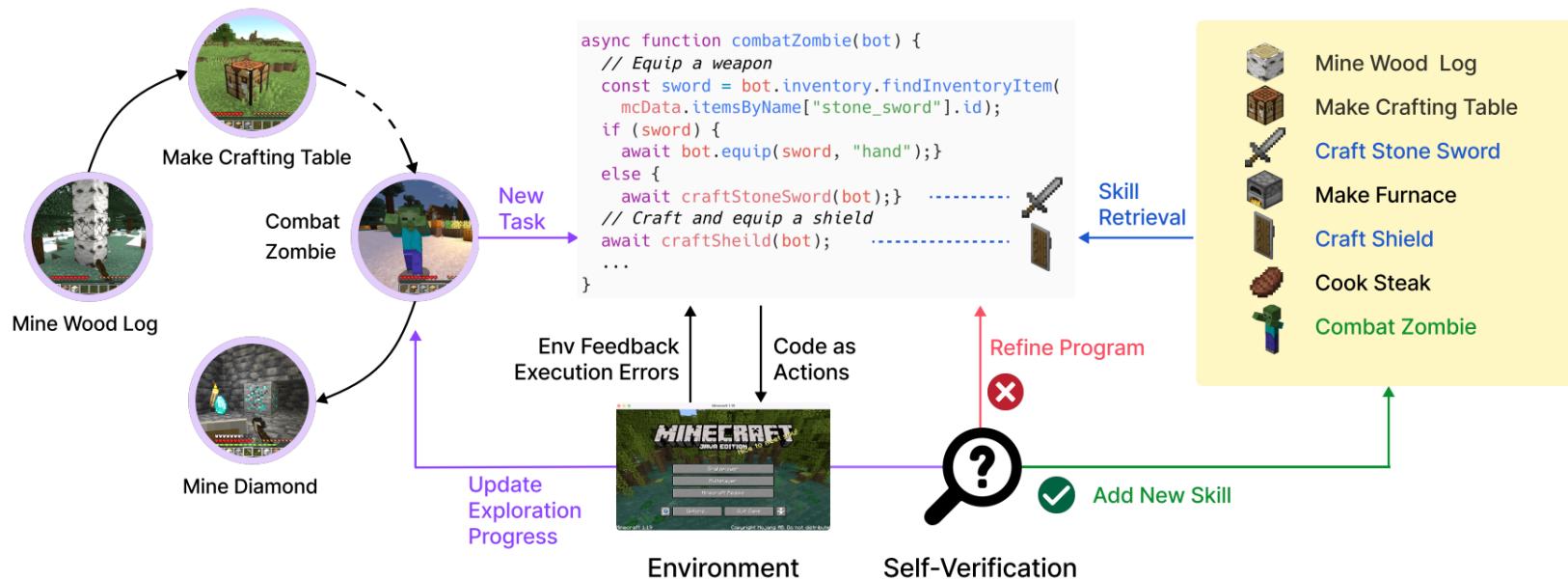


Figure 2: VOYAGER consists of three key components: an automatic curriculum for open-ended exploration, a skill library for increasingly complex behaviors, and an iterative prompting mechanism that uses code as action space.

Automatic Curriculum Generator

The teacher

Automatic Curriculum Generator

- Generated by GPT-4 based on the overarching goal of “discovering as many diverse things as possible”



Inventory (5/36): {'oak_planks': 3, 'stick': 4, 'crafting_table': 1, 'stone': 3, 'wooden_pickaxe': 1}

GPT-4

Reasoning: Since you have a wooden pickaxe and some stones, it would be beneficial to upgrade your pickaxe to a stone pickaxe for better efficiency.
Task: Craft 1 stone pickaxe.



Biome: river

Inventory (4/36): {'sugar_cane': 3, 'paper': 3, 'fishing_rod': 1, 'wooden_pickaxe': 1}

GPT-4

Reasoning: Since you have a fishing rod in your inventory and you are near a river biome, it would be a good opportunity to catch some fish for food and experience.
Task: Catch 1 fish.



Nearby entities: pig, cat, villager

Health: 12/20

Hunger: 0/20

GPT-4

Reasoning: Your hunger is at 0, which means you need to find food to replenish your hunger. Since there are pigs nearby, you can kill one to obtain raw porkchops.
Task: Kill 1 pig.



Inventory (6/36): {'furnace': 1, 'stone_pickaxe': 1, 'oak_planks': 7, 'crafting_table': 1, 'raw_iron': 4, 'coal': 1}

GPT-4

Reasoning: You have raw iron and coal, and you have a furnace. It's time to smelt the iron to make iron ingots, which can be used to craft better tools and armor.
Task: Smelt 4 raw iron.



Time: night

Nearby entities: zombie

Equipment: 'stone_sword', 'shield'

GPT-4

Reasoning: Since it's night and there's a zombie nearby, it's a good opportunity to try killing the zombie now that you have a stone sword and a shield equipped.
Task: Kill 1 zombie.

Prompt for Teacher: Curriculum Generator

- **Overall prompt encouraging diverse behaviors and imposing constraints**
 - “My ultimate goal is to discover as many diverse things as possible ... The next task should not be too hard since I may not have the necessary resources or have learned enough skills to complete it yet.”
- **The agent's current state:** Inventory, equipment, nearby blocks and entities, biome, time, health and hunger bars, and position
- **History:** Previously completed and failed tasks, reflecting the agent's current exploration progress and capabilities frontier
- **Memory retrieval:** GPT-3.5 to self-ask questions based on the agent's current state and exploration progress and self-answer questions with a wiki knowledge base to provide additional context to GPT-4.

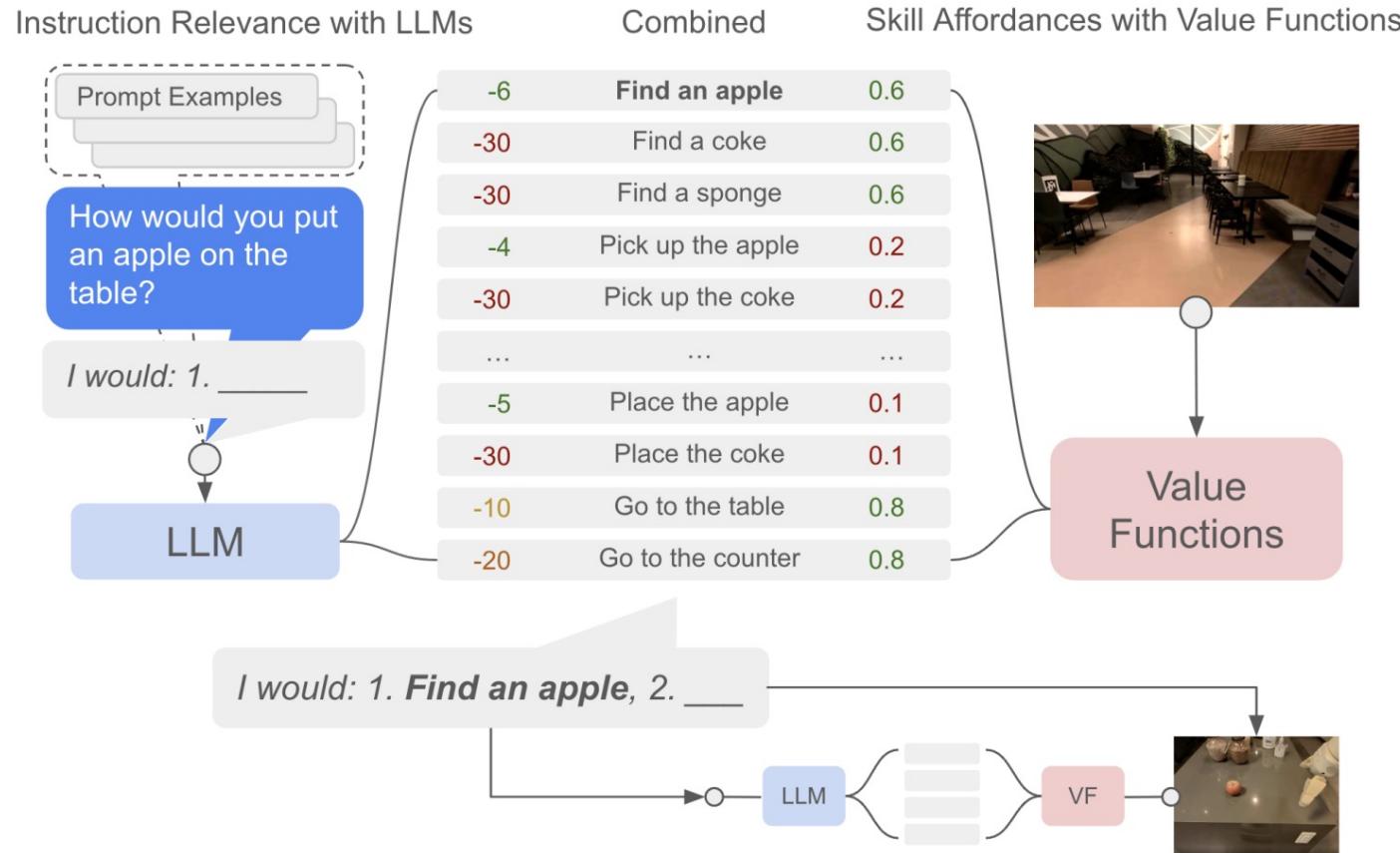
Iterative Prompting

Now that teacher has come up with a curriculum, student needs to learn from it

Problems with GPT generation

- Lacks grounding with the environment
- May not know what are the functions available to use in the javascript code for Minecraft
- Code may not lead to completion of task

Previously: Value Function Grounding



- This is a **slow way of learning** as value functions take a long time to converge
- Encoding the environment feedback via text may be more natural

Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. Ahn et al. 2022

Iterative Prompting : Give the agent environment and code feedback!

- **Environment Feedback**
 - `bot.chat()` inside control primitive APIs to generate environment feedback and prompt GPT-4 to use this function as well during code generation
- **Code Feedback**
 - Execution errors Generated by program interpreter that reveals any invalid operations or syntax errors in programs
- **Self-verification for checking task success**
 - Another GPT-4 agent for self-verification. Takes in current state and task, and outputs if program achieves the task.
 - Gives critique (constructive feedback) if unable to achieve.
 - **(Potentially limited) GPT-4 is still unable to simulate multiple time steps well without using the environment engine**

Actor: RL Loop now via past-history prompting!

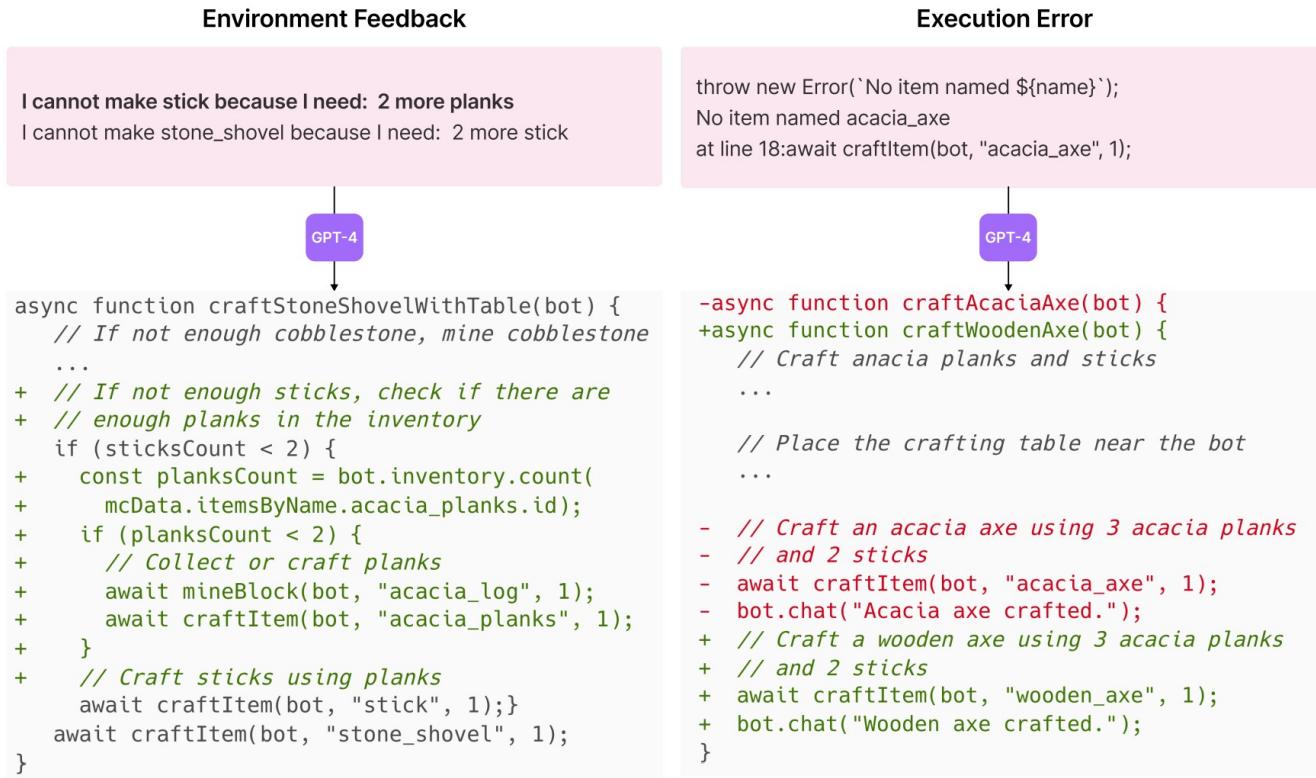
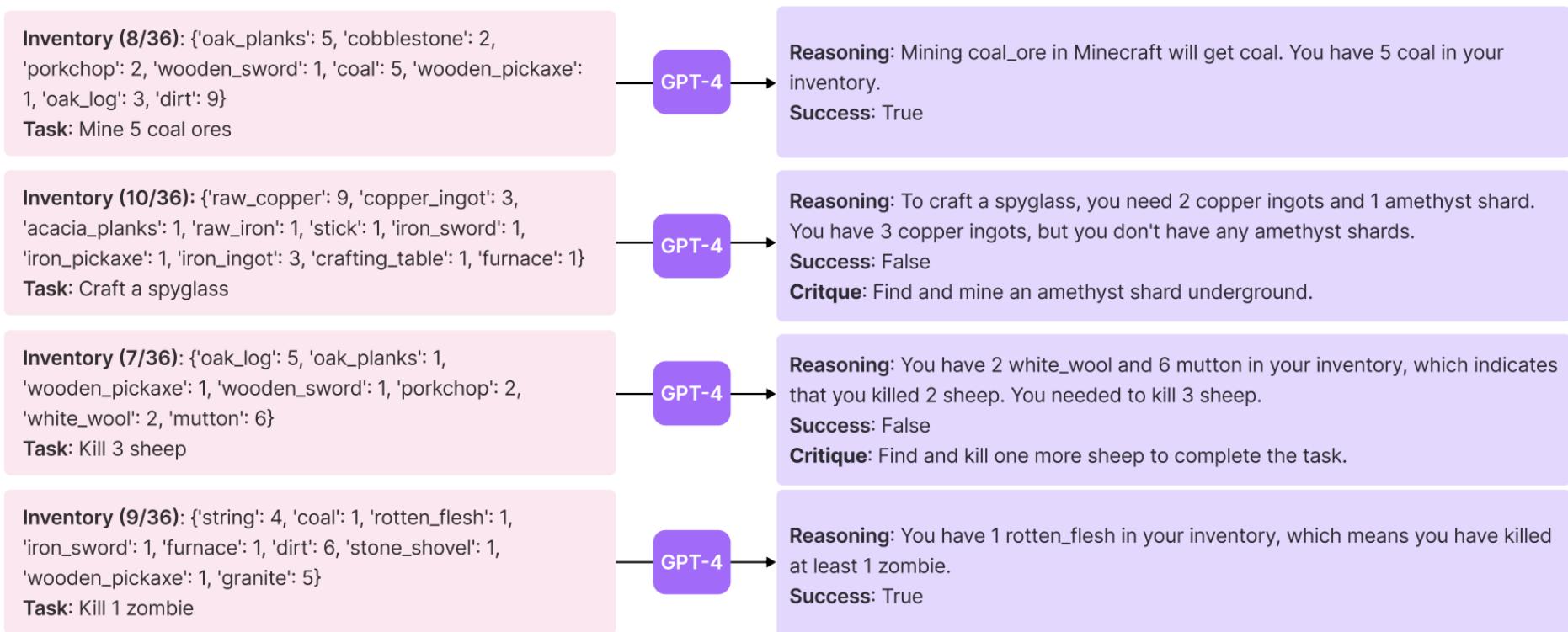


Figure 5: **Left: Environment feedback.** GPT-4 realizes it needs 2 more planks before crafting sticks. **Right: Execution error.** GPT-4 realizes it should craft a wooden axe instead of an acacia axe since there is no acacia axe in Minecraft. We only display the partial prompt for brevity. The full prompt structure for code generation is in Appendix, Sec. A.4.

- Mitigates lack of grounding with environment
- Interpreter solves to a limited extent GPT not knowing which functions are available

Critic: Using GPT4 to verify if task is completed



Does GPT4 already have the answers?

- Prompt: “Generate me the Minecraft Java code to craft a diamond tool using the Mineflayer API”

```
// Determine the recipe
const recipe = bot.recipesFor(Item.toNotch('diamond_pickaxe'))[0];

if (!recipe) {
    console.log("I don't know how to craft that!");
    return;
}
```

Does GPT4 already have the answers (CoT)?

- Prompt: “How do we generate a diamond tool in Minecraft?”
- Followed by “Generate me the Minecraft Java code to craft a diamond tool using the Mineflayer API”

```
if (recipe) {  
    bot.chat(`Crafting ${itemName}`);  
    bot.craft(recipe, 1, craftingTable, (err) => {  
        if (err) bot.chat(err.message);  
        else bot.chat(`Crafted ${itemName}`);  
    });  
} else {  
    bot.chat(`No recipe for ${itemName}`);  
}
```

Skill Generation

Student can learn better using past successful memories

Skill as Code

- Represent each skill with **executable code** that scaffolds temporally extended actions for completing a specific task proposed by the automatic curriculum



GPT Prompt for Skill Generation

- **Overall prompt to encourage generic skills:** “Your function will be reused for building more complex functions. Therefore, you should make it generic and reusable.”
- **Tool Description:** Control primitive APIs, and relevant skills retrieved from the skill library
- **History:** The generated code from the last round, environment feedback, execution errors, and critique, based on which GPT-4 can self-improve
- **Agent information:** The agent’s current state, including inventory, equipment, nearby blocks and entities, biome, time, health and hunger bars, and position
- **Chain-of-thought prompting** to do reasoning before code generation.

Skill Library (aka Memory)

Program Generated by GPT-4

```
async function combatZombie(bot) {  
    // Equip a weapon  
    const sword =  
    bot.inventory.findInventoryItem(  
        mcData.itemsByName[  
            "stone_sword"  
        ].id  
    );  
    if (sword) {  
        await bot.equip(sword, "hand");  
    } else {  
        await craftStoneSword(bot);  
    }  
    ...  
    // Craft and equip a shield  
    ...  
    // Recover hunger  
    ...  
    // Look for and combat a zombie  
    ...  
}
```

Program Description

```
async function combatZombie(bot) {  
    // The function is about  
    equipping a stone sword to combat  
    a zombie. If a stone sword is not  
    found, it will craft one.  
    Additionally, it crafts and equips  
    a shield for added protection.  
    Afterwards, it proceeds to cook  
    sticks in order to restore hunger.  
    Once hunger is replenished, it  
    actively searches for a zombie and  
    engages in combat with it.  
}
```

GPT-3.5

Embedding

Key

Value

Skill Library



Task: Craft Iron Pickaxe

How to craft an iron pickaxe in Minecraft?

GPT-3.5

To craft an iron pickaxe, you
need to 3 iron ingots and 2
sticks. Once you have gathered
the materials,

Environment Feedback

Embedding

Top-5 Relevant Skills



Query

Retrieve

Skill Library

- **Storage:** Skill stored with summarized program description text embedding as key, program code as value
- **Retrieval:** Task converted into description taking into account environment feedback, use embedding to find closest match to skills

Results

Let's analyze why the results are better

Baselines

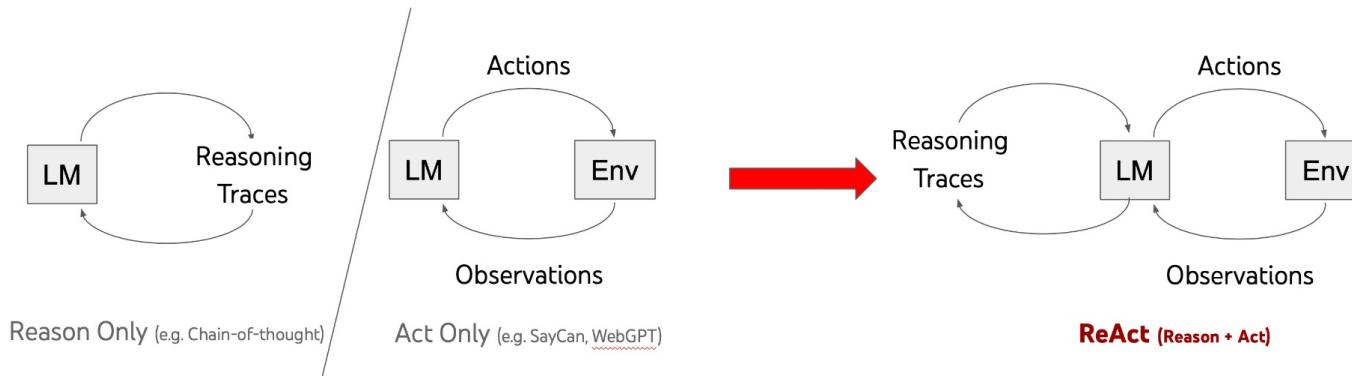


Image from: <https://react-lm.github.io/>

- **ReAct**: Chain of thought prompting plus environment grounding.
 - Thought, Action, Observation
- **Reflexion**: ReAct plus “Reflect on your answer”
- **AutoGPT**: Sub-task decomposition in a ReAct-like framework

Tech Tree Mastery

Table 1: Tech tree mastery. Fractions indicate the number of successful trials out of three total runs. 0/3 means the method fails to unlock a level of the tech tree within the maximal prompting iterations (160). Numbers are prompting iterations averaged over three trials. The fewer the iterations, the more efficient the method.

Method	Wooden Tool	Stone Tool	Iron Tool	Diamond Tool
ReAct [29]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
Reflexion [30]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28]	92 ± 72 (3/3)	94 ± 72 (3/3)	135 ± 103 (3/3)	N/A (0/3)
VOYAGER w/o Skill Library	7 ± 2 (3/3)	9 ± 4 (3/3)	29 ± 11 (3/3)	N/A (0/3)
VOYAGER (Ours)	6 ± 2 (3/3)	11 ± 2 (3/3)	21 ± 7 (3/3)	102 (1/3)

Map Exploration

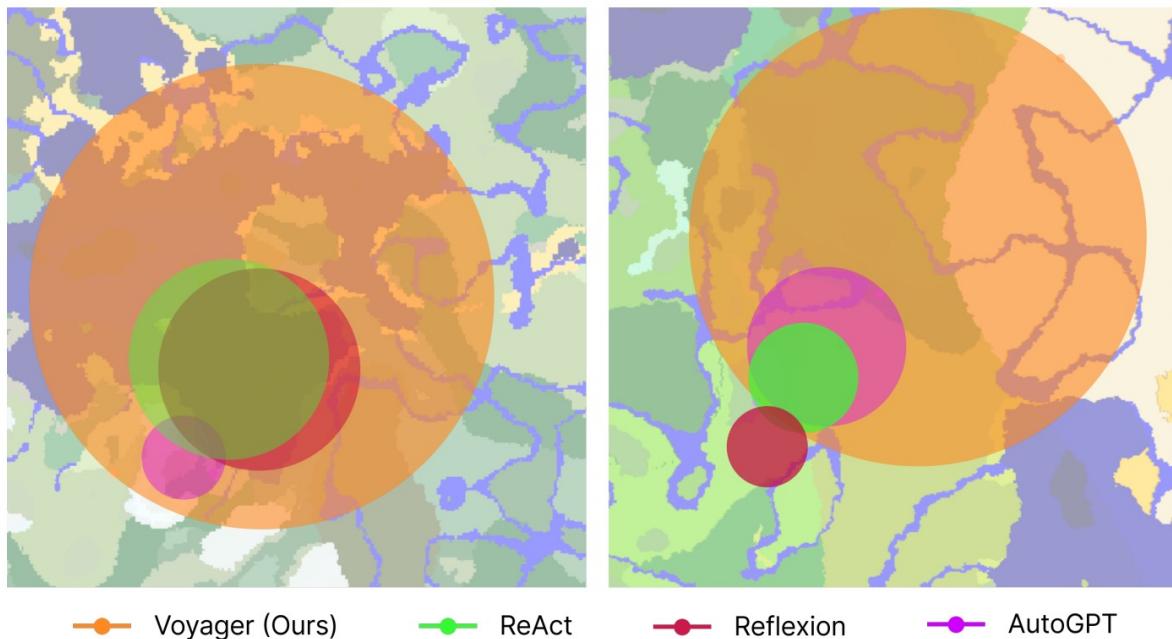


Figure 7: Map coverage: bird’s eye views of Minecraft maps. VOYAGER is able to traverse $2.3\times$ longer distances compared to baselines while crossing diverse terrains.

Zero-shot generalization to unseen tasks

- Skill library (aka memory) makes a huge difference here

Table 2: Zero-shot generalization to unseen tasks. Fractions indicate the number of successful trials out of three total attempts. 0/3 means the method fails to solve the task within the maximal prompting iterations (50). Numbers are prompting iterations averaged over three trials. The fewer the iterations, the more efficient the method.

Method	Diamond Pickaxe	Golden Sword	Lava Bucket	Compass
ReAct [29]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
Reflexion [30]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28] w/ Our Skill Library	39 (1/3)	30 (1/3)	N/A (0/3)	30 (2/3)
VOYAGER w/o Skill Library	36 (2/3)	30 ± 9 (3/3)	27 ± 9 (3/3)	26 ± 3 (3/3)
VOYAGER (Ours)	19 ± 3 (3/3)	18 ± 7 (3/3)	21 ± 5 (3/3)	18 ± 2 (3/3)

Conclusion

- GPT learns very fast: it is very good at zero-shot description-based learning, and few-shot example-based learning
- Imbuing GPT prompts with in-game context can help ground it better
 - Having a feedback loop to the environment via code can help
- Having external memory in the form of code-blocks of skills can help improve performance

Critique (my view)

- Prompts are too hand-tuned (look at Appendix). It cannot generalize beyond Minecraft. Perhaps we can have retrieval-augmented generation so that the prompts can be made more generic and augmented with domain-specific knowledge
- Agent does not start from scratch here (MineFlayer API has some basic functions like movement). We can perhaps also learn the basic level directly via an embedding method like Decision Transformers.
- (In paper) Can combine with visual models to augment more information to GPT-4
 - My view: No need to have visual models as it is too high-dimensional. Need to better improve vision-to-text models like BLIP for text input.

Potential Improvements (my view)

- Learning from expert trajectories
 - Humans also learn from each other, especially the best performing one
 - Helps a lot to have model answer (refer to synthetic examples in AlphaTensor / HyperTree Proof Search)
- Prompting from broad to specific actions might help better with hierarchical planning
- Critique can be given the environment (something like world model) to evaluate code

Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory

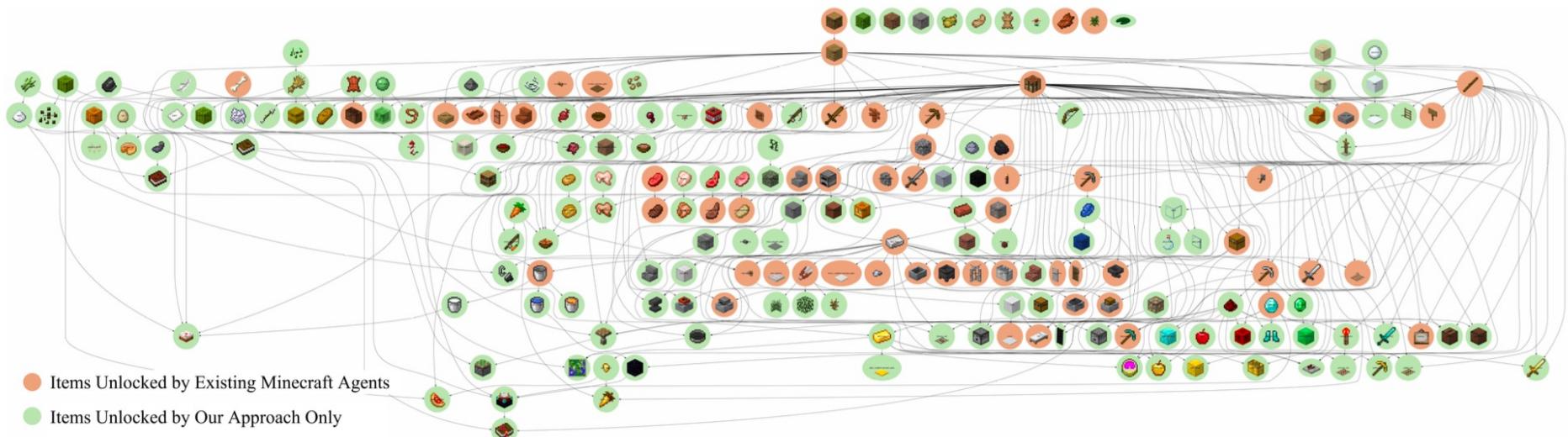
Xizhou Zhu^{1,2*}, Yuntao Chen^{3*}, Hao Tian^{2*}, Chenxin Tao^{1,2*}, Weijie Su^{2,4*}, Chenyu Yang^{1*}, Gao Huang¹, Bin Li⁴, Lewei Lu², Xiaogang Wang^{2,5}, Yu Qiao⁶, Zhaoxiang Zhang⁷, Jifeng Dai^{1,6✉}

Interpreted by:
John Tan Chong Min



Can do hierarchical decomposition pretty well!

- Unlocks all craftable items! (that don't require special preparation)



LLM is better than RL (even specialized ones)

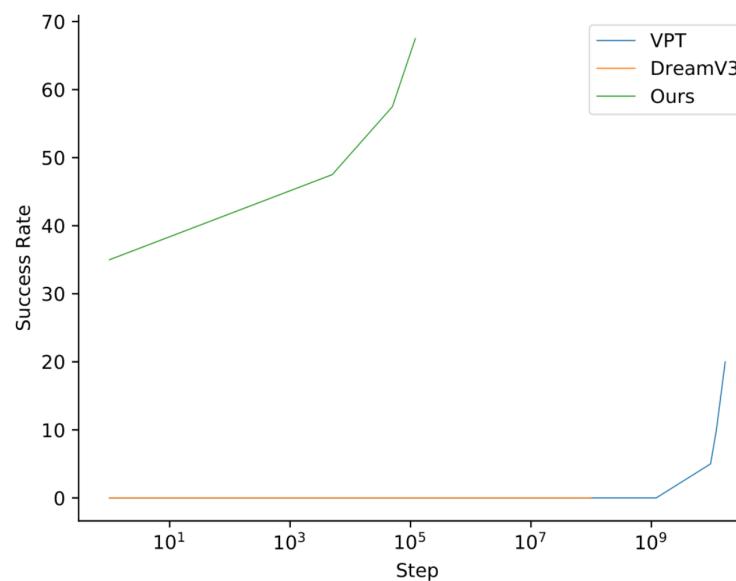
Table 5: **Ablation study.** The milestone items from left to right are crafting table , wooden pickaxe , stone pickaxe , iron pickaxe , and diamond . The success rate is calculated under time limit of 12000 steps (total) and query limit of 30 (each sub-goal). “Goal Decomp.” indicates whether to use LLM Decomposer to decompose the goal into sub-goals. “Goal / Sub-Goal to Structured Actions / Keyboard & Mouse Mapping” indicates which method is used for the mapping from goal / sub-goals to structured actions / keyboard & mouse operations.

Goal Decomp.	Structured Action	Goal / Sub-Goal to Structured Actions / Keyboard & Mouse Mapping	Success Rate (%)				
							
(a)		Specialist RL Model (VPT)	100.0	100.0	100.0	85.0	20.0
(b)		Goal-conditioned RL Model (DEPS)	0.0	0.0	0.0	0.0	0.0
(c)		Our LLM Planner	0.0	0.0	0.0	0.0	0.0
(d)	✓	Goal-conditioned RL Model (DEPS)	90.0	80.0	30.0	0.0	0.0
(e)	✓	Our LLM Planner	0.0	0.0	0.0	0.0	0.0
(f)	✓	Our LLM Planner	57.5	32.5	5.0	0.0	0.0
(g)	✓	Our LLM Planner	100.0	100.0	100.0	95.0	67.5

Much better learning than RL methods

- As a side note, I always thought RL methods and learning world models are sample inefficient. LLM-powered semantic learning is the way to go.

Learning Efficiency



GITM only requires a single CPU node with 32 cores for training. Compared with 6,480 GPU days of OpenAI's VPT and 17 GPU days of DeepMind's DreamerV3, GITM improves the efficiency by at least 10,000 times.

Can create Nether Portal without Human Feedback!

- Uses only ChatGPT! (not GPT4), and can construct advanced objects
- In this aspect it is better than Voyager already

E Applications



(a) Shelter with Farmland



(b) Iron Golem



(c) Redstone Circuit



(d) Nether Portal

Figure 8: Demonstration of the applications. GITM can construct Shelter with Farmland and Iron Golem for survival, Redstone Circuit for automation equipment, and Nether Portal for the Nether world exploration.

Comparison with RL methods

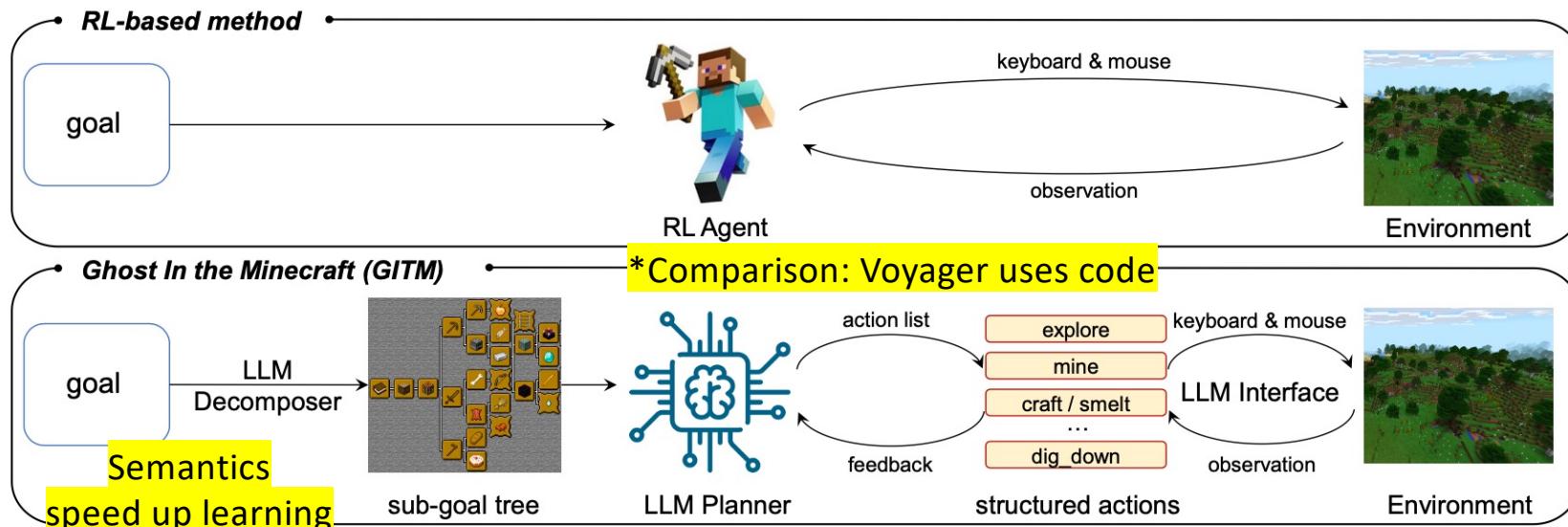


Figure 2: **Comparison between RL-based method and our GITM.** RL agents try to map an complex goal directly to a sequence of low-level control signals, while our GITM leverages LLM to break down the goals and map them to structured actions for final control signals.

Overall Flow

- **LLM Decomposer** first decomposes a goal in Minecraft into a series of well-defined sub-goals
- **LLM Planner** then plans a sequence of structured actions for each sub-goal. Memory is used for redoing performant actions.
- **LLM Interface** executes the structured actions to interact with the environment by processing raw keyboard/mouse input and receiving raw observations.

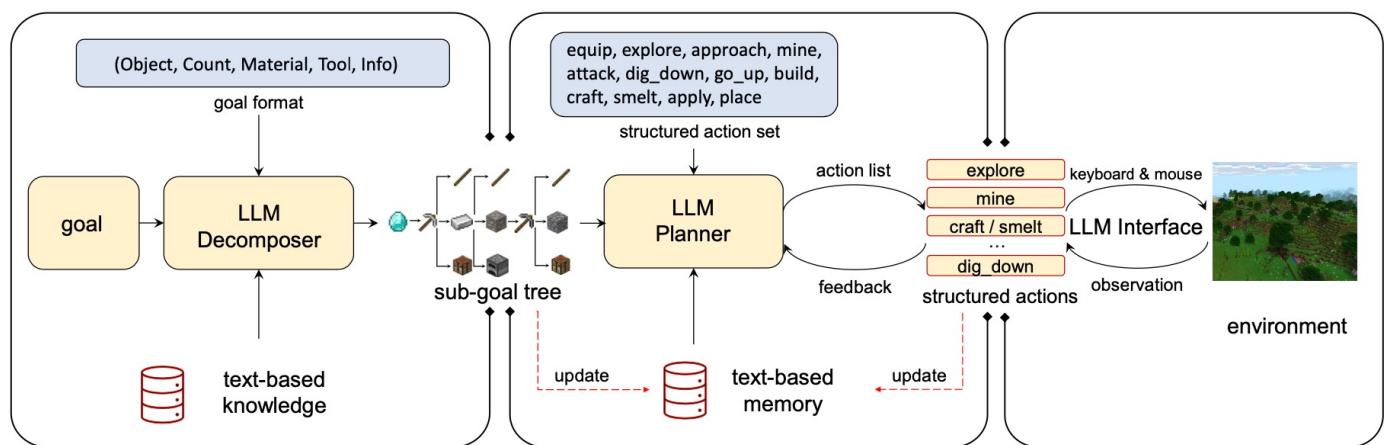
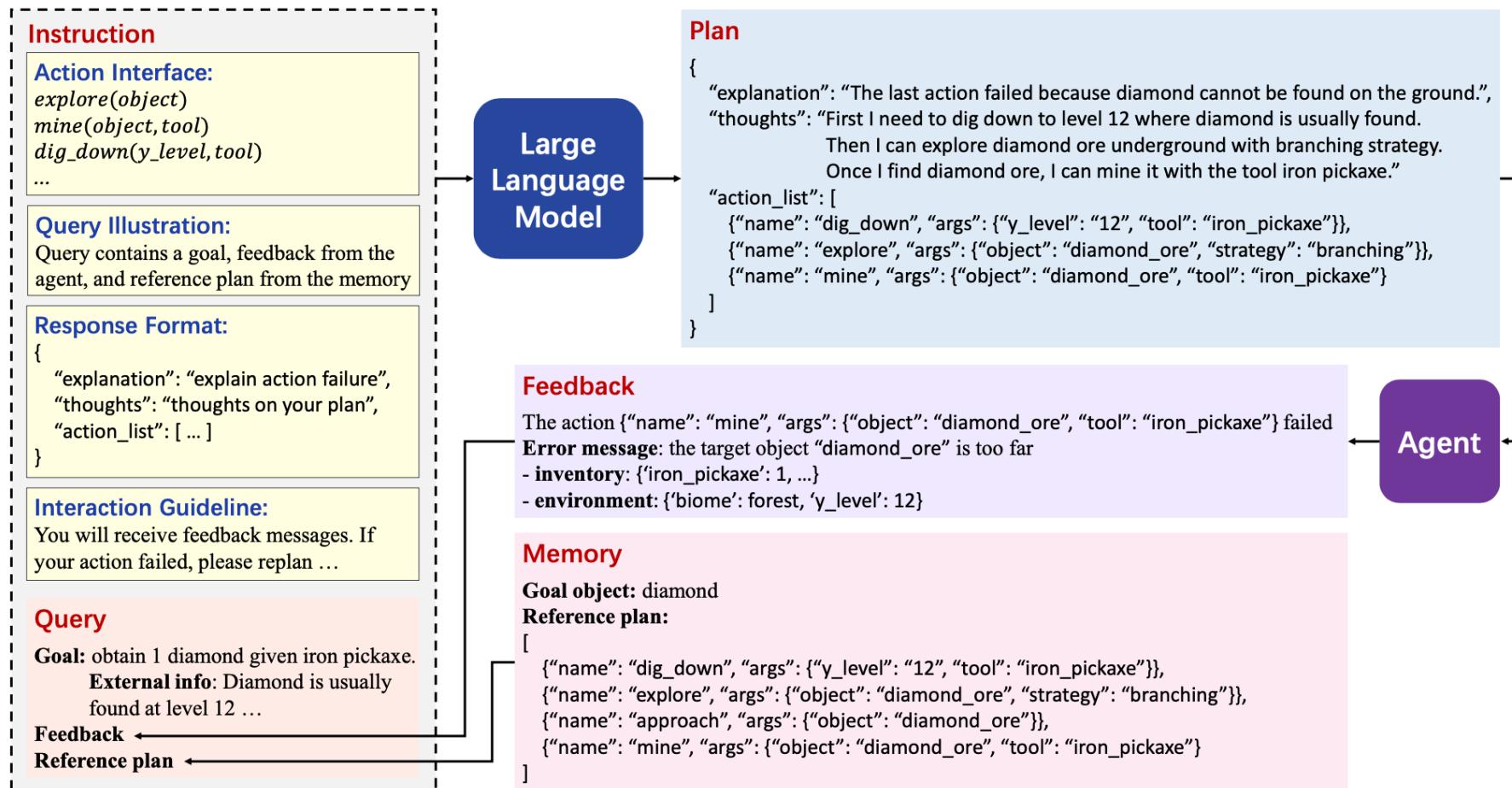


Figure 3: **Overview of our GITM.** Given a Minecraft goal, the LLM Decomposer divides the goal into a sub-goal tree. The LLM Planner then plans an action sequence for each sub-goal. Finally, the LLM Interface executes each action in the environment. Our LLM-based agents can be further enhanced by leveraging text-based knowledge and memory.

Detailed Process for LLM Planner



Subgoal Decomposition

- Almost all answers are already given in Wiki! Given directly to LLM Decomposer
- Same as Voyager, but in a more succinct and formatted structure! (Good!)
- LLM Decomposer takes in Target object quantity and name, and knowledge base, and comes up with the following format:

```
{  
  "object": "wooden_pickaxe",  
  "count": 1,  
  "material": { "planks": 3, "stick": 2},  
  "tool": "crafting_table",  
  "info": "wooden pickaxe can be crafted with 3 planks and 2 stick as the material and  
  crafting table as the tool."  
}
```

Action Primitives from Predefined Tasks

- First ask LLM to generate a tree-structured action planning for each of the 3141 predefined tasks provided by MineDojo, and then converts each action step into a (verb, object, tool, material) tuple
- Extract the structured actions by selecting frequent actions and merging actions with similar functionalities
- Final set of structured actions is {equip, explore, approach, mine/attack, dig_down, go_up, build, craft/smelt, apply}

Broad-Specific Action Generation

- This is how I would do it too😊 - Broad, then specific

SYSTEM:

You serve as an assistant that helps me play Minecraft.

I will give you my goal in the game, please break it down as a tree-structure plan to achieve this goal.

The requirements of the tree-structure plan are:

1. The plan tree should be exactly of depth 2.
2. Describe each step in one line.
3. You should index the two levels like '1.', '1.1.', '1.2.', '2.', '2.1.', etc.
4. The sub-goals at the bottom level should be basic actions so that I can easily execute them in the game.

USER:

The goal is to {goal description}. Generate the plan according to the requirements.

Convert each action description into domain-specific action

SYSTEM:

You serve as an assistant that helps me play Minecraft.

I will give you a sentence. Please convert this sentence into one or several actions according to the following instructions.

Each action should be a tuple of four items, written in the form ('verb', 'object', 'tools', 'materials')

'verb' is the verb of this action.

'object' refers to the target object of the action.

'tools' specifies the tools required for the action.

'material' specifies the materials required for the action.

If some of the items are not required, set them to be 'None'.

USER:

The sentence is {sentence}. Generate the action tuple according to the requirements.

Action List Generation

- (+) Constraining the possible actions improves performance
- (-) No referencing of skill library / no more learning of primitive actions possible
- (+) Code will always run!

5. You must describe your thoughts on the plan in natural language at the beginning. After that, you should write all the actions together. The response should follow the format:

```
{  
    "explanation": "explain why the last action failed, set to null for the first planning",  
    "thoughts": "Your thoughts on the plan in natural languag",  
    "action_list": [  
        {"name": "action name", "args": {"arg name": value}, "expectation": "describe the  
expected results of this action"},  
        {"name": "action name", "args": {"arg name": value}, "expectation": "describe the  
expected results of this action"},  
        {"name": "action name", "args": {"arg name": value}, "expectation": "describe the  
expected results of this action"}  
    ]  
}
```

Observation Space

- The observation of the action contains **LiDAR rays** with an interval of 5 degrees in the horizon and vertical direction for locating objects, and **voxels with 10 units radius** only for navigation, inventory, life status, and agent location status
- Used as observational inputs to perform primitive actions

Training Process (somewhat like Curriculum Planner?)

- LLM Decomposer generate sub-goal trees for all items, acquiring the set of all sub-goals involved.
- Then for each sub-goal, the LLM planner plays multiple times given its prerequisites including the required materials and tools.
- The learning process of the sub-goal is finished once we obtain $N = 5$ successful action sequences and summarize them into one common plan solution for reference.
 - Summarization takes multiple action lists and identifies only the key ones needed to perform action (some memory consolidation process)

Memory enables sequential hierarchical learning!

- **Without prior experience:** When encountering a new sub-goal that is not in the memory, the LLM planner creates plans without reference. Once the sub-goal is achieved, the entirely executed action sequence would be stored into the memory
- **With prior experience:** When encountering a sub-goal with memory, the first action sequence in the recording list for this goal is retrieved as the reference plan, with which the LLM planner tries to achieve the goal. If it succeeds, the new executed action sequence will be added to the last of the recording list

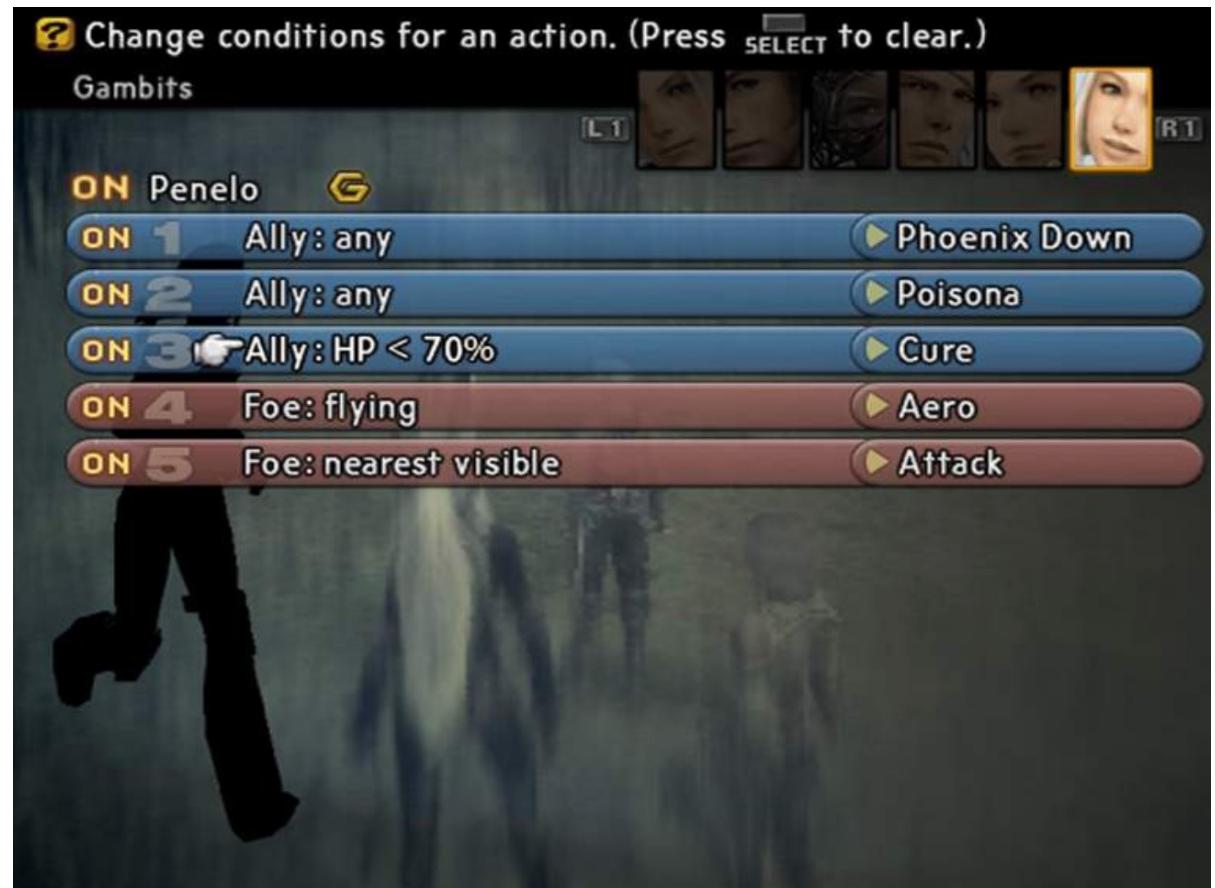
Thoughts

- This architecture is more generic than Voyager
- Primitive actions can be built by considering a baseline set of tasks for any environment
- No learning of new functions may restrict adaptability, though if all new functions can be formed from primitive actions, then maybe memory can be good enough!
- Using LIDAR rays to describe environment is a good way to do it (RGB 3D input is just too high-dimensional)
- So far model answer is given to the agent! What if agent needs to find out what to do purely through exploration???

What should action space be?

A new type of code

- Runs all the time from highest to lowest priority
- Will not have runtime errors
- Can potentially be hierarchical if it calls another function as the outcome



Final Fantasy XII, Gambit System

Possible Code

- So a navigation code could look like this: Run from top to bottom
 - Input: State x, y, Goal gx, gy
 - Sensory-based functions: check_right(), check_left(), check_up(), check_down()
 - Movement-based functions: move_right(), move_left(), move_up(), move_down()
 - Memory-based functions: hasvisitedcell(), choose_next_cell_by_least_recent_visit()
 - If hasvisitedcell(): choose_next_cell_by_least_recent_visit()
 - If $x < gx$ and check_right() is not obstacle: move_right()
 - If $x > gx$ and check_left() is not obstacle: move_left()
 - If $y < gy$ and check_up() is not obstacle: move_up()
 - If $y > gy$ and check_down() is not obstacle: move_down()
- In this way, we can just ask the LLM to come up with instructions of the following nature:
 - If {one or more conditions}: {function call}
- The lowest-level functions will have to be given, with their descriptions made known to the model, and the LLM will have to decide what to use for compositionality, much like Voyager
- Only difference is that the function generated will be guaranteed to run with this kind of condition-outcome structure.

Questions to Ponder

- Curriculum Generation: How does GPT know what are the tasks to create given the inventory, nearby entities and equipment?
- Is there a better way to represent skills other than through code? Should skills be represented as a sequence of primitive actions (Ghost in the Minecraft paper)?
- Can all action spaces be mapped into code? What are the pros and cons of denoting actions as code?
- Should memory be purely just skills alone? Should there be memory at more levels?

Extra Information

Voyager Pseudocode

Pseudocode 1: VOYAGER algorithm.

```
def voyager(
    environment, # environment that uses code as action space
    curriculum_agent, # curriculum agent for proposing the next task
    action_agent, # action agent for code generation
    critic_agent, # critic agent for self-verification
    skill_manager, # skill manager for adding new skills and skill
    retrieval
):
    agent_state = environment.reset()
    while True:
        exploration_progress = (
            curriculum_agent.get_exploration_progress(
                curriculum_agent.get_completed_tasks(),
                curriculum_agent.get_failed_tasks(),
            )
        )
        task = curriculum_agent.propose_next_task(
            agent_state, exploration_progress
        )
        code = None
        environment_feedback = None
        execution_errors = None
        critique = None
        success = False
```

Voyager Pseudocode

```
# try at most 4 rounds before moving on to the next task
for i in range(4):
    skills = skill_manager.retrieve_skills(
        task, environment_feedback
    )
    code = action_agent.generate_code(
        task,
        code,
        environment_feedback,
        execution_errors,
        critique,
        skills,
    )
    (
        agent_state,
        environment_feedback,
        execution_errors,
    ) = environment.step(code)
    success, critique = critic_agent.check_task_success(
        task, agent_state
    )
    if success:
        break
if success:
    skill_manager.add_skill(code)
    curriculum_agent.add_completed_task(task)
else:
    curriculum_agent.add_failed_task(task)
```

Action Space

Name	Description	Argument
no_op	Do nothing.	\emptyset
use	Use the item held in the main hand.	\emptyset
drop	Drop the item held in the main hand.	\emptyset
attack	Attack with barehand or tool held in the main hand.	\emptyset
craft	Execute a crafting recipe to obtain new items.	Index of recipe
equip	Equip an inventory item.	Slot index of the item
place	Place an inventory item on the ground.	Slot index of the item
destroy	Destroy an inventory item.	Slot index of the item

Observation Space in MineDojo

Modality	Shape	Description
RGB	(3, H, W)	Ego-centric RGB frames.
Equipment	(6,)	Names, quantities, variants, and durabilities of equipped items.
Inventory	(36,)	Names, quantities, variants, and durabilities of inventory items.
Voxel	(3, 3, 3)	Names, variants, and properties of $3 \times 3 \times 3$ surrounding blocks.
Life statistics	(1,)	Agent's health, oxygen, food saturation, etc.
GPS	(3,)	GPS location of the agent.
Compass	(2,)	Yaw and pitch of the agent.
Nearby tools	(2,)	Indicate if crafting table and furnace are nearby the agent.
Damage source	(1,)	Information about the damage on the agent.
Lidar	(Num rays,)	Ground-truth lidar observation.