# Supplementary Materials

Zhonghan Zhao[1,♠], Wenhao Chai[2,♠], Xuan Wang[1,♠], Boyi Li[1], Shengyu Hao[1], Shidong Cao[1], Tian Ye[3], and Gaoang Wang[1,†]

[1] Zhejiang University
[2] University of Washington
[3] Hong Kong University of Science and Technology (GZ)

*The supplementary material is structured as follows:*

- We begin with the "Contribution and Idea" of the STEVE to explain the reason of data collection and fine-tuning in Appendix A.
- Then, we introduce the "STEVE Pseudo Code" section, detailing the STEVE method, encompassing input requirements, variables, functions, and the procedure of the STEVE Method algorithm in Appendix B
- Next, the "Component Comparison" section offers a comparison of the STEVE system with representative and concurrent works, focusing on various system characteristics such as availability of demos, nature of rewards, and types of observations, and even the specific core method differences and whether there is control capability, among others, in Appendix C
- The "Implementation Detail" section outlines the steps involved in implementing STEVE, including the warm-up with the Question-Answering pair, collection of Vision-Environment pair, and training and fine-tuning processes in Appendix D.
- In the "STEVE-7B/13B" section, we introduce the features and capabilities of the STEVE-7B/13B model, along with the Pearson correlation coefficient analysis for various score methods and showcases the efficiency of language models in the Minecraft knowledge QA domain, as shown in Appendix E.
- The "Prompt" section provides insights into the LLM Decomposer, the process for generating Question-Answering pairs, and the evaluation methodology in Appendix F.
- The "Demo Image and Video" section presents practical demonstrations of STEVE in various tasks, supported by figures and video content, and examples of skills and code in Appendix G.
- Lastly, the "Skill-Code Example" section contains examples from the STEVE-21K skill-code part, illustrating specific skill codes used in the STEVE system, as showvn in Appendix H

## A Contribution and Motivation

STEVE is the first **pure LLM-based** embodied agent with **vision perception** and **comprehensive control capabilities**, as shown in Table C2. We get **SOTA** performance as an LLM-based model, providing excellent performance

in visual perception and **interpretable planning** (Section 5.3) with **flexibility** on tasks. We explain why we collect data and fine-tune a smaller LLM instead of relying solely on GPT-4 as follows.

**(1) Innovative embodied agent tasks with a smaller LLM:** We showed that complex tasks can be done with few LLM parameters while ensuring interpretability. This is necessary for cost-effective solutions. STEVE-7B/13B has better efficiency than GPT-4, achieving SOTA performance by adding visual perception. Our work verifies that models with small parameter quantities can perform well beyond models with extremely large parameter quantities through simple finetuning on a specific domain. Although it may sacrifice some performance in unseen scenarios, it is common for most non-gradient-free and finetuned LLM methods, as shown in Table C2.

**(2) Superior performance and vision integration:** Our work enhances the SOTA LLM-based model Voyager with visual capabilities. STEVE has shown a significant performance boost in tasks like tech tree mastery and block searching, suggesting potential improvements in navigation tasks. This approach bridges the gap between virtual environments and the real world for embodied agents. We train the vision encoder with object-level labels, providing fine-grained recognition.

**(3) Multi-role collaboration in language instruction:** The Language Instruction system, composed of different roles of the STEVE-7B/13B model (Planner, Critic, Curriculum, and Describer) with buffer to manage tasks in Minecraft by formulating long-term plans, refining strategies through feedback, adapting via a curriculum of complex tasks, and translating strategies into executable textual actions.

**(4) Online training via simulation:** We provide online training for visual encoders, multi-role language models, and agent simulation for faster and more realistic training. No need for separate visual and language model training or offline strategies.

**(5) Cost-effectiveness lightweight and sustainable development:** Many methods rely directly on GPT-4, combined with RL model. The fine-tuning process LoRA we employed is cost-effective without any API expense. STEVE is also very lightweight because using only the 7B/13B model without other additional models will not bring additional costs, and it also has control capabilities. STEVE will be open-source, reducing ongoing development expenses. This makes our model more practical for complex tasks such as RLEF (Environment Feedback) and Role Agents.

**(6) Value of the dataset:** We contribute a comprehensive multimodal dataset of **actual simulation** that spans visual perception, knowledge instruction, and action execution. This dataset has 20K Minecraft question-answering text pairs, 600 vision-environment pairs including more than ***36K*** temporal frames with corresponding environment information and LLM context, and 210 skill code snippets with descriptions (GPT generation). We collect a comprehensive dataset including simulation (vision-environment-chatflow), instruction (Knowledge QA), and skill codebase.

## B STEVE Pseudo Code

STEVE takes an image, the agent's current state, and a task as inputs and uses a series of functions like a vision encoder, text tokenizer, and the core STEVE-7B/13B model to process these inputs. The model generates a plan consisting of steps, each of which is decomposed into an executable format, encoded into a query, and then used to retrieve an action code from a skill database. This process ultimately produces a series of action codes that guide the agent in performing the specified task.

**Algorithm 1** STEVE Method $F$

**Input:** Image $\mathcal{I}$, Agent State $\mathcal{S}$, Task $\mathcal{T}$
**Variables:**
  $step_e$: Executable step
  $q$: Action query
  $a$: Action code
**Functions:**
  $V$: Vision encoder
  $T$: Tokenizer for text
  $S$: STEVE-7B/13B
  $E$: Query encoder
  $R$: Retrieval module on Skill Database
**procedure** $F(\mathcal{I}, \mathcal{S}, \mathcal{T})$
    $O_V \leftarrow V(\mathcal{I})$
    $O_S \leftarrow T(\mathcal{S})$
    $O_T \leftarrow T(\mathcal{T})$
    $P_H \leftarrow S(O_V, O_S, O_T)$
    **for** each *step* in $P_H$ **do**
        $step_e \leftarrow Decompose(step)$
        $Query \leftarrow E(step_e)$
        $a \leftarrow R(Query)$
    **end for**
**end procedure**

## C Component Comparison

As shown in Tab. C1, we compare STEVE with several existing works in the field. We consider the following characteristics of each system: availability of demos, nature of rewards, types of observations, action form, use of iterative planning, existence and capacity of a skill database, and whether they are gradient-free. As shown in Table C2, our STEVE uses Ego-view RGB to directly perceive visual information within a 256-block range, enhancing realism and directional sensitivity. It also features **unique multi-role** language models with continuous learning for long-term planning and task adaptation.

| | VPT [1] | DreamerV3 [5] | DECKARD [7] | DEPS [11] | Plan4MC [13] | Voyager [9] | STEVE (ours) |
|---|---|---|---|---|---|---|---|
| Demos | Videos | None | Videos | None | None | None | Videos |
| Rewards | Sparse | Dense | Sparse | None | Dense | None | None |
| Observations | Pixels Only | Pixels & Meta | Pixels & Inventory | Feedback & Inventory | Pixels & Meta | Feedback & Meta & Inventory | Pixels & Feedback & Meta & Inventory |
| Actions | Keyboard & Mouse | Discrete | Keyboard & Mouse | Keyboard & Mouse | Discrete | Code | Code |
| Iterative Planning | | | | ✓ | | ✓ | ✓ |
| Skill Database | | | | | 9 | 172 | 210 |
| Gradient-Free | | | | | | ✓ | ✓ |

**Table C1:** Comparison between **STEVE and representative works.** It is a system-level comparison consisting of LLM-based and RL-based methods.

| Method | LLM. | Perception | Training | Dataset |
|---|---|---|---|---|
| Voyager | GPT-4-0613 | Meta, r = 32 | - | - |
| **STEVE (ours)** | STEVE-13b | Ego RGB, r = 256 | Online | simulation |

**Table C2: Component comparison with Voyager.** Meta is the metadata from the background, Ego RGB is for Ego-view image, and r represents the maximal distance of perceived entities.

We also compare agents using non-pure LLM and RL, excelling in environments with defined objectives but challenging to address open-ended tasks. Our STEVE is designed for open-ended tasks with varying language-based objectives, operating autonomously in the unrestricted Minecraft environment, contrasting with RL agents confined to specific settings like Minedojo. As a loose comparison, it achieves a 19% success rate in the Diamond Tool task within 10 minutes, surpassing RL-based VPT [2] at 12%.

## D Implementation Detail

**(1) RayTracing module:** Raytracing is a perception module we developed. Through this module, we can ensure that all content outside the robot's perspective screen is eliminated and only all visible blocks and entities are saved. Note that it is only used to collect data. This module is removed in STEVE since a vision encoder is used for perception.
**(2) LLM warm-up:** We use LoRA finetuning on the 20K Knowledge QA pairs of STEVE-21K, using LLaMA2-7B/13B, before being deployed in a simulated environment. The warm-up step is a one-time process that allows the model to

absorb extensive knowledge. It is a crucial step in the initial simulation, which enables the collection of the vision-environment pair.

**(3) Second stage of LLM finetuning:** We simulate STEVE-13B in Minecraft, a simple embodied agent that uses the RayTracing module directly. After 5,000 simulations, we collected context information from successful runs, including sequences of QA. We then use these to fine-tune STEVE-13B via the LoRA process. Note that the process is to adjust STEVE-13B to work on **correct simulation knowledge** while remaining **adapted to visual perception**.

**(4) Vision-environment pair collection:** We run STEVE-13B with a second-round Minecraft simulation. We recorded 600 **videos**, collected environmental data with RayTracing, saved state information, and chatflow. We collect the dataset in six different terrains: forest, desert, coastal, *etc*. We use the STEVE-7B/13B model to enable robots to autonomously plan and execute actions based on tasks defined by human supervisors. We record the video of the robot operation, environment information, and all the corresponding chat flow. Note that we use ray tracing [12] to ensure the environmental information obtained is the blocks and entities seen in the field of vision.

**(5) Multi-role instruction:** The Planner uses visual information for task decomposition and sequencing, while the Critic, inspired by Voyager, provides essential error correction. The Curriculum stores extensive semantic information to support continuous learning and preserve effective task sequences. Lastly, the Describer, influenced by the chain of summarization [6], condenses historical data to improve long-term memory.

**(6) STEVE-21K dataset:** It is not only specifically tailored for training our STEVE system but is used as the foundation for developing STEVE from scratch, utilizing the open-source LLaMA as the base model. The dataset contains visual environment data and contextual LLM QA pairs, with 50% used for real-time learning and the rest for diversity. STEVE-21K effectively trains STEVE, showing its potential for wider language model training.

**(7) Experiment and evaluation:** We propose Continuous Block Search, Knowledge QA, and Tech Tree Mastery (Section 5.3). Our model has achieved SOTA results in each domain. **Continuous Block Search** enhances exploration abilities by evaluating the visual perception capabilities of the model. It is inspired by Voyager's map exploration tasks, emphasizing purpose-driven searches influenced by multimodal navigation tasks [10]. **Knowledge QA** explicitly demonstrates the language model's performance (more Language model efficiency details in Supplementary Material D.). LLM-based methods can reflect simulation performance to an extent. High QA performance can indicate high simulation performance and vice versa. **Tech Tree Mastery** demonstrates the control capabilities. We use the same settings as the comparison method, including the code database.

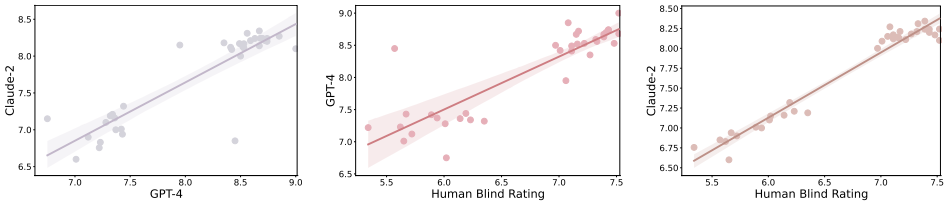
# E STEVE-7B/13B

We propose STEVE-7B/13B, a powerful language model series derived from LLaMA2 [4], fine-tuned specifically on Minecraft-related content from the Minecraft-wiki and Reddit corpus. This model's expertise covers a broad spectrum of game-specific knowledge areas including:

- **World Understanding:** Geography, biomes, and entity interactions.
- **Player Mechanics:** Health management, combat, and mobility.
- **Survival Strategies:** Food sourcing, shelter building, and enemy evasion.
- **Resource Management:** Item gathering, mining techniques, and inventory optimization.
- **Crafting and Construction:** Recipe knowledge, item crafting, and structure creation.
- **Utility and Tool Usage:** Tool selection, item upgrades, and special abilities.

We test the performance of our STEVE-7B/13B on the above-mentioned functional indicators. The entire knowledge question and answering section of our paper used 1000 pairs that were divided from our STEVE-21K. Among these 1000 pairs, the categories World & Entities, Player Mechanics & Survival, Knowledge & Discovery, Resources & Crafting, Tools & Utilities, and Miscellaneous are respectively 332, 152, 108, 219, 169, 20.

## E.1 Detailed evaluation




**(a) GPT-4 and Claude-2: 0.976** **(b) GPT-4 and Human Blind Rating: 0.965** **(c) Claude-2 and Human Blind Rating: 0.998**

**Fig. E1:** Pearson Correlation Analysis, including scatters and trend. Note that all scatters come from Tab. E3 E4 E5

*Pearson correlation coefficient on score methods.* The formula represents the Pearson correlation coefficient:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where:

| Method | World & Entities | Player Mechanics & Survival | Knowledge & Discovery | Resources & Crafting | Tools & Utilities | Miscellaneous | Overall |
|---|---|---|---|---|---|---|---|
| GPT-4 | 8.53 | 8.56 | **8.85** | 8.49 | 8.63 | **9.00** | 8.59 |
| Llama2-7B | 7.22 | 7.43 | 7.12 | 7.01 | 7.28 | 8.45 | 7.23 |
| Llama2-13B | 7.34 | 7.32 | 7.42 | 7.37 | 7.44 | 6.75 | 7.36 |
| **STEVE-7B** | 8.67 | 8.41 | 8.42 | 8.53 | 8.35 | 8.50 | 8.52 |
| **STEVE-13B** | **8.74** | **8.67** | 8.72 | **8.68** | **8.74** | 7.95 | **8.70** |

**Table E3: Quantitive comparison on knowledge question and answering task by GPT-4 [8].** It is rated on a scale of 0 to 10; Higher scores indicate greater alignment of the generated answers with the ground truth.

| Method | World & Entities | Player Mechanics & Survival | Knowledge & Discovery | Resources & Crafting | Tools & Utilities | Miscellaneous | Overall |
|---|---|---|---|---|---|---|---|
| GPT-4 | 8.17 | 8.31 | **8.27** | 8.17 | **8.24** | 8.10 | 8.21 |
| Llama2-7B | 6.76 | 6.94 | 6.90 | 6.60 | 7.10 | 6.85 | 6.83 |
| Llama2-13B | 7.21 | 7.19 | 7.01 | 7.00 | 7.32 | 7.15 | 7.16 |
| **STEVE-7B** | 8.14 | 8.12 | 8.09 | 8.11 | 8.18 | 8.00 | 8.13 |
| **STEVE-13B** | **8.24** | **8.34** | 8.21 | **8.24** | 8.20 | **8.15** | **8.24** |

**Table E4: Quantitive comparison on knowledge question and answering task by Claude-2 [3].** It is rated on a scale of 0 to 10; Higher scores indicate greater alignment of the generated answers with the ground truth.

| Method | World & Entities | Player Mechanics & Survival | Knowledge & Discovery | Resources & Crafting | Tools & Utilities | Miscellaneous | Overall |
|---|---|---|---|---|---|---|---|
| GPT-4 | **7.48** | 7.33 | 7.08 | 7.11 | 7.39 | **7.52** | 7.32 |
| Llama2-7B | 5.34 | 5.67 | 5.72 | 5.65 | 6.01 | 5.57 | 5.62 |
| Llama2-13B | 6.23 | 6.35 | 5.89 | 5.94 | 6.19 | 6.02 | 6.14 |
| **STEVE-7B** | 7.15 | 7.11 | 7.01 | 7.22 | 7.27 | 6.97 | 7.16 |
| **STEVE-13B** | 7.43 | **7.39** | **7.17** | **7.52** | **7.43** | 7.06 | **7.41** |

**Table E5: Quantitive comparison on knowledge question and answering task by human blind rating.** It is rated on a scale of 0 to 10; Higher scores indicate greater alignment of the generated answers with the ground truth.

- $r_{xy}$ is the Pearson correlation coefficient between two variables $x$ and $y$.
- $x_i$ and $y_i$ are the individual sample points for variables $x$ and $y$, respectively.
- $\bar{x}$ and $\bar{y}$ are the means (averages) of the $x$ and $y$ samples, respectively.
- $n$ is the number of sample points.

This formula essentially measures the degree of linear relationship between two variables. It compares the product of their deviations from their respective means. The numerator captures the covariance between the two variables, and the denominator normalizes this value, ensuring that the coefficient remains between -1 and +1. The Pearson correlation coefficient measures how much two variables change together compared to how much they vary individually.

As shown in Tab. E6, the Pearson correlation coefficients are calculated between these methods: GPT-4 vs. Claude-2 (0.976), GPT-4 vs. Human Blind Rating (0.965), and Claude-2 vs. Human Blind Rating (0.998). It suggests a high level of agreement among these evaluation approaches. The alignment of scores from different evaluation methods reinforces the reliability of our assessment. Our STEVE models outperform other language models in knowledge question and answering tasks, including the benchmark GPT-4. This superior performance of STEVE, particularly the 13B version, is evident across a broad spec-

| Method | Pearson Correlation Coefficients |
|---|---|
| GPT-4 VS. Claude-2 | 0.976 |
| GPT-4 VS. Human Blind Rating | 0.965 |
| Claude-2 VS. Human Blind Rating | 0.998 |

**Table E6:** Comparison on **Pearson correlation coefficients between score methods**. It calculates the mean score across each score dimensions for each language methods and then computes the Pearson correlation between these means for each pair of score methods (GPT-4, Claude-2 and Human blind rating). The Pearson correlation coefficient ranges from -1 to +1. Higher Pearson correlation coefficient (closer to +1) means that there is a stronger positive linear relationship between the two sets of data.

| Method | time(s) (↓) |
|---|---|
| Llama2-7B [4] | 18.34 |
| Llama2-13B [4] | 21.17 |
| GPT-4 [8] | 22.86 |
| **STEVE-7B** | **7.19** |

**Table E7:** Comparison on **language model efficiency**. It shows the average time for each question-answering.

trum of categories. This suggests that our model has a deeper understanding of Minecraft-related content and exhibits a more accurate and consistent ability to generate relevant responses.

### E.2 Language model efficiency

We primarily test the reasoning efficiency of large language models on Minecraft knowledge QA and showcased case studies. We conduct this experiment in the Question-Answering part of our STEVE-21K dataset. As shown in Tab. E7, our model STEVE-7B/13B achieves leading performance in terms of time cost. This is due to its smaller number of parameters compared to GPT-4, and its shorter yet more accurate responses compared to the Llama series.

## F Prompt

### F.1 LLM Decomposer

We use STEVE-7B/13B for goal decomposition. The format of the prompt is presented thus: directions under the "SYSTEM" role and questions under the

"USER" role. The `{target amount}`, `{target name}` and `{extensive knowledge}` are designed to be filled with specific content, which will then be inputted into the language model after substitution.

### F.2 Generate Question-Answering pairs

We employ the following prompt during the collection of Question-Answer pairs. In this context, the placeholder `{text}` is intended to be filled with cleaned Minecraft-Wiki text.

### F.3 Evaluation

We utilize GPT-4 and Claude-2 to evaluate various models, employing the following prompt structure. Within this structure, the placeholders `{question}`, `{ground truth}`, and `{answer}` are designated to be filled with questions, standard answers, and model-generated answers, respectively.

## G Demo Image and Video

As shown in Fig. H2, we experiment with our STEVE on various practical tasks, covering collecting, combating, mining *etc*.

## H Examples

The section contains examples from the STEVE-21K skill-code part, illustrating specific skill codes used in the STEVE system.

**SYSTEM:**

You are designated as an assistant for the game Minecraft. Your task involves formulating goals related to obtaining specific objects within the game. I'll provide you with a specific target and relevant extensive information. Please provide a standardized format for acquiring this target as a goal.

Goal Structure:
{
"object": "[Name of Target Object]",
"amount": [Target Amount],
"material": {
[Material Name]: [Material Quantity],
...
},
"tool": "[Tool Required]",
"info": "[Concise Information Related to the Goal]"

}
- Target: Enter the name of the target object you wish to obtain or craft.,
- Amount: Specify the amount of the target object needed.,
- Material: List the materials required to achieve this goal. Format each entry as "material name": quantity. If no material is required, set this field to None,
- Tool: Indicate the most basic tool necessary for this goal. If multiple tools can be used, choose the simplest one. If no tool is required, set this to None,
- Info: Provide essential information related to the goal. Summarize the knowledge in up to three sentences, focusing on key details about obtaining or crafting the target object.

Requirements:
1. Goals must be constructed based on the provided knowledge, rather than relying solely on pre-existing knowledge.
2. The "info" section should be concise and informative, limited to a maximum of three sentences. Extracting and summarizing the key information from the provided knowledge is essential, rather than replicating the entire text.

Example Goal 1:
{
"target": "bed",
"amount": 1,
"material": {"wool": 3, "planks": 3},
"tool": "crafting table",
"info": "A bed is crafted using 3 wool and 3 wooden planks on a crafting table. Beds allow players to skip the night and reset their spawn point."
}
Example Goal 2:
{
"target": "paper",
"amount": 3,
"material": {"sugar cane": 3},
"tool": "None",
"info": "Paper is crafted from 3 sugar cane, arranged in a row. It's used for creating maps and books."
}

**USER:**

Target info: `{target amount} {target name}`
Knowledge info: `{extensive knowledge}`

**SYSTEM:**

You are a helpful assistant engaging in a conversation with an in-game Minecraft agent. This agent seeks to enhance both its survival techniques and interaction tactics within the Minecraft universe.

Your task is to extract crucial questions and their corresponding answers from the provided Minecraft information. These may pertain to survival challenges, crafting recipes, interactions with in-game entities, and other relevant aspects.

##INSTRUCTIONS:

- Craft questions to prompt the in-game Minecraft agent to contemplate various aspects of gameplay, including survival challenges, crafting recipes, managing resources, navigating terrain, interactions with in-game entities, and more.
- For each question, extract a suitable answer. Each answer should elucidate the agent's perspective, emphasizing effective strategies, item applications, or interaction approaches.

**USER:**

The information about Minecraft is: `{text}`.

Present your output in the mold of a Python dictionary string, utilizing 'Q' for questions and 'A' for the agent's anticipated responses. For multiple questions derived, maintain the 'Q' and 'A' structure.

An example could be: 'Q': 'How to craft a wooden pickaxe in Minecraft?', 'A': 'To craft a wooden pickaxe, the approach involves...', 'Q': '...', 'A':'...', ...

**SYSTEM:**

You are an intelligent chatbot designed for evaluating the correctness of generative outputs for question-answer pairs.

Your task is to compare the predicted answer with the correct answer and determine if they match meaningfully. Here's how you can accomplish the task:

##INSTRUCTIONS:

- Focus on the meaningful match between the predicted answer and the correct answer.
- Consider synonyms or paraphrases as valid matches.
- Evaluate the correctness of the prediction compared to the answer.

**USER:**

Please evaluate the following question-answer pair:
Question: `{question}`
Correct Answer: `{ground truth}`
Predicted Answer: `{answer}`

Provide your evaluation only in the form of a score, where the score is an integer value between 0 and 10, with 10 indicating the highest meaningful match.

Do not provide any other output text or explanations. Only provide output score. For example, your response should look like this: '9'.

**Skill-Code example 1.1:** craftShieldImproved.

```
async function craftShieldImproved(bot) {
  // Check if there are enough oak planks in the inventory
  let oakPlanksCount = bot.inventory.count(mcData.itemsByName
    .oak_planks.id);

  // If not, check if there are enough oak logs in the
    inventory
  if (oakPlanksCount < 6) {
    let oakLogsCount = bot.inventory.count(mcData.itemsByName
    .oak_log.id);
    const planksToCraft = Math.ceil((6 - oakPlanksCount) / 4)
    ;

    // If not, explore to find and mine oak logs
    if (oakLogsCount < planksToCraft) {
      await exploreUntil(bot, new Vec3(1, 0, 1), 60, () => {
        const oak_log = bot.findBlock({
          matching: mcData.blocksByName["oak_log"].id,
          maxDistance: 32
        });
```

```
        return oak_log;
      });
      await mineBlock(bot, "oak_log", planksToCraft -
    oakLogsCount);
      bot.chat("Collected oak logs.");
    }

    // Craft oak planks from oak logs
    await craftItem(bot, "oak_planks", planksToCraft);
    bot.chat("Crafted oak planks.");
    oakPlanksCount = bot.inventory.count(mcData.itemsByName.
    oak_planks.id);
  }

  // Check if there are enough iron ingots in the inventory
  let ironIngotsCount = bot.inventory.count(mcData.
    itemsByName.iron_ingot.id);

  // If not, explore to find and mine iron ores
  if (ironIngotsCount < 1) {
    await exploreUntil(bot, new Vec3(0, -1, 0), 60, () => {
      const iron_ore = bot.findBlock({
        matching: mcData.blocksByName["iron_ore"].id,
        maxDistance: 32
      });
      return iron_ore;
    });
    await mineBlock(bot, "iron_ore", 1);
    bot.chat("Collected iron ores.");

    // Smelt iron ores into iron ingots
    await smeltItem(bot, "iron_ore", "coal", 1);
    bot.chat("Smelted iron ores into iron ingots.");
  }

  // Place the crafting table near the bot
  const craftingTablePosition = bot.entity.position.offset(1,
     0, 0);
  await placeItem(bot, "crafting_table",
    craftingTablePosition);

  // Craft a shield using the crafting table
  await craftItem(bot, "shield", 1);
  bot.chat("Crafted a shield.");
}
```

**Skill-Code example 1.2:** craftBed.

```
async function craftBed(bot) {
  // Check if there is a crafting table in the inventory or
    nearby
  let craftingTable = bot.inventory.findInventoryItem(mcData.
    itemsByName["crafting_table"].id);
  if (!craftingTable) {
    craftingTable = bot.findBlock({
      matching: mcData.blocksByName["crafting_table"].id,
      maxDistance: 32
    });
    if (!craftingTable) {
      // Mine oak logs and craft oak planks
      await mineBlock(bot, "oak_log", 1);
      await craftItem(bot, "oak_planks", 2);
      // Craft a crafting table using oak planks
      await craftItem(bot, "crafting_table", 1);
    }
  }

  // Find a suitable position to place the crafting table if
    it's not already placed
  const tablePosition = bot.entity.position.offset(1, 0, 0);
  const tableBlock = bot.blockAt(tablePosition);
  if (!tableBlock || tableBlock.name !== "crafting_table") {
    if (tableBlock && tableBlock.name === "lily_of_the_valley
    ") {
      // Find a different position to place the crafting
    table
      const newPosition = bot.entity.position.offset(-1, 0,
    0);
      await placeItem(bot, "crafting_table", newPosition);
    } else {
      await placeItem(bot, "crafting_table", tablePosition);
    }
  }

  // Check if there are enough wool blocks in the inventory
  let wool = bot.inventory.findInventoryItem(mcData.
    itemsByName["white_wool"].id);
  if (!wool || wool.count < 3) {
    // Find a sheep and use shears to collect wool
    bot.chat("Finding a sheep to collect wool...");
    const sheep = await exploreUntil(bot, new Vec3(1, 0, 1),
    60, () => {
      const sheep = bot.nearestEntity(entity => {
        return entity.name === "sheep" && entity.position.
    distanceTo(bot.entity.position) < 32;
      });
```

```
      return sheep;
    });
    await bot.pathfinder.goto(new GoalGetToBlock(sheep.
    position.x, sheep.position.y, sheep.position.z));
    await bot.equip(mcData.itemsByName["shears"].id, "hand");
    await bot.useOn(sheep);
    wool = bot.inventory.findInventoryItem(mcData.itemsByName
    ["white_wool"].id);
  }

  // Check if there are enough wooden planks in the inventory
  let oakPlanks = bot.inventory.findInventoryItem(mcData.
    itemsByName["oak_planks"].id);
  if (!oakPlanks || oakPlanks.count < 3) {
    // Mine oak logs and craft oak planks
    bot.chat("Mining oak logs to craft oak planks...");
    await mineBlock(bot, "oak_log", 1);
    bot.chat("Crafting oak planks...");
    await craftItem(bot, "oak_planks", 1);
  }

  // Craft a bed using the wool blocks and wooden planks
  bot.chat("Crafting a bed...");
  await craftItem(bot, "white_bed", 1);
  bot.chat("Bed crafted.");
}
```

**Skill-Code example 1.3:** craftPaper.

```
async function findSuitablePosition(bot) {
  const offsets = [new Vec3(1, 0, 0), new Vec3(-1, 0, 0), new
     Vec3(0, 0, 1), new Vec3(0, 0, -1)];
  for (const offset of offsets) {
    const position = bot.entity.position.offset(offset.x,
    offset.y, offset.z);
    const block = bot.blockAt(position);
    if (block.name === "air" || block.name === "water") {
      const belowBlock = bot.blockAt(position.offset(0, -1,
    0));
      if (belowBlock.name !== "air" && belowBlock.name !== "
    water") {
        return position;
      }
    }
  }
  return null;
}

async function craftPaper(bot) {
  // Check if there are enough sugar canes in the inventory
  const sugarCanes = bot.inventory.findInventoryItem(mcData.
    itemsByName.sugar_cane.id);
  if (!sugarCanes || sugarCanes.count < 3) {
    // Explore the area to find more sugar canes
    await exploreUntil(bot, new Vec3(1, 0, 1), 60, () => {
      const sugarCane = bot.findBlock({
        matching: mcData.blocksByName.sugar_cane.id,
        maxDistance: 32
      });
      return sugarCane;
    });

    // Mine the sugar canes
    await mineBlock(bot, "sugar_cane", 3 - (sugarCanes ?
    sugarCanes.count : 0));
  }

  // Check if there is a crafting table nearby or in the
    inventory
  const craftingTable = bot.findBlock({
    matching: mcData.blocksByName.crafting_table.id,
    maxDistance: 32
  }) || bot.inventory.findInventoryItem(mcData.itemsByName.
    crafting_table.id);

  // If not, craft a crafting table using the oak_log in the
    inventory
```

```
  if (!craftingTable) {
    await craftItem(bot, "crafting_table", 1);
  }

  // Find a suitable position to place the crafting table if
    it's not already placed
  const craftingTableBlock = bot.findBlock({
    matching: mcData.blocksByName.crafting_table.id,
    maxDistance: 32
  });
  let craftingTablePosition;
  if (!craftingTableBlock) {
    craftingTablePosition = await findSuitablePosition(bot);
    if (!craftingTablePosition) {
      bot.chat("No suitable position found to place the
    crafting table.");
      return;
    }
  } else {
    craftingTablePosition = craftingTableBlock.position;
  }

  // Place the crafting table at the suitable position
  await placeItem(bot, "crafting_table",
    craftingTablePosition);

  // Craft 3 paper using the sugar canes
  await craftItem(bot, "paper", 1);
  bot.chat("3 paper crafted.");
}
```

**Skill-Code example 1.4:** craftShieldWithIron.

```
async function craftShieldWithIron(bot) {
  // Check if we have enough spruce planks
  const sprucePlankCount = bot.inventory.count(mcData.
    itemsByName.spruce_planks.id);
  if (sprucePlankCount < 6) {
    // Mine 3 more spruce planks
    await mineBlock(bot, "spruce_log", 3);
    // Craft spruce planks from spruce logs
    await craftItem(bot, "spruce_planks", 3);
    bot.chat("Spruce planks crafted.");
  }

  // Check if we have enough iron ingots
  const ironIngotCount = bot.inventory.count(mcData.
    itemsByName.iron_ingot.id);
  if (ironIngotCount < 1) {
    // Mine iron ore
```

```
    await mineBlock(bot, "iron_ore", 1);
    bot.chat("Iron ore mined.");

    // Place a furnace if it's not already placed
    const furnacePosition = bot.entity.position.offset(1, -1,
     0);
    const furnaceBlock = bot.blockAt(furnacePosition);
    if (!furnaceBlock || furnaceBlock.name !== "furnace") {
      await placeItem(bot, "furnace", furnacePosition);
      bot.chat("Furnace placed.");
    }

    // Smelt iron ore into iron ingot using a furnace
    await smeltItem(bot, "raw_iron", "coal", 1);
    bot.chat("Iron ingot smelted.");
  }

  // Place the crafting table if it's not already placed
  const craftingTablePosition = bot.entity.position.offset(1,
     -1, 0);
  const craftingTableBlock = bot.blockAt(
    craftingTablePosition);
  if (!craftingTableBlock || craftingTableBlock.name !== "
    crafting_table") {
    await placeItem(bot, "crafting_table",
    craftingTablePosition);
    bot.chat("Crafting table placed.");
  }

  // Craft the shield using the 6 spruce planks and 1 iron
    ingot
  await craftItem(bot, "shield", 1);
  bot.chat("Shield crafted.");
}
```

**Skill-Code example 1.5:** mineFiveLapisOres.

```
async function mineFiveLapisOres(bot) {
  // Check if the bot has a stone pickaxe, iron pickaxe, or
    diamond pickaxe in its inventory
  const stonePickaxe = bot.inventory.findInventoryItem(mcData
    .itemsByName["stone_pickaxe"].id);
  const ironPickaxe = bot.inventory.findInventoryItem(mcData.
    itemsByName["iron_pickaxe"].id);
  const diamondPickaxe = bot.inventory.findInventoryItem(
    mcData.itemsByName["diamond_pickaxe"].id);

  // Equip the best pickaxe available
  if (diamondPickaxe) {
    await bot.equip(diamondPickaxe, "hand");
```

```
  } else if (ironPickaxe) {
    await bot.equip(ironPickaxe, "hand");
  } else if (stonePickaxe) {
    await bot.equip(stonePickaxe, "hand");
  } else {
    bot.chat("I don't have a stone, iron, or diamond pickaxe
    to mine lapis ores.");
    return;
  }

  // Explore the underground area to find lapis ores
  const lapisOres = await exploreUntil(bot, new Vec3(0, -1,
    0), 60, () => {
    const lapis_ores = bot.findBlocks({
      matching: mcData.blocksByName["lapis_ore"].id,
      maxDistance: 32,
      count: 5
    });
    return lapis_ores.length >= 5 ? lapis_ores : null;
  });

  // Mine 5 lapis ores using the equipped pickaxe
  if (lapisOres) {
    await mineBlock(bot, "lapis_ore", 5);
    bot.chat("5 lapis ores mined.");
  } else {
    bot.chat("Could not find 5 lapis ores.");
  }
}
```

**Skill-Code example 1.6:** smeltTwelveRawCopper.

```
async function smeltTwelveRawCopper(bot) {
  // Check if the bot has a furnace in its inventory
  const furnaceItem = bot.inventory.findInventoryItem(mcData.
    itemsByName.furnace.id);
  if (!furnaceItem) {
    // Craft a furnace if not found in the inventory
    await craftItem(bot, "furnace", 1);
    bot.chat("Furnace crafted.");
  }

  // Find a suitable position to place the furnace
  const furnacePosition = bot.entity.position.offset(1, -1,
    1);

  // Place the furnace at the found position
  await placeItem(bot, "furnace", furnacePosition);
  bot.chat("Furnace placed.");
```

```
  // Check if the bot has enough coal in its inventory to
    smelt 12 raw copper
  const coalCount = bot.inventory.count(mcData.itemsByName.
    coal.id);
  if (coalCount < 12) {
    // Mine enough coal to smelt 12 raw copper
    await mineBlock(bot, "coal_ore", 12 - coalCount);
    bot.chat("Coal mined.");
  }

  // Smelt the 12 raw copper using the furnace and the
    available coal as fuel
  await smeltItem(bot, "raw_copper", "coal", 12);
  bot.chat("12 raw copper smelted.");
}
```

**Skill-Code example 1.7:** obtainTenWoodLogs.

```
async function obtainTenWoodLogs(bot) {
  const logNames = ["oak_log", "birch_log", "spruce_log", "
    jungle_log", "acacia_log", "dark_oak_log", "mangrove_log"
    ];
  let totalLogs = 0;

  // Check if there are already 10 wood logs in the inventory
  for (const logName of logNames) {
    const logItem = bot.inventory.findInventoryItem(mcData.
    itemsByName[logName].id);
    if (logItem) {
      totalLogs += logItem.count;
    }
  }
  if (totalLogs >= 10) {
    bot.chat("Already have 10 wood logs in inventory.");
    return;
  }
  const logsNeeded = 10 - totalLogs;
  bot.chat(`Mining ${logsNeeded} wood logs...`);

  // Equip the wooden axe if available in the inventory
  const woodenAxe = bot.inventory.findInventoryItem(mcData.
    itemsByName["wooden_axe"].id);
  if (woodenAxe) {
    await bot.equip(woodenAxe, "hand");
  }
  for (let i = 0; i < logsNeeded; i++) {
    // Use exploreUntil to find a wood log if not found
    nearby
    const logBlock = bot.findBlock({
      matching: block => logNames.includes(block.name),
```

```
      maxDistance: 32
    });
    if (!logBlock) {
      await exploreUntil(bot, new Vec3(1, 0, 1), 60, () => {
        const foundLog = bot.findBlock({
          matching: block => logNames.includes(block.name),
          maxDistance: 32
        });
        return foundLog;
      });
    }

    // Mine the required number of wood logs using mineBlock
    await mineBlock(bot, logBlock.name, 1);
  }
  bot.chat("Obtained 10 wood logs.");
}
```

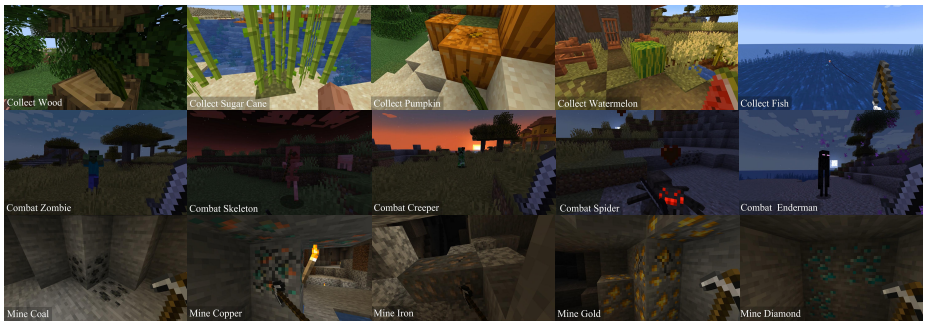



**Fig. H2: Demo of basic survival skills.**

## References

1. Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., Clune, J.: Video pretraining (vpt): Learning to act by watching unlabeled online videos. arXiv preprint arXiv: Arxiv-2206.11795 (2022)
2. Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., Clune, J.: Video pretraining (vpt): Learning to act by watching unlabeled online videos. Advances in Neural Information Processing Systems **35**, 24639–24654 (2022)
3. Talk to claude (2023), `https://claude.ai`
4. Gao, P., Han, J., Zhang, R., Lin, Z., Geng, S., Zhou, A., Zhang, W., Lu, P., He, C., Yue, X., et al.: Llama-adapter v2: Parameter-efficient visual instruction model. arXiv preprint arXiv:2304.15010 (2023)

5. Hafner, D., Pasukonis, J., Ba, J., Lillicrap, T.: Mastering diverse domains through world models. arXiv preprint arXiv: Arxiv-2301.04104 (2023)
6. Ma, W., Mi, Q., Yan, X., Wu, Y., Lin, R., Zhang, H., Wang, J.: Large language models play starcraft ii: Benchmarks and a chain of summarization approach. arXiv preprint arXiv:2312.11865 (2023)
7. Nottingham, K., Ammanabrolu, P., Suhr, A., Choi, Y., Hajishirzi, H., Singh, S., Fox, R.: Do embodied agents dream of pixelated sheep?: Embodied decision making using language guided world modelling. ARXIV.ORG (2023). https://doi.org/10.48550/arXiv.2301.12050
8. OpenAI: Gpt-4 technical report. arXiv preprint arXiv: Arxiv-2303.08774 (2023)
9. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291 (2023)
10. Wang, H., Liang, W., Van Gool, L., Wang, W.: Towards versatile embodied navigation. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
11. Wang, Z., Cai, S., Liu, A., Ma, X., Liang, Y.: Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. arXiv preprint arXiv:2302.01560 (2023)
12. Whitted, T.: An improved illumination model for shaded display. In: ACM Siggraph 2005 Courses, pp. 4–es (2005)
13. Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., Lu, Z.: Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. arXiv preprint arXiv:2303.16563 (2023)