A Comparative Analysis of Embodied AI Agents in Minecraft

Anonymous submission

Abstract

Minecraft has emerged as a compelling testbed for embodied AI research, offering open-ended gameplay that demands long-term planning, adaptability, and multimodal interaction. This paper presents a comparative study of four AI agents: STEVE-1, Voyager, r1, and r2. Each model was evaluated in terms of architecture, capabilities, and limitations. In addition, a human-subject evaluation was conducted on PLAICraft, a custom multiplayer Minecraft environment, to assess in-game interactions across factors such as helpfulness, human believability, and communication quality. The results underscore the importance of real-time perception, persistent memory, and adaptive behavior in creating more effective agents. Future directions include integrating state of the art multimodal agent frameworks such as JARVIS-VLA and Optimus-3.

1 Introduction

Minecraft is a sandbox game that poses unique challenges for AI. Unlike traditional, rule-bound games with clearly defined objectives, Minecraft provides an open-ended environment where creativity, adaptability, and long-term planning are essential. Agents operating in Minecraft should be able to manage a broad range of tasks—such as gathering resources, constructing elaborate structures, and navigating complex terrain—while continuously adapting to new situations. As such, Minecraft serves not only as a benchmark for in-game performance but also as a testbed for developing more general-purpose decision-making systems.

Minecraft AI agents can broadly be categorized into two types by their control mechanisms. Models such as STEVE-1(Lifshitz et al. 2023) operate at a low level, using keyboard and mouse inputs to directly interact with the game environment. These agents often learn from human video demonstrations or reinforcement learning, allowing them to reproduce fine-grained behaviors. While this can lead to more embodied, humanlike interactions, such agents often struggle with long-term planning and complex goal-setting. In contrast, models like Voyager(Wang et al. 2023) function at a higher level of abstraction, issuing strategic action plans through API calls. This enables more scalable, goal-directed behavior—such as multi-step tool crafting or parallel task coordination—but comes at the cost of reduced reactivity and embodied realism. These systems often rely on handcrafted APIs and predefined success metrics, which limits their flexibility and creative autonomy.

This paper compares agents across this spectrum—from low-level control to high-level planning—within the Minecraft environment.[1] Our goal is to explore how design choices in AI agents influence human perceptions of their behavior in a real-time, open-ended setting.

2 Related Works

Most prior research on Minecraft agents has focused on improving training methods, with less attention paid to how these agents are perceived by human users.

Recent work has begun to explore broader forms of evaluation. For instance, MineAnyBuild (Wei et al. 2025), is a benchmark that tests spatial planning abilities in Minecraft through multimodal prompts and creative construction tasks. Their framework evaluates agents along dimensions such as spatial understanding, reasoning, and commonsense. While the emphasis is on task competence rather than subjective experience, the work demonstrates how Minecraft can support richer, more open-ended evaluations beyond goal completion.

The TeamCraft benchmark (Long et al. 2024) further extends this direction by evaluating multimodal, multi-agent collaboration in Minecraft across 55,000 procedurally generated tasks. Although their focus is on agent–agent teamwork and generalization, rather than human perception, their results highlight the challenges of building agents that behave robustly in complex, dynamic environments—an issue our study also touches on from a user-facing perspective.

Minecraft Universe (MCU) (Zheng et al. 2025) proposes a large-scale task benchmark to evaluate agent performance and generalization across a large suite of atomic tasks. Their evaluation framework is designed to align with human judgments of task success, but does not capture subjective qualities of agent behavior.

Our work is similar in spirit to these directions, but

[1] Project code and additional materials available at https://github.com/g30133/r2.

instead of proposing a new benchmark or training method, we focus on collecting human responses to existing agents. Specifically, participants engaged directly with the agents in a shared, real-time Minecraft multiplayer environment. After each interaction, they were asked to reflect on their experience, providing impressions of the agent's behavior. Our goal was to examine how differences in control architecture—ranging from low-level embodied policies to high-level planning—shape people's perceptions of qualities like realism, responsiveness, and creativity.

3 Model Architectures and Design

In this section, we provide a detailed overview of the 4 AI agents under consideration: STEVE-1, Voyager, r1 and r2. Each model has a distinct approach to representation, policy generation, and learning, reflecting different design philosophies for tackling the open-ended and complex challenges in Minecraft.

3.1 STEVE-1

STEVE-1 first maps text instructions into a visual embedding (MineCLIP) and then uses a policy network finetuned from VPT(Video Pretraining)(Baker et al. 2022) to generate actions that achieve the desired visual goal in Minecraft.

MineCLIP Embeddings. The MineCLIP model consists of two encoders: a video encoder $\mathrm{Enc}_{\mathrm{vid}}$ that maps short (16-frame) Minecraft clips to a visual embedding, and a text encoder $\mathrm{Enc}_{\mathrm{text}}$ that maps text prompts into a text embedding. Both embeddings live in the same vector space, enabling alignment between short clips of gameplay and natural language instructions. A 16-frame snippet of gameplay is encoded by $\mathrm{Enc}_{\mathrm{vid}}$ into $z_{\tau_{\text{goal}}} \in \mathbb{R}^d$, while any text instruction y is mapped by $\mathrm{Enc}_{\mathrm{text}}$ to $z_y \in \mathbb{R}^d$.

Prior Model for Text-to-Visual Goals. To transform text embeddings into visual embeddings, STEVE-1 employs a conditional variational autoencoder (CVAE) that learns

$$p_\phi\big(z_{\tau_{\text{goal}}} \mid z_y\big),$$

where $z_{\tau_{\text{goal}}} = \mathrm{Enc}_{\mathrm{vid}}$ (16-frame video) and $z_y = \mathrm{Enc}_{\mathrm{text}}(y)$. Given a small dataset of $(\text{text embedding}, \text{clip embedding})$ pairs, this prior is trained to predict the visual embedding that best corresponds to each text embedding. Once trained, the prior can sample a visual embedding for the text embedding of any text prompt, thus acting as a text-to-visual "translator".

Policy Network (Finetuned VPT). STEVE-1's policy is derived from the large-scale VPT model, which is trained on 70k hours of YouTube Minecraft gameplay. VPT uses a ResNet to process 128×128 RGB frames into a feature vector x_t and a Transformer-XL to autoregressively predict the next action. STEVE-1 then introduces a small affine transformation that injects the MineCLIP goal embedding $z_{\tau_{\text{goal}}}$ into the policy:

$$x_t \;\rightarrow\; x_t \;+\; W\, z_{\tau_{\text{goal}}} \;+\; b,$$

where W and b are learned parameters. The Transformer-XL then outputs a distribution over actions, conditioning on both the observed frames and the desired goal embedding.

Packed Hindsight Relabeling for Policy Training. The policy is finetuned via behavioral cloning on an offline dataset of collected Minecraft gameplay. Rather than requiring explicit text labels for every segment, STEVE-1 uses a self-supervised technique called *packed hindsight relabeling*. Given a trajectory of frames and actions $(o_1, a_1, \ldots, o_T, a_T)$, we randomly select k timesteps $i_1, i_2, \ldots, i_k$ and retrieve the MineCLIP embedding $z_{\tau_{i_j}}$ from a 16-frame snippet preceding each chosen i_j. For all frames t in the range $[i_{j-1}+1,\ i_j]$, the sub-goal is assigned to $z_{\tau_{i_j}}$. This effectively teaches the policy to produce the same actions that reached the future state depicted in the snippet by splitting up the task into multiple subgoals.

Inference and Classifier-Free Guidance. At test time, users provide a text prompt y (e.g., "Chop down a tree"). The prior model samples a visual embedding $z_{\tau_{\text{goal}}}$ based on z_y, and the policy executes low-level actions (mouse/keyboard) conditioned on that embedding. STEVE-1 also adapts *classifier-free guidance*, whereby the policy logits are blended between a conditional pass (goal embedding present) and an unconditional pass (goal embedding dropped), allowing finer control over how strictly the agent follows the prompt.

By decomposing the text-to-behavior problem into a text-to-visual translation step (the CVAE prior) and a visual goal-conditioned policy (finetuned VPT), STEVE-1 achieves flexible instruction-following without requiring an extensive labeled dataset. This approach demonstrates the power of leveraging large pretrained models for sequential decision-making in complex environments like Minecraft. Despite these strengths, it is important to note that STEVE-1 is constrained by the content of its training set. It excels at following instructions observed in training but struggles with novel or multi-step compositional taks.

3.2 Voyager

Voyager casts Minecraft primarily as a program generation problem. Rather than emitting keyboard or mouse events, the agent writes and executes short JavaScript *skills* that call a restricted high-level Minecraft API (e.g. block queries, path-finding, crafting, inventory management). Whenever a skill succeeds, its code and a one-line description are stored into a reusable skill library.

Control loop (one iteration every $\approx$ 60 in-game seconds).

1. *State acquisition.* Structured metadata (position, biome, inventory, nearby mobs, ...) is read via the Mineflayer API and summarised into one sentence σ_t by a small language model.
2. *Skill retrieval.* The summary σ_t is converted to a dense vector, and the k most similar skill descriptions $\mathcal{K}_t$ are fetched from the library with a vector-similarity search.
3. *Prompting GPT-4.* A prompt containing the JSON game state, σ_t, the retrieved skills $\mathcal{K}_t$, the open objective list, and any previous error log is sent to GPT-4.

4. *Sandbox execution.* The generated code runs in a Node.js sandbox (3s timeout). Compile or runtime errors are fed back to GPT-4 for up to two automatic repair attempts.
5. *Intrinsic reward & archiving.* Voyager sums (i) XP gained, (ii) a bonus for genuinely new skills, and (iii) a novelty score based on diversity of achievements (as in MineDojo (Fan et al. 2022)). If the total exceeds a threshold δ, the new skill and its description are added to the library.

Skill Library. A key distinctive feature of Voyager is its *skill library*. Each time the agent successfully completes a new sub-task, the corresponding code snippet is saved by prompting GPT to describe function, then storing an embedding of the description and the original generated program as a key value pair. When doing tasks in the future, we can then retrieve skills the skill library by getting a text embedding of the current task and environment and searching the accumulated skill library.

Automatic curriculum Because Voyager lacks an external reward signal or specific task curriculum, it relies on a self-driven curriculum that maximizes exploration. The LLM planner suggests high-level goals according to the environment and metadata it is provided (e.g., "build a shelter," "collect diamonds," or "explore the Nether"). This process can unfold indefinitely, as new exploratory goals emerge from the agent's accumulated knowledge. The open-ended paradigm of Voyager thus mirrors human discovery, in which novel objectives arise organically from previous achievements and perceived environment advantages.

Architectural Trade-offs Despite its strengths in symbolic planning and self-directed task construction, Voyager exhibits several key limitations that restrict its generality and expressive capacity. Its core mechanism for goal checking is largely heuristic: tasks are considered complete if certain items appear in the agent's inventory or if predefined environmental markers are satisfied. This reliance on hard-coded indicators introduces a narrow success criterion and discourages improvisation or open-ended interaction. Furthermore, Voyager's self-curriculum is largely structured around climbing the Minecraft tech tree—collecting tools, crafting items, progressing toward armor or advanced machinery. While this resembles human progression in the game, it also reveals a lack of creative flexibility. The agent does not engage in aesthetic building, narrative roleplay, social interaction, or any emergent behavior outside of material accumulation. In essence, its "exploration" is bounded by a fixed progression of crafting-related goals rather than a rich, evolving world model. Unlike truly creative agents, Voyager does not reframe goals, invent new structures, or interpret ambiguous environments with flexibility. Its symbolic intelligence is powerful, but also rigid—highlighting the need for agents that can reason not only about success conditions, but also about meaning, intention, and possibility.

3.3 r1 and r2

r1 is a lightweight vision-language agent that operates by prompting GPT-4V on periodic snapshots of the environment to generate natural language commentary. It does not perform actions or maintain long-term memory. Implementation details are provided in the Appendix. With its simplicity, r1 serves as a minimal baseline to isolate the impact of perception and narration, without action or memory

r2 builds on r1 by adding actual gameplay control. It combines r1's screenshot-to-instruction pipeline with STEVE-1's ability to carry out tasks using a goal-conditioned policy network. In short: GPT-4 figures out what to do, and STEVE-1 figures out how to do it.

Architecture. At each decision cycle, r2 captures the latest gameplay frame and recent context, and prompts GPT-4V to generate a low-level task description (e.g., "collect wood"). The generated task is translated into a MineCLIP goal embedding, which conditions a VPT-based policy network derived from STEVE-1. The policy then produces low-level mouse and keyboard actions to execute the task within the environment.

Coordination Between Planning and Execution. GPT-4V serves as a high-level planner, proposing environment-dependent goals based on visual observations. The STEVE-1 policy module acts as the low-level controller, transforming these goals into continuous action sequences. Upon task completion or failure, r2 captures a new observation, updates its context window, and initiates the next planning-execution cycle.

Strengths and Limitations. r2 substantially improves over r1 by enabling active movement, resource collection, and environment interaction. However, it remains constrained by snapshot-based perception without persistent world memory. Its planning horizon is short, and task success heavily depends on the specificity and accuracy of the goals generated by GPT-4V. Misalignment between generated goals and executable behaviors can result in suboptimal performance, particularly in dynamic or ambiguous environments.

4 Comparison and Evaluation Framework

Some aspects of agent behavior—such as memory limits, action modalities, and input structure—can be inferred directly from the model's architecture. These are fixed architectural choices that do not require empirical testing but still shape the agent's potential capabilities.

Notation

A_{out}: audio output generated by the agent

A_{in}: audio the agent receives from an external source

K: keyboard in some representation that is model specific

M: mouse

V: video

T: text

Memory depth. The *Markov order* column shows how many past observations an agent can condition on. STEVE-1 and r2 inherit a VPT-style Transformer with a limited attention window, so their effective memory is "variable but short." Voyager is formally first-order (GPT-4 sees only the current

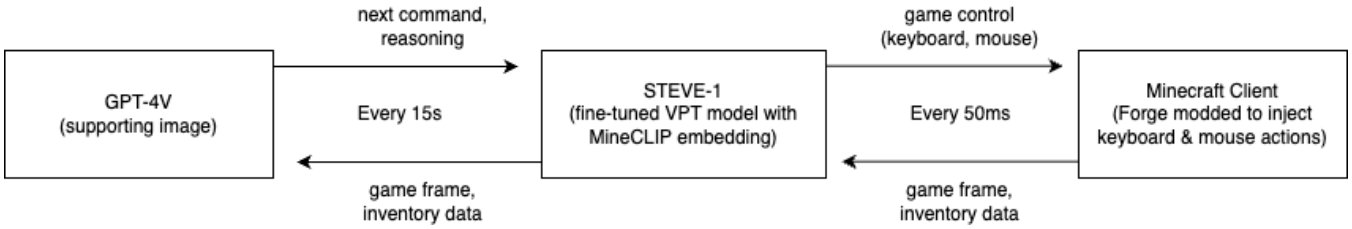

Figure 1: r2 architecture overview.

Table 1: Capabilities of AI Models for Minecraft

Model	Markov Order	Learning	Observation Space	Action Space
STEVE-1	Variable(short)	No	T_{prompt}, V	K, M
Voyager	Unbounded	Yes	$T_{\text{Minecraft world data}}$	T_{code}
r1	1	No	A_{in}, V	A_{out}
r2	Variable(short)	No	$T_{\text{inventory data}}, V$	A_{out}, K, M
Human	Unbounded	Yes	A_{in}, A_{out}, K, M, V	A_{out}, K, M

symbolic state) but it maintains a growing skill library for external long-term memory for code.

Online learning. Only *Voyager* (via continual skill-library updates) and humans adapt during deployment. All other agents are frozen policies: once inference starts, no new weights or behaviors are added.

Sensing and acting. The two right-most columns expose each model's embodied bandwidth. STEVE-1 and the execution side of r2 rely on raw pixels (V) and low-level motor commands (K, M). Voyager, by contrast, sees rich structured data by directly accessing Minecraft metadata ($T_{\text{Minecraft world data}}$) but not images or sound. It "acts" by emitting code (T_{code}) rather than direct keyboard and mouse. r1 is the opposite: a thin visual snapshot plus microphone input (A_{in}, V) and language output only (A_{out}).

Broadly, there are two types of evaluation: programmatic comparison, which focuses on objective performance metrics, and observational comparison, which centers on human perceptions during real-time interaction. Given our interest in how agents are perceived—not just how they perform—we chose the observational approach. This allowed us to study agents in a more naturalistic, multiplayer setting. To that end, we conducted a series of in-person interactions with agents in the PLAICraft environment, where participants could directly engage with each agent and provide feedback on their experience.

4.1 Methodology for Observational Comparison

To assess player perceptions of integrated agents in Minecraft within the PLAICraft server, participants were recruited to play and interact with the four agents one at a time. They were instructed to observe and interact with the agents using in-game chat and/or voice, depending on the agent's capabilities. STEVE-1 and r2 performed low-level embodied tasks based on fixed or text-provided goals. Voyager and r1 engaged in higher-level planning or conversational responses. Participants were encouraged to explore freely, observe behavior, and communicate with each agent as appropriate. These differences gave participants a chance to experience how different agent architectures influenced reactivity, embodiment, and believability. Interactions averaged 40 minutes per agent.

Questionnaire design To evaluate players' subjective impressions of different embodied AI agents, we designed a post-interaction questionnaire aimed at capturing human perceptions of intelligence, presence, and collaboration in a live Minecraft environment. The questionnaire includes 4 sections and took 20 minutes to complete. It collects both structured and open-ended responses across several key dimensions:

- **Embodiment and Presence:** Questions assess whether the AI felt human-like, emotionally engaging, and immersive within the game world.
- **Gameplay and Task Performance:** Participants rate the agent's usefulness, autonomy, environmental awareness, and ability to collaborate.
- **Communication:** This section evaluates the clarity, timeliness, and naturalness of interactions via speech or text.
- **Open Feedback:** Participants are invited to describe moments of surprise, frustration, or believability, and offer suggestions for improvement.

This feedback was essential for evaluating not just an agent's functional competence, but its perceived social presence, cooperative behavior, and believability as a real character.

Ethical Statement. The PLAICraft environment in which the questionnaire were conducted received ethics approval from our institution's behavioral research ethics board. All participants provided informed consent before interacting with the agents and completing the questionnaire. No personally identifying information was collected, and participation was voluntary.

Category	r1	r2	STEVE-1	Voyager
Human-likeness	2.88	**3.62**	3.29	2.11
Body motion matching task	1.38	**4.12**	3.71	3.89
Emotional reaction to agent	**3.75**	3.50	2.43	1.67
Immersion	2.75	**3.25**	2.43	1.67
Helpfulness in tasks	2.25	**3.25**	3.00	3.22
Environmental awareness	3.00	**3.38**	3.14	3.33
Communication clarity	1.62	1.25	**3.43**	1.56
Timely responses	3.00	2.25	**3.71**	1.56
Language naturalness	**3.62**	3.25	1.14	1.33
Felt "present"	**2.88**	1.25	2.71	1.22
Comfortable to interact again	**3.50**	3.12	2.86	2.00

Table 2: Average Player Ratings of AI Agents (1 = low, 5 = high)

5 Results

We collected questionnaire responses from 9 participants. Each participant evaluated the agents along multiple dimensions, including perceived human-likeness, emotional engagement, task helpfulness, communication clarity, and immersion.

5.1 Structured Responses.

Table 2 summarizes the average scores (out of 5) for key aspects across the four AI agents. The highest scoring model for each category is in bold. Participants rated r2 highest overall in human-likeness, body motion realism, and environmental awareness, likely reflecting its combination of GPT-driven planning and STEVE-1's fine-grained embodied control. r1 elicited the strongest emotional reactions and was rated most natural in language use, despite its limited embodiment. STEVE-1 demonstrated strong responsiveness and clarity in understanding typed commands but was less effective at higher-level dialogue or long-horizon tasks. Voyager achieved high scores in task helpfulness but was perceived as less emotionally engaging and less immersive, consistent with its architecture's emphasis on symbolic planning rather than embodied presence.

5.2 Open Ended Observations.

Open-ended questionnaire responses provided further insight into participants' experiences. Several players highlighted moments where the AI felt unexpectedly intelligent, such as when an agent "saw a spider in a tree and reacted" or "broke down a high-level task into smaller steps." Positive comments often focused on the AI's ability to engage conversationally or carry out visible in-game tasks, with players noting they "enjoyed asking it questions about its environment" or "watching it complete its own menial tasks." However, participants also expressed frustrations around slow response times, limited awareness, and agents occasionally "getting stuck in loops" or "taking a long time to figure out what to do next." These qualitative impressions reinforce the trends observed in the quantitative ratings, emphasizing the importance of both real-time embodiment and coherent planning in shaping players' sense of agency and immersion. These findings highlight the varied strengths and weaknesses of different architectural approaches to embodied AI in Minecraft, motivating a deeper analysis of the trade-offs between planning, embodiment, and adaptability in open-world environments.

6 Discussion

Our study findings suggest that players respond most positively to agents that combine embodied reactivity with coherent goal pursuit. High-level planners like Voyager were often rated as useful but emotionally flat, while agents with visible control over their bodies (like r2 and STEVE-1) felt more "alive" even when less successful at task completion. This highlights the importance of embodiment and moment-to-moment responsiveness in shaping perceptions of intelligence and presence. In this section, we examine the architectural challenges and opportunities surfaced by our comparative study.

The Challenge of Reinforcement Learning in Minecraft. Minecraft presents unique challenges for reinforcement learning (RL): a vast state and action space, extremely sparse rewards, and long episodic horizons. Traditional RL approaches struggle under these conditions, often flailing aimlessly without structured feedback. VPT proposed a solution by pretraining on 70,000 hours of human gameplay, capturing early-game behaviors through behavioral cloning. However, pure imitation remains insufficient for long-horizon reasoning and abstract planning, leading to early plateaus in agent progression.

STEVE-1: Building on the VPT Foundation. STEVE-1 extends VPT by introducing goal-conditioning through MineCLIP and packed hindsight relabeling. Rather than requiring explicit instruction labels, STEVE-1 retrospectively assigns visual sub-goals based on future frames, enabling flexible instruction following without dense reward shaping. However, because MineCLIP embeddings are based on short 16-frame clips, STEVE-1's planning horizon remains limited to local objectives, and it inherits VPT's challenges with long-term reasoning.

STEVE-1's Limitations. Although STEVE-1 uses a Transformer-XL model for temporal processing, its memory is limited to a short sequence of recent frames. This makes it difficult for the agent to reason over longer timescales or keep track of extended goals. Even when conditioned on a MineCLIP goal embedding, the policy does not explicitly plan multiple steps ahead—it responds reactively, optimizing behavior in the short term without long-horizon structure.

Packed hindsight relabeling improves training efficiency and control over local behavior, but it doesn't give the model actual foresight or hierarchical planning ability. In this way, STEVE-1 meaningfully extends VPT, but inherits its core limitation: it performs well at short tasks, but fails to generalize across long-term narratives or high-level strategies.

Voyager: Symbolic Planning Without Embodiment. Voyager adopts a fundamentally different strategy, using

GPT-4 to generate executable code snippets that interact with Minecraft via high-level APIs. This enables flexible symbolic planning, automatic skill accumulation, and generalization across a wide range of tasks. However, Voyager's reliance on structured metadata and heuristic goal-checking (e.g., inventory changes, coordinate targets) introduces fragility: brittle code execution and the absence of grounded perceptual feedback limit the agent's reactivity, creativity, and narrative flexibility in open-ended environments. Planning is effectively decoupled from low-level control, resulting in strong task decomposition but reduced robustness and adaptability during dynamic gameplay.

Voyager's Limitations. Despite its strong planning capabilities, Voyager struggles with fragile execution. Even with safeguards like timeouts and error correction, code generation remains inherently brittle: invalid or looping programs can slow down or disrupt progress, especially during longer or more complex tasks. Because Voyager operates purely on symbolic state information and lacks direct perceptual grounding, it can lose coherence over time, making its behavior unstable in unpredictable environments.

Goal checking also imposes important constraints. Voyager considers tasks complete based on simple heuristics, such as changes to inventory, reaching a coordinate, or detecting specific blocks. While effective for structured objectives, this approach limits creativity and adaptability in more open-ended situations where success is harder to define. Voyager cannot reinterpret goals or improvise when the environment changes—it relies on a static checklist rather than building a flexible internal model of the world.

r1 and r2 Limitations. r1 serves as a lightweight baseline, combining vision and language to comment on the Minecraft environment without embodied control. Despite lacking agency over the game world, r1 was rated highly for emotional engagement and natural language use, showing that language models can still feel personable even when limited to static images. However, r1's lack of movement, delayed reactions, and missing temporal continuity sharply limited players' sense of immersion and realism.

r2 extends r1 by incorporating STEVE-1's goal-conditioned policy network for real-time control. GPT-4V serves as the high-level planner, interpreting screenshots and proposing actionable goals, while STEVE-1 executes them at the keyboard and mouse level. This modular combination yielded strong ratings in human-likeness, body realism, and task helpfulness, confirming the value of connecting large language models to grounded embodied policies. Nevertheless, r2 inherits some of the same limitations as r1: it relies on single-frame screenshots fed into GPT-4V for decision-making, limiting its temporal awareness. As a result, it can misjudge the current state—for example, attempting to re-complete tasks that are no longer visible or forgetting recently executed actions—due to the absence of memory or video context.

7 Limitations and Future Work

Sample Size. Our study included a small sample size of participants, which limits the generalizability of the results. While the questionnaire captured a range of experiences, a larger sample size would be needed to draw statistically robust conclusions.

Models. Several recent agents—such as Jarvis-VLA (Li et al. 2025a) and Optimus-3 (Li et al. 2025b)—have achieved state-of-the-art performance on diverse Minecraft benchmarks through improved grounding, planning, and post-training. Future work should include them to assess whether their improved task success translates to improved human perception in multiplayer settings.

Collaborative Agents. Recent work has also explored agent-to-agent interaction in open-ended environments. The MINDCraft platform and MineCollab benchmark (White* et al. 2025) evaluate how LLM agents cooperate in Minecraft. Results suggest that natural language communication remains a major bottleneck for collaboration. Extending PLAICraft to support agent-agent interaction may help test these limits under live multiplayer conditions.

Morgan: A Unified Embodied Agent. A concurrent research effort is developing Morgan, a recurrent, multimodal agent that unifies perception, memory, and control into an end-to-end architecture. Unlike modular systems such as r2 or Voyager, Morgan is trained on continuous gameplay data in PLAICraft and aims to support temporally grounded action through persistent internal state. While not evaluated in this study, Morgan reflects the architectural direction needed to address the limitations of snapshot-based perception and modular planning. We plan to include Morgan in future evaluations as its development progresses.

8 Conclusion

We reviewed, implemented, and tested several leading Minecraft AI models in a real (non-simulated) Minecraft environment. Through an interactive questionnaire, we gathered player feedback to assess how humanlike the agents appeared. Although some models showed promise in specific areas, the questionnaire results consistently highlight that significant challenges remain in achieving truly humanlike, interactive behavior. This study reveals consistent patterns in how players perceived each agent's capabilities, especially in terms of believability, reactivity, and multimodal presence. These findings suggest that building robust embodied AI for open-ended worlds like Minecraft requires more than local control or high-level planning in isolation. Reaching the level of human players will require agents to combine grounded perception, long-horizon memory, and adaptive reasoning in a single, unified system.

References

Baker, B.; Akkaya, I.; Zhokhov, P.; Huizinga, J.; Tang, J.; Ecoffet, A.; Houghton, B.; Sampedro, R.; and Clune, J. 2022. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos. arXiv:2206.11795.

Fan, L.; Wang, G.; Jiang, Y.; Mandlekar, A.; Yang, Y.; Zhu, H.; Tang, A.; Huang, D.-A.; Zhu, Y.; and Anandkumar, A. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Li, M.; Wang, Z.; He, K.; Ma, X.; and Liang, Y. 2025a. JARVIS-VLA: Post-Training Large-Scale Vision Language Models to Play Visual Games with Keyboards and Mouse. *arXiv:2503.16365*.

Li, Z.; Xie, Y.; Shao, R.; Chen, G.; Guan, W.; Jiang, D.; and Nie, L. 2025b. Optimus-3: Towards Generalist Multimodal Minecraft Agents with Scalable Task Experts. *arXiv preprint arXiv:2506.10357*.

Lifshitz, S.; Paster, K.; Chan, H.; Ba, J.; and McIlraith, S. 2023. STEVE-1: A Generative Model for Text-to-Behavior in Minecraft.

Long, Q.; Li, Z.; Gong, R.; Wu, Y. N.; Terzopoulos, D.; and Gao, X. 2024. TeamCraft: A Benchmark for Multi-Modal Multi-Agent Systems in Minecraft. *arXiv preprint arXiv:2412.05255*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv: Arxiv-2305.16291*.

Wei, Z.; Lin, B.; Jiao, Z.; Nie, Y.; Ma, L.; Liu, Y.; Zhuang, Y.; and Liang, X. 2025. MineAnyBuild: Benchmarking Spatial Planning for Open-world AI Agents. arXiv:2505.20148.

White*, I.; Nottingham*, K.; Maniar, A.; Robinson, M.; Lillemark, H.; Maheshwari, M.; Qin, L.; and Ammanabrolu, P. 2025. Collaborating Action by Action: A Multi-agent LLM Framework for Embodied Reasoning. *arXiv preprint arXiv:2504.17950*.

Zheng, X.; Lin, H.; He, K.; Wang, Z.; Zheng, Z.; and Liang, Y. 2025. MCU: An Evaluation Framework for Open-Ended Game Agents. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

Appendix

9 Integration of AI Models into PLAICraft

Deploying AI models originally designed for controlled or simplified environments into the real Minecraft environment introduces distinct technical challenges, necessitating careful engineering and optimization. This section provides a comprehensive overview of the integration process for two AI models, Voyager and STEVE-1, detailing the specific adaptations and technical considerations involved in enabling these models to function as intended within the live Minecraft environment hosted on the PLAICraft server.

9.1 Voyager Integration

Voyager was originally designed with the capability to operate in a real Minecraft environment, utilizing high-level API calls and metadata. The integration process was relatively straightforward, involving resolving minor versioning and dependency issues on Ubuntu. Specifically, the primary challenges included ensuring compatibility between GPT-generated JavaScript code snippets and the in-game command interface, as well as fine-tuning the skill library to efficiently execute high-level tasks within the live server environment.

9.2 STEVE-1 Integration

STEVE-1 was originally developed to operate within MineRL, a simplified and discretized version of Minecraft made specifically as a test bed for RL agents. Integrating STEVE-1 into real Minecraft thus required some engineering effort, primarily to overcome latency and real-time interaction constraints inherent in real Minecraft:

Forge Mod Development. A custom Forge mod was created to facilitate communication between the STEVE-1 Python runtime environment and Minecraft's Java-based game engine. This mod directly hooks into the game loop, capturing video frames from the game in real-time to send to the policy network, and low-level action commands injected back into the game.

Shared Memory Buffers. STEVE-1 is trained on data of 20FPS Minecraft. So each frame action injection has a 50ms window to read the current screen, feed it into the model, get the action back from the model, then inject the action back into the game. It is critical to reduce latency to ensure we aren't skipping frames.

To reduce latency, a shared memory buffer system was implemented. This enabled efficient and rapid data exchange between the Python policy network and the Java runtime environment, drastically cutting down communication overhead compared to traditional network-based methods.

Latency Management. Addressing latency was a critical task due to STEVE-1's reliance on fine-grained, low-level control signals (keyboard and mouse inputs). Multiple adjustments, including synchronizing data capture with Minecraft's internal tick rate and optimizing buffer size and serialization processes, were implemented to ensure STEVE-1's actions remained responsive and realistic.

Limitations. Since STEVE-1 lacks autonomous high-level planning, action goals were manually input via terminal by the experimenter during evaluation. This ensured the policy network had a clear instruction to follow, but highlights a limitation compared to fully integrated systems like r2.

9.3 r1 Integration

r1 is a lightweight agent designed to test basic vision-language grounding within Minecraft. It operates without direct control capabilities, serving instead as an observational system that interprets periodic snapshots of the environment through language-based reasoning.

Control Loop. At each cycle, r1 performs the following steps:

1. Captures the latest gameplay frame from a shared memory buffer,
2. Records incoming audio from Simple Voice Chat,

3. Packages the visual and audio inputs, along with recent interaction history, into a prompt sent to GPT-4o,
4. Parses the returned natural language description or response,
5. Converts the response into synthesized speech and plays it back through the game client.

Architecture. r1 maintains a short conversation history to provide minimal temporal context across interactions. However, it does not build or maintain an internal world model. The system operates purely on static frame-based inputs and produces natural language outputs without direct manipulation of the Minecraft environment through keyboard or mouse actions.

Limitations. Due to its reliance on periodic static frames and limited memory, r1 lacks persistent situational awareness and temporal continuity. It does not engage in direct interaction with the environment and is restricted to passive commentary. Latency introduced by visual encoding and large language model inference further constrains real-time responsiveness. Nonetheless, r1 establishes a baseline for evaluating vision-language alignment and the ability of large language models to interpret and communicate about complex visual environments.

9.4 r2 Integration

r2 extends the basic r1 pipeline by adding real-time action execution through STEVE-1. Integrating these components into the live PLAICraft server environment required coordinating high-frequency motor control with slower high-level planning.

Control Flow Integration. r2 operates on two tightly coupled timescales:

- **Low-level control (50 ms).** Every game tick a custom Forge mod dumps the current 128×128 RGB frame to a video shared-memory buffer and, in parallel, serialises the complete 46-slot inventory (item IDs and stack counts) to a *separate* structured-data buffer. STEVE-1 consumes the video stream, produces a mouse/keyboard action via its VPT-based policy, and the action is injected back into Minecraft before the next tick.
- **High-level planning (15 s).** Once every 15 seconds r2 takes a snapshot of the latest video frame together with the most recent inventory JSON, wraps both into a single prompt, and sends it to GPT-4V. GPT-4V returns a low-level task description (e.g. "collect wood"), which is converted into a MineCLIP goal embedding that steers STEVE-1 until the next planning step.

This two-tiered system allows r2 to execute fine-grained real-time actions while periodically updating its goals based on larger shifts in the environment.

Inventory Instrumentation. GPT /4V is still unreliable at recognizing fine-grained inventory details from pixels alone. To guarantee accurate state, we wrote a custom Forge mod that, every game tick ($\approx$ 50 ms), extracts the full 46-slot inventory (item ID and stack size) and dumps it to a *separate* shared-memory ring buffer dedicated to structured data. A small daemon converts the latest dump into a one-sentence summary that is appended to the GPT-4V prompt. Without this explicit channel the planner often mis-identified held items and produced infeasible goals. The need for such a workaround underscores a current limitation of vision- only LLM planners: while they handle coarse navigation reliably, they still require privileged metadata for precise, object-level reasoning such as inventory management.

Shared Memory and Latency Management. Frame capture and action injection for STEVE-1 are handled via shared memory buffers to minimize communication overhead. The 15-second high-level planning cycle was manually tuned to balance responsiveness with computational cost, ensuring that GPT inference latency did not interrupt continuous low-level control.

Audio Integration. Simple Voice Chat is used to record ambient audio inputs, similar to r1. However, in r2, audio inputs are not used to influence task generation or control policies and are stored passively for potential future expansion.

Limitations. Because high-level planning only occurs every 15 seconds, r2 may continue pursuing a task even if it becomes infeasible within that window. Additionally, r2 inherits the snapshot-based limitations of r1: it lacks persistent video memory and can miss important temporal cues between high-level goal updates. These constraints point toward future directions involving streaming video inputs and longer-term memory mechanisms.