Vision Voyager: Minecraft agent with open-Source LLMs and vision models

Anasse El Boudiri (374212), Jan Steiner (374940)
CS-503 Final Project Report

***Abstract*—In this project, we extend the Voyager framework for autonomous agents in Minecraft by incorporating visual perception and supporting open-source large language models (LLMs). While Voyager demonstrated strong performance using GPT-4 and text-based state representations, it lacked visual grounding and relied on a closed-source, high-cost model. To address these limitations, we replaced GPT-4 with GPT-4.1 and open-source alternatives using the Ollama framework, allowing for easy model switching. We also integrated a vision pipeline using a first-person screenshot mechanism and vision-language models (VLMs) to enrich the agent's perception and reasoning. Vision was selectively injected into the Curriculum and Critic agents to improve spatial understanding without overwhelming the LLM prompts. Our experiments show that while quantitative gains from vision were modest, qualitative improvements were significant: agents demonstrated more coherent task sequences and built more structured environments. Despite some stability challenges such as execution bugs and prompt complexity, this work lays the foundation for cost-effective, multimodal embodied agents. It also offers valuable insights into integrating open models and vision in interactive environments, paving the way for future research in open-ended, vision-enabled AI agents.**

I. Introduction

Minecraft has emerged as a valuable playground for embodied AI research due to its open-ended gameplay and complex, survival-oriented world. Agents in Minecraft can practice exploration, resource gathering, crafting, and construction. Skills analogous to those required in real-world robotics and autonomy. Recent work Voyager [1] demonstrated the power of LLM-based agents in Minecraft: by interacting with GPT-4 as a planning and coding oracle, Voyager learned an extensive skill library and achieved state-of-the-art exploration and survival capabilities. Voyager's success highlighted how an LLM can generate flexible plans and code for an open-world agent. **However, the approach has two key limitations.**

First, it relies on GPT-4 an expansive proprietary model usage costs and rate limits. This dependence hampers deployment and experimentation by researchers without access to GPT-4. In fact, follow-up analysis showed that replacing GPT-4 with a weaker model (GPT-3.5) in the Voyager framework caused a drastic drop in performance, underscoring the challenge of using smaller or open models.

Second, Voyager lacked visual perception; it could not see the Minecraft world, instead relying on text-based feedback (like game information and code execution results). Without vision, the agent may misinterpret the spatial context or fail to perform obvious actions (e.g mining a diamond directly in front of it) and tends to struggle with tasks that require accurate visual reasoning, such as constructing complex structures.

Text-only Voyager (GPT-4.1) Vision Voyager (GPT-4.1)

Figure 1. Comparison of Voyager's performance with and without vision on the house-building task.

Motivated by these issues, our project aims to make the model choice flexible, open-source the Voyager agent's intelligence, and enhance it with vision capabilities. We replace GPT-4 with GPT 4.1 and open-source LLMs that can run locally, making the agent more accessible and cost-effective. In addition, we integrate a vision loop: the agent periodically captures an image of its first-person view and uses a vision-language model to interpret the scene (for example, checking placement of blocks). By fusing textual and visual feedback, the agent can verify its actions, take inspiration from environment and adjust plans; for instance, confirming that a built structure matches the intended design. We hypothesize that visual reasoning will particularly improve performance on tasks like navigation and construction, where textual state descriptions are insufficient.

This approach has broader implications: it advances the development of autonomous agents that are both cost-effective and multimodal, bringing us closer to real-world robotic applications. Embodied AI systems equipped with onboard, open-source intelligence could be more easily deployed in field robotics or simulators. We believe that LLM-powered agents hold significant potential for robotics and interactive gaming, and we hope to encourage further research at the intersection of language, vision, and embodied autonomy.

II. Related Work

Minecraft Embodied Agents: MineDojo [2] introduced a massive simulation suite in Minecraft with thousands of free-form tasks and a rich knowledge base for agents.

To evaluate agent success on arbitrary goals, MineDojo included MineCLIP, a video-language model that aligns gameplay video clips with text descriptions, enabling reward feedback for open-vocabulary tasks. Building on this environment, Voyager [1] was recently proposed as the first LLM-powered lifelong learning agent in Minecraft. Voyager uses GPT-4 (via API calls) to continually generate and refine Python code "skills" that the agent executes to act in the world. It features an automatic curriculum (setting its own progressively harder goals), an iterative prompting loop with self-verification for correcting errors, and a persistent skill library to accumulate behaviors. Voyager demonstrated impressive performance, vastly outperforming prior reinforcement learning agents on metrics like unique items collected and tech-tree milestones unlocked. A crucial drawback, however, is the dependency on the GPT-4 API (closed-source and expensive), and the fact that Voyager has no visual input; it relies on high-level state info and game engine feedback, not raw pixels. This means it cannot directly interpret the rich 3D visual scene.

Vision-Language Models: To incorporate vision into AI decision-making, recent multimodal LLMs combine image encoders with language models. MiniGPT-4 [3] aligns a frozen visual encoder to a LLM (Vicuna) using a single projection layer, resulting in a model that can accept images and produce detailed descriptions or answers (mimicking some abilities of GPT-4's vision model). Similarly, LLaVA (Large Language and Vision Assistant) [4] is an end-to-end trained VLM that connects a CLIP-based vision encoder with a Vicuna LLM, achieving impressive general-purpose image understanding and instruction following. These works show that open-source multimodal models can be built to approach GPT-4's capabilities. However, integrating such VLMs into an embodied agent loop (where an agent continuously perceives and acts in a dynamic 3D world) remains relatively unexplored. Our project draws inspiration from these VLMs to give Voyager a visual sense, and is novel in deploying a full Minecraft agent using only open-source models for both language and vision. We also emphasize cost-awareness: by substituting GPT-4 with local or cheaper alternatives, we make long-term exploration feasible on a student budget, whereas prior setups would cost tens of dollars per run.

Our Approach – Open LLMs + Vision in Embodied Agents: Our project bridges both language models and visual perception in an embodied setting. We also conduct task-specific evaluations (e.g. counting items collected, success in more visual tasks) to quantify the impact of adding vision, extending the analysis of prior studies. This contributes toward developing autonomous agents that are more accessible and potentially transferable to physical robotics scenarios.

III. Method

In our approach, we replaced GPT-4 with GPT-4.1 (which is cheaper and multimodal).

A. Pipeline Overview

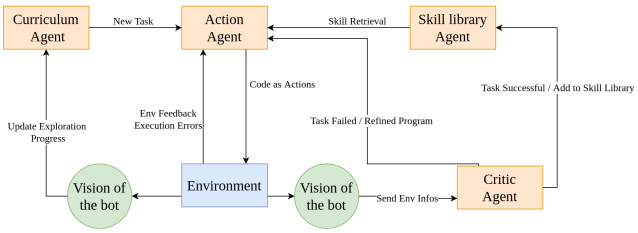

Figure 2. Vision voayger pipeline

In Vision Voyager, visual input is integrated at two key stages: the Curriculum Agent and the Critic Agent. Vision helps the Curriculum Agent better assess exploration progress by providing scene-level cues (e.g. recognizing biomes or nearby structures). The Critic Agent uses visual information to better evaluate task success or detect failures when execution errors alone are insufficient (e.g. misaligned structures, missing blocks). We avoid injecting vision directly into the Action Agent to keep prompts compact and reduce instability in LLM planning, especially with open-source models that are sensitive to long multimodal contexts.

B. Open-Source LLM Setup

To enable open-source LLMs in the Voyager loop, we integrated Ollama, an open-source LLM server, deployed on SCITAS. Ollama provides an OpenAI-compatible API, allowing us to easily replace GPT-4 calls with local models while keeping the original Voyager prompt structure and iterative prompting mechanism, ensuring fair comparison with the baseline. Initially, we experimented with smaller models such as Mistral 7B and LLaMA 2 13B, but they consistently failed even on basic tasks like crafting a pickaxe or navigating, often omitting crucial subtasks and hallucinating invalid Mineflayer API calls, sometimes getting stuck in infinite loops. These failures were likely due to limited Minecraft knowledge and insufficient reasoning capacity.

To address this, we moved to larger models while staying below 30B parameters (due to computational constraints). The best performing models were Codestral-22B (specialized for code generation) and Mistral-Small-3.1 (24B). We assigned Codestral specifically to the Skill Library Agent for generating and refining executable code, while Mistral-Small-3.1 handled the Curriculum, Iterative Prompting, and Critic Agents. Despite better reasoning, the open models still faced challenges such as limited context windows and occasional hallucinations, including generating incorrect or non-recoverable code loops.

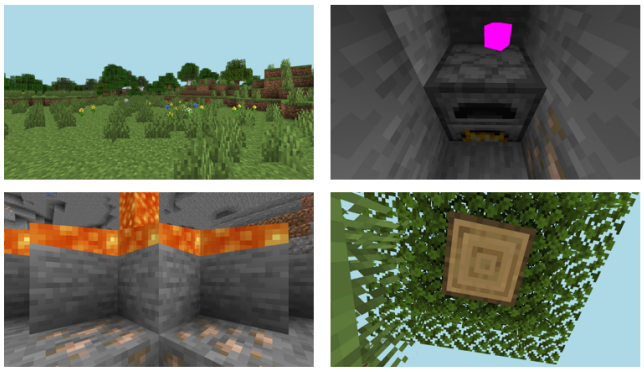

Figure 3. Examples of first-person view of the bot.

C. Restoring vision to the agent

We augmented the agent with a visual perception module that periodically provides an image of the agent's first-person view. This was implemented using Prismarine-Viewer (a web-based Minecraft world renderer) in combination with Puppeteer to automate screenshots. Concretely, the Minecraft bot (running with Mineflayer, as in Voyager) streams world state to a local web client. We position a virtual camera at the bot's location and orientation, then use Puppeteer (headless Chrome) to capture a PNG screenshot of what the bot "sees." This image capture is triggered every 10 seconds. The image is saved and passed into a Vision-Language Model for analysis.

D. Incorporating Vision Strategies

Adding vision to the agent is a natural extension to improve both realism and spatial reasoning. However, Voyager's architecture presents several challenges for integrating vision effectively. First, Voyager operates through iterative reasoning cycles, where each iteration generates an entire task plan rather than step-by-step actions. As a result, injecting vision at every low-level step would disrupt this structure and risk destabilizing the agent's planning loop. Visual inputs can only influence decisions between iterations, once full plans are executed and feedback is processed.

Second, Voyager already receives structured world state information from the Mineflayer API, including nearby entities, coordinates, and inventory content. In many cases, this structured input provides sufficient environmental awareness, reducing the added value of vision for routine tasks like resource collection. Vision becomes more relevant for higher-level spatial understanding, obstacle detection, or complex structure-building tasks.

To integrate vision while respecting Voyager's architecture, we evaluated several possible strategies and ultimately adopted a **frame sampling approach**. In this setup, we capture one or more representative first-person frames per reasoning cycle (typically at the end and after significant substeps). These images are directly injected into the VLM's prompt for the next reasoning iteration (see Figure 2). This method preserves Voyager's high-level reasoning loop while allowing the agent to leverage visual context without requiring real-time low-level perception. It also minimizes prompt length and avoids unnecessary visual inputs for simple tasks.

An alternative strategy we considered was the **Image Captioning + Text Prompt** approach. In this two-step pipeline, an external captioning model would first convert the image into a textual description (e.g., "The agent is facing a partially built wooden hut; the front wall has a gap"), which would then be injected into the LLM prompt as additional context. While viable, we decided against this approach since the first method was already fully integrated into our pipeline and better aligned with Voyager's reasoning design. Importantly, by directly feeding the image to the vision-language model, we let the model autonomously attend to relevant visual features rather than relying on a potentially lossy or biased caption. Captioning also introduces additional complexity, risks omitting subtle but important scene details, consumes extra context window tokens, and adds another model into the inference chain—further increasing latency and error propagation. Overall, the direct visual injection strategy provided a cleaner, more end-to-end solution for our agent's multimodal reasoning.

E. Challenges and Mitigations:

Integrating a smaller LLM and vision into Voyager was not without difficulties. One issue was hallucinations; open LLMs sometimes invented nonexistent Minecraft actions or misused Mineflayer API formats (e.g., calling unsupported actions). GPT-4.1 rarely encountered this, but open-source models required more guidance. Another challenge was crafting: complex crafting operations require using a crafting table with the correct recipe. The agent often forgot to place or use the crafting table, resulting in repetitive failures. Although open-source models handled text-based reasoning fairly well, adding vision introduced numerous failures (infinite loops, environment crashes). We suspect that adding image inputs significantly increased prompt complexity beyond what these models could handle. To still evaluate the vision component, we decided to test vision integration only with GPT-4.1. Even though vision is available for some open-source models, proper testing would require larger models, which we could not deploy.

IV. Experiments

We reproduce the same evaluation pipeline described in the original paper. This includes comparing the number of distinct items collected by each model (open-source and GPT-4.1). As mentioned earlier, vision evaluation was performed only with GPT-4.1 since open-source models struggled. We evaluated vision benefits on both a typical

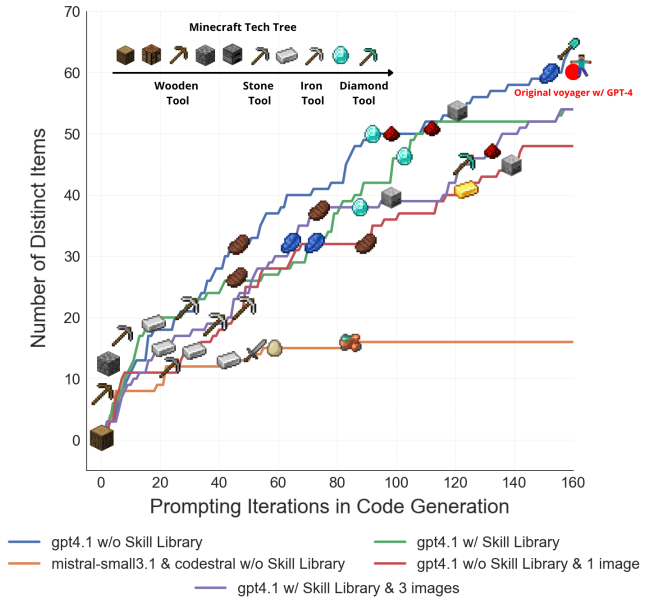

Figure 4. Self-driven exploration of new Minecraft items and skills via open-source LLMs and GPT4.1 (with and without vision)

Minecraft task (crafting a golden sword) and on house construction. To ensure fairness across all experiments, all agents were evaluated on the same Minecraft world seed and environment, eliminating potential variability due to world generation.

A. Minecreft tech tree

A first way to assess the contribution of vision is by comparing models with and without visual input in terms of exploration and efficiency. A simple yet informative metric is the number of distinct items collected per episode. As shown in Figure 4, mistral-small 3.1 and codestral perform significantly worse than GPT-4.1. We also tested these models with both the skill library and vision, but performance remained similar; to maintain clarity, these results are omitted from the figure.

Notably, our experiments use GPT-4.1, while the original Voyager relied on GPT-4. Despite this difference, GPT-4.1 performs comparably, if not slightly better, in terms of item collection. Among the GPT-4.1 variants, we observe only minor differences between configurations with and without the skill library and vision.

Interestingly, the best-performing setup was GPT-4.1 without vision or the skill library, an unexpected outcome given Voyager's original findings. This could be due to randomness in exploration or to noise introduced when vision and the skill library are not tightly integrated with the agent's planning processes.

Nonetheless, qualitative observations highlight clear benefits from incorporating vision, especially alongside the skill library. Vision helps the agent better interpret its environment and execute logically ordered actions. For example, agents using vision often crafted a diamond pickaxe immediately after mining diamonds, suggesting a higher-level understanding of item utility within the game's tech tree.

In contrast, text-only agents tended to perform actions in a less structured way due to limited situational awareness. While these benefits may not be fully captured by current metrics, they reflect improved task coherence and semantic understanding made possible through visual grounding. Future work should aim to better quantify these effects and refine the use of vision for more consistent improvements.

B. Craft a golden sword

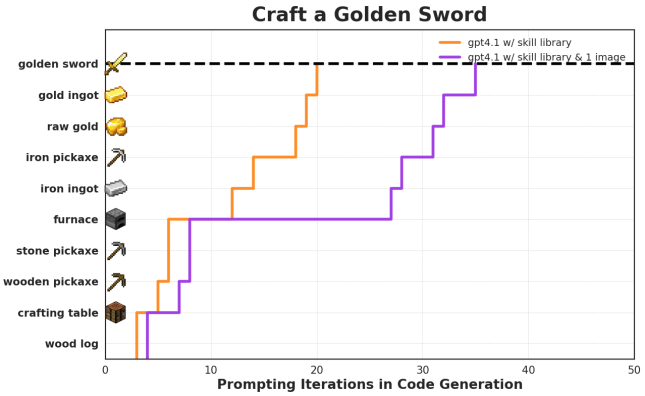

Figure 5. Number of iterations before crafting a Golden Sword

We can observe from Figure 5 that both agents, GPT-4.1 with a skill library (orange) and GPT-4.1 with a skill library plus vision (purple), successfully complete the task of crafting a golden sword. The agent using only the skill library progresses through the required steps more efficiently, completing the task in fewer prompting iterations.

The vision-augmented agent exhibits a noticeable plateau at the furnace stage, caused by a repeated failure to place the furnace correctly. Specifically, the agent places the furnace and immediately destroys it, entering an unproductive loop. However, this issue is not unique to the vision-based agent. Similar behavior has been observed in cases without vision, particularly when placing items like the crafting table, indicating that this is a general execution issue rather than one introduced by visual input.

If this placement bug were resolved, both agents would likely perform similarly on this task. Their progressions follow comparable patterns, and the overall variation in performance would be minimal. This suggests that vision does not inherently degrade performance and may offer benefits in other contexts, but effective integration requires ensuring stability in action execution to avoid regressions caused by low-level control issues.

C. Build a house

Figure 1 presents a qualitative comparison between the house-building outcomes of two Voyager agents: one using text-only inputs (left) and the other enhanced with vision (right). Both agents use GPT-4.1 and were tasked with constructing a house in Minecraft.

The text-only agent produced a structure that meets basic requirements but appears fragmented and spatially inconsistent. The house consists of multiple disjointed sections with irregular placement of windows and materials, suggesting a lack of holistic spatial planning. This outcome reflects the agent's limited environmental grounding, relying solely on symbolic reasoning and textual memory.

In contrast, the vision-augmented agent constructs a more cohesive and realistic structure. The house is well-aligned, symmetrical, and includes consistent window placement, a centered doorway, and walls of uniform height. These improvements suggest that access to visual context allows the agent to better perceive spatial relations, maintain consistency in building dimensions, and correct earlier mistakes.

Overall, these results indicate that vision substantially enhances the agent's ability to reason about structure and spatial coherence in complex construction tasks. While the text-only agent can fulfill task objectives at a functional level, vision enables more deliberate and refined execution, leading to significantly higher quality outputs.

D. Perceived Benefits of Vision

Although quantitative metrics show limited gains from adding vision, our experiments reveal several qualitative improvements. Vision-equipped agents displayed more coherent task execution, for instance, crafting a diamond pickaxe immediately after mining diamonds, indicating a better understanding of item use and task sequencing.

In construction tasks, vision-enhanced spatial reasoning is required. The agent built more consistent and symmetrical structures, benefiting from real-time visual feedback to assess and adjust its progress. Visual input also helped resolve ambiguities that text alone could not address, such as verifying block placement or alignment.

These benefits were achieved without overloading the agent's prompts, as vision was injected selectively into the Curriculum and Critic modules. Overall, vision improved the agent's environmental awareness, planning quality, and behavior coherence, suggesting strong potential for future multimodal integration.

V. Conclusion and Limitations

This project presents a flexible and extensible version of the Voyager agent by integrating open-source language models and visual perception. A key contribution is the use of the Ollama framework, which allows rapid switching between models, enabling accessible and cost-effective experimentation.

We introduced vision into the agent's reasoning pipeline, specifically in the Curriculum and Critic agents, to improve environmental grounding. Visual input supported more structured behaviors, such as executing logical task sequences and building coherent structures. While quantitative improvements were limited, qualitative gains in task consistency and spatial reasoning were notable.

A significant advantage of our approach is its cost-efficiency. Using GPT-4.1 via API, the entire set of experiments cost around $50, equivalent to the cost of just 160 iterations in the original Voyager paper. This highlights the accessibility of our setup for extended experimentation under practical budget constraints.

Despite this, the system faces limitations. Long runs sometimes led to infinite loops, timeouts, or execution failures requiring manual resets. A recurring issue involved item placement, where agents would place and destroy blocks like furnaces repeatedly, seen with and without vision. Visual input also increased prompt complexity, which can challenge smaller open-source models.

Beyond technical outcomes, the project provided valuable learning experiences. It enhanced our understanding of multimodal agent design, prompt engineering, and system integration. Working across vision and language in interactive environments offered deep insight into the practical use of modern AI tools.

In summary, our work advances the development of open, multimodal agents and sets the stage for future improvements in robustness, vision integration, and behavior evaluation.

VI. Individual contributions

This project was a collaborative effort, with both team members contributing to the design, implementation, experimentation, and documentation stages. Together, we explored the integration of visual perception and open-source LLMs within the Voyager framework and tested various configurations to evaluate their impact on agent performance.

Anasse El Boudiri set up the visual data acquisition pipeline from the Mineflayer bot, using Prismarine-Viewer and Puppeteer, and updated key dependencies in the Voyager codebase. He also conducted testing and benchmarking of several open-source LLMs on Minecraft-related tasks.

Jan Steiner integrated the Ollama framework both locally and on the SCITAS cluster, enabling support for open-source LLMs through an OpenAI-compatible API. He worked on model evaluation, tested multiple configurations, and modified the Voyager pipeline for local execution and benchmarking.

Together, we designed the updated pipeline architecture, discussed integration strategies, and ran experiments across different combinations of model types, skill libraries, and vision settings. We also co-authored this report, jointly analyzing the results and refining the project throughout.

REFERENCES

[1] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.16291

[2] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar, "Minedojo: Building open-ended embodied agents with internet-scale knowledge," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. [Online]. Available: https://openreview.net/forum?id=rc8o_j8I8PX

[3] D. Zhu, J. Chen, X. Shen, X. Li, and M. Elhoseiny, "Minigpt-4: Enhancing vision-language understanding with advanced large language models," 2023. [Online]. Available: https://arxiv.org/abs/2304.10592

[4] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2304.08485