APT: Architectural Planning and Text-to-Blueprint Construction Using Large Language Models for Open-World Agents

Jun Yu Chen, Tao Gao

University of California, Los Angeles
Los Angeles, CA, USA

Abstract

We present APT, an advanced Large Language Model (LLM)-driven framework that enables autonomous agents to construct complex and creative structures within the Minecraft environment. Unlike previous approaches that primarily concentrate on skill-based open-world tasks or rely on image-based diffusion models for generating voxel-based structures, our method leverages the intrinsic spatial reasoning capabilities of LLMs. By employing chain-of-thought decomposition along with multimodal inputs (textual and visual), the framework generates detailed architectural layouts and blueprints that the agent can execute under zero-shot or few-shot learning scenarios. Our agent incorporates both memory and reflection modules to facilitate lifelong learning, adaptive refinement, and error correction throughout the building process. To rigorously evaluate the agent's performance in this emerging research area, we introduce a comprehensive benchmark consisting of diverse construction tasks designed to test creativity, spatial reasoning, adherence to in-game rules, and the effective integration of multimodal instructions. Experimental results using various GPT-based LLM backends and agent configurations demonstrate the agent's capacity to accurately interpret extensive instructions involving numerous items, their positions, and orientations. The agent successfully produces complex structures complete with internal functionalities such as Redstone-powered systems. A/B testing indicates that the inclusion of a memory module leads to a significant increase in performance, emphasizing its role in enabling continuous learning and the reuse of accumulated experience. Additionally, the agent's unexpected emergence of scaffolding behavior highlights the potential of future LLM-driven agents to utilize subroutine planning and leverage emergence ability of LLMs to autonomously develop human-like problem-solving techniques.

Code — https://github.com/spearsheep/APT-Architectural-Planning-LLM-Agent

Introduction

In recent years, autonomous agents within the Minecraft environment have become a focal point of research, with methods like reinforcement learning(Baker et al. 2022) and large language models (LLMs) playing a central role(Wang et al. 2024; Fan et al. 2022; Yuan et al. 2023). These methods allow agents to learn by engaging directly with the game world, making real-time decisions, and adapting based on accumulated experiences. The aim is to enable these agents to develop lifelong learning capabilities, improving their skills continuously and handling increasingly complex tasks over time. Yet, while existing research demonstrates potential, it often neglects nuanced creativity and spatial reasoning needed for more sophisticated construction tasks that reflect human-like abilities.

Current research predominantly concentrates on single agents performing straightforward tasks aligned with Minecraft's technology tree, such as tool crafting, mining, and item productions(Wang et al. 2023b; Yu et al. 2024). While these tasks are essential to game mechanics, they follow predictable, task-driven pathways and do not inherently challenge the agent to imagine or create complex building structures. For instance, building houses or farms requires precise block placement in a three-dimensional space with spatial reasoning, which is a far more intricate process than crafting tools or mining resources. Constructing these complex structures involves long-term planning, the ability to envision an architectural blueprint, and a sequential building execution that current agent systems typically lack.

Moreover, while a few projects have integrated LLMs to enhance agent creativity, their generative capacities are limited to the pre-existing Minecraft knowledge embedded in online data. LLMs can replicate known structures but struggle with original designs or untrained configurations, limiting their adaptability in constructing unique or complex architectures, such as villages or multi-room buildings. Thus, despite their generative strengths, LLMs are often unable to build beyond the constraints of their training data and adapt to unfamiliar domains(Ahn et al. 2022), which hinders them from achieving the originality and spatial reasoning required for elaborate construction projects.

Related Work on Structure-Building Agents

Diffusion Model for Creation Although research on structure-building agents is relatively unexplored, one notable study demonstrates an innovative approach by utilizing embodied agents that leverage a diffusion model as an ”imagination” to generate voxel-based images of struc-

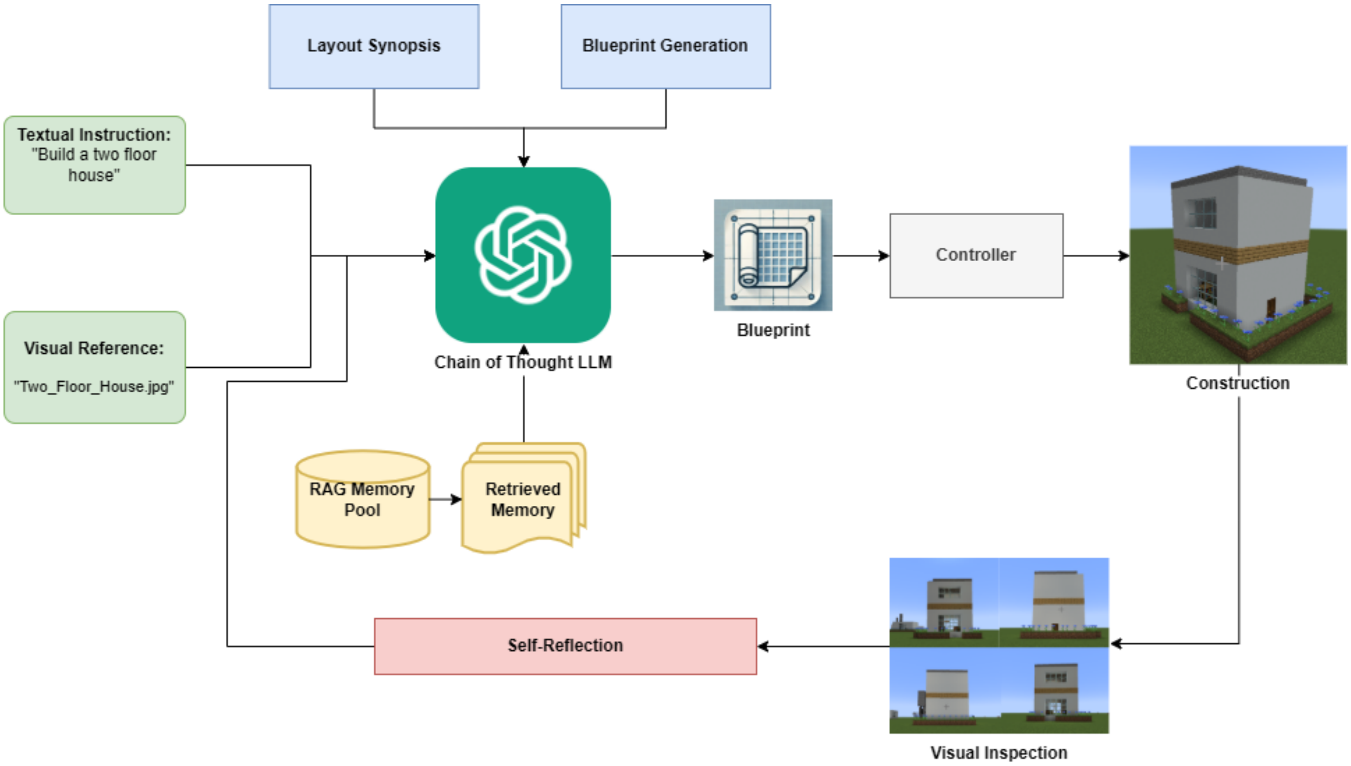


Figure 1: Agent workflow for open-ended construction tasks. The agent begins with either textual instructions, a visual reference, or both. These inputs are processed through a Chain-of-Thought (CoT) module, which first generates a layout synopsis and subsequently produces the blueprint code.Relevant construction memories retrieved from the RAG memory pool are also integrated into the CoT module to enhance planning. The blueprint is then executed by the controller, which utilizes primitive actions to perform the construction within the environment. In cases of unsuccessful executions, the agent employs visual inspection and self-reflection to identify and correct errors, iteratively refining its construction plans in a closed-loop process.

tures(Zhang et al. 2023; Xie et al. 2019). A trained controller then translates these visual outputs into construction actions. This method highlights the potential of integrating generative models in autonomous building tasks. However, it relies heavily on image-based diffusion models(Rombach et al. 2021), where the construction process begins with generating a visual representation of the structure. This dependency, while effective for capturing aesthetic designs, leaves room for exploring alternative approaches.

In particular, leveraging the innate capabilities of Large Language Models (LLMs) to directly map textual instructions to structured blueprints offers a promising direction. Such a direct mapping bypasses the intermediate step of visual generation, potentially streamlining the process and enhancing efficiency in design creation. By utilizing LLMs for zero-shot or few-shot blueprint generation, agents could simplify complex workflows and expand their adaptability across diverse building tasks(Wei et al. 2022a).

Interior Design and Spatial Representation The reliance on a single exterior image for voxel-based blueprint generation introduces challenges in accurately representing internal spatial configurations and item placements. For functional structures requiring precise layouts—such as redstone wiring or detailed item orientations—capturing logical spatial relationships becomes critical. While diffusion models excel in generating stylistic and creative exterior designs, they often struggle to incorporate the fine-grained interior details necessary for replicating realistic architectural functionality. This opens an avenue for approaches that integrate spatial reasoning and logical representation to address both the exterior and interior requirements of complex structures.

Lack of Standardized Benchmarks As the field of autonomous structure-building agents is emerging, a lack of standardized benchmarks poses a challenge for objective performance evaluation. Developing and implementing standardized benchmarks would allow for more consistent skill assessment and comparative analysis across different architectures and methodologies.

Research Objectives

We propose APT, a few-shot learning framework designed to develop LLM-based agents capable of constructing intricate structures in Minecraft by interpreting both textual instructions and visual references. APT emphasizes creativity and accuracy in structure building, aiming to push the boundaries of what AI-driven agents can achieve in this domain. The key objectives of our study are as follows:

1. **Exploring LLM Capabilities for Structure Building:** We aim to leverage advances in recent GPT models to assess their ability for strategic and creative envisioning. Specifically, we will explore how LLMs can translate complex textual instructions directly into blueprint-level details, enabling precise and innovative structure designs.
2. **Incorporating Memory and Reflection Mechanisms:** To foster adaptive and lifelong learning, we plan to implement memory and reflection modules. These mechanisms will allow agents to learn from past building experiences, refine their strategies, and improve performance over time, contributing to more dynamic and intelligent behavior[9,20].
3. **Establishing Comprehensive Benchmarks:** To address the lack of standardized evaluation metrics in this domain, we propose a robust benchmark for structure-building tasks. This benchmark will rigorously evaluate agents across multiple skill areas, including instruction interpretation, creativity, adherence to game rules, and integration of visual inputs.

Through this research, we aim not only to advance the capabilities of autonomous agents in Minecraft but also to contribute to the broader field of creative, task-oriented agent performance assessment by establishing more robust evaluation standards.

APT Multi-modal LLM Agent Architecture

In this section, we introduce the architecture of the APT agent, as illustrated in Figure 1, emphasizing its modular design and workflow. We demonstrate how LLMs leverage spatial reasoning and planning to generate structure blueprints from textual instructions, with enhancements provided by memory and reflection mechanisms. Additionally, Figure 2 showcases example outputs from each component of the workflow, offering a detailed perspective on the agent's operation.

Multimodal Instruction Processing

Our agent leverages the GPT-4o LLM backend to interpret and respond to both textual and visual instructions, allowing for versatile task inputs(OpenAI 2024, 2023). For example, in a task such as building a wooden house, the agent can receive a detailed textual description specifying components and dimensions, a reference image of the house, or both. This multimodal capability enhances the agent's adaptability and accuracy, as it can process comprehensive information in different formats to initiate the building task in Minecraft.

Chain-of-Thought(CoT) Module

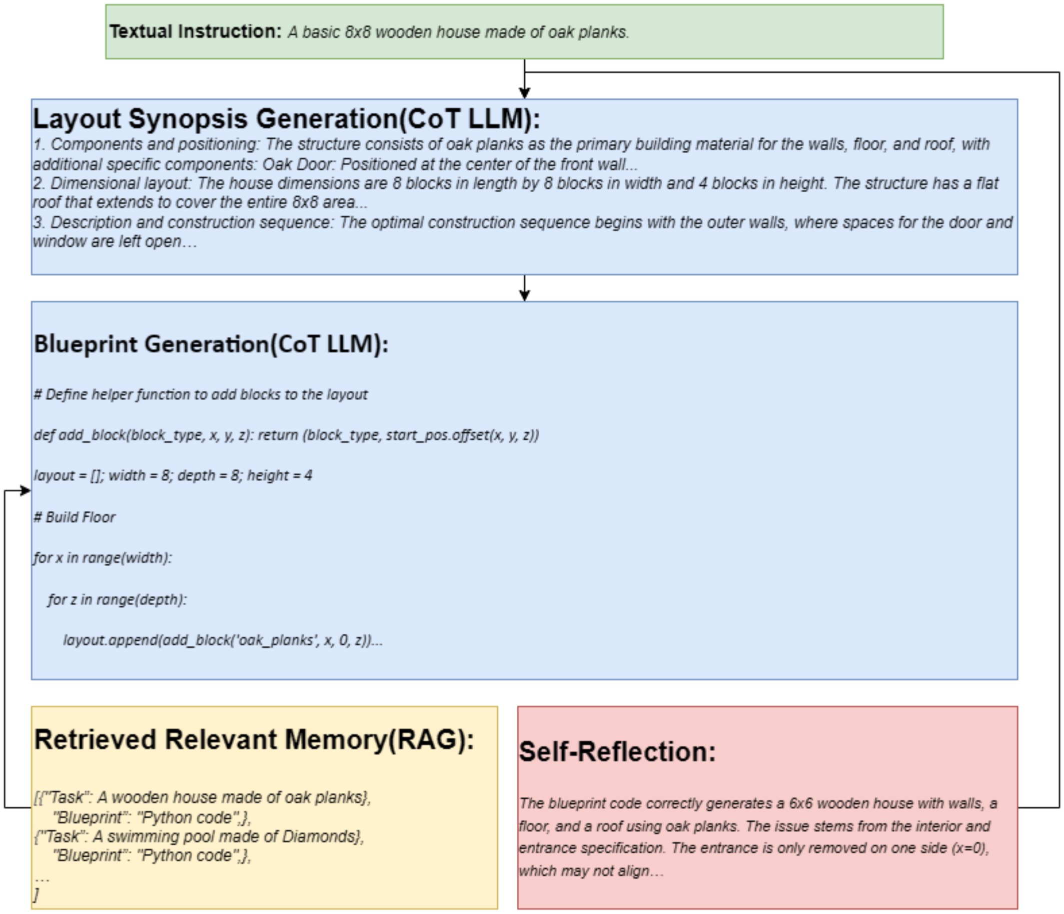
The Chain-of-Thought (CoT) module is central to our agent's architecture, designed to decompose complex tasks into manageable steps for improved reasoning and accuracy (Chen et al. 2023). As shown in Figure 1, the module consists of two primary components: Architecture Layout Synopsis and Blueprint Generation.

Architecture Layout Synopsis We apply prompt engineering in this step to guide the LLM in breaking down the input (image or text) into structured information: components and positioning, dimensional layout, description, and construction sequence, as shown in Figure 2. The components and positioning output identifies items within the structure and their spatial relations, focusing particularly on interior designs. The dimensional layout provides an overall size and shape specification for the structure. The description and construction sequence outlines the structure's purpose and design while detailing a logical order for building—such as completing floors before upper levels and roofs before interior furnishings. This structured approach reduces the chances of execution errors, such as inaccessible areas, and optimizes building efficiency by prioritizing foundational elements first.

Blueprint Generation Following the architecture layout synthesis, the structured information feeds into the second phase of the Chain-of-Thought (CoT) module: blueprint generation. In this step, the agent is prompted to generate Python code that generates a 3D layout blueprint, as depicted in 2. The blueprint is represented as a list of tuples, where each tuple specifies the block type and its exact 3D coordinates within the Minecraft environment. This approach directly engages the spatial reasoning and planning capabilities of the LLM, allowing it to map high-level instructions into detailed spatial and positional representations. By working in a structured text-to-blueprint paradigm, the agent bypasses the intermediate image-generation step, reducing complexity, training requirements, and potential sources of error.

Construction via Primitive Actions The final execution of the blueprint occurs through a sequence of primitive actions facilitated by the Mineflayer library(PrismarineJS 2024), including pathfinding and block manipulation functions adapted from Voyager's primitive action list(Wang et al. 2024). Our approach emphasizes fundamental actions—such as placeBlock, jump, pathfinding, and mineBlock—to construct the structure directly. This contrasts with other agents, such as Jarvis-1, which use MineDojo controllers for higher-level construction management(Fan et al. 2022).

Retrieval-Augmented Planning To enable learning from past tasks, we incorporated a memory pool supported by a Retrieval-Augmented Generation (RAG) system(Brown et al. 2020), using Chroma vector databases in LangChain. This memory architecture allows the agent to retrieve similar past plans when working on new tasks, avoiding repeated errors and enhancing task efficiency. Retrieval queries use cosine similarity search, with the top-k results retrieved as contextual input for blueprint generation, allowing the agent to adapt prior successful strategies to current challenges. As illustrated in Figure 2, multiple past construction memories are stored as key value pairs, with the most relevant one—in this case, a wooden house made of oak planks—being retrieved.


Textual Instruction:
Layout Synopsis Generation(CoT LLM):
Blueprint Generation(CoT LLM):
Retrieved Relevant Memory(RAG):
Self-Reflection:

A basic 8x8 wooden house made of oak planks.

1. Components and positioning: The structure consists of oak planks as the primary building material for the walls, floor, and roof, with additional specific components: Oak Door: Positioned at the center of the front wall...
2. Dimensional layout: The house dimensions are 8 blocks in length by 8 blocks in width and 4 blocks in height. The structure has a flat roof that extends to cover the entire 8x8 area...
3. Description and construction sequence: The optimal construction sequence begins with the outer walls, where spaces for the door and window are left open…

```
# Define helper function to add blocks to the layout

def add_block(block_type, x, y, z): return (block_type, start_pos.offset(x, y, z))

layout = []; width = 8; depth = 8; height = 4

# Build Floor

for x in range(width):

  for z in range(depth):

    layout.append(add_block('oak_planks', x, 0, z))...
```

```
[{"Task": A wooden house made of oak planks},
   "Blueprint": "Python code",},
{"Task": A swimming pool made of Diamonds},
   "Blueprint": "Python code",},
…
]
```

The blueprint code correctly generates a 6x6 wooden house with walls, a floor, and a roof using oak planks. The issue stems from the interior and entrance specification. The entrance is only removed on one side (x=0), which may not align…

Figure 2: Agent Workflow Example: Crafting a Simple Wooden House Task.

Self-Reflection and Error Correction As depicted in Figure 1 and 2, our agent includes an optional self-reflection loop, designed to address execution errors or suboptimal constructions(Shinn et al. 2023). In case of errors, the agent can perform a visual inspection of the structure through multiple first-person screenshots taken from various perspectives, facilitated by Prismarine viewers(PrismarineJS 2024). These images, combined with the original instructions and the execution code, are reprocessed through the CoT module to refine the blueprint and identify specific issues(Wei et al. 2022b). The agent's reflective loop parameter can be adjusted to control the degree of error-checking.

Benchmark Tasks

To rigorously evaluate our agent's construction capabilities in Minecraft, we have developed a benchmark consisting of five structured construction tasks. Each task is crafted to test specific skills, such as spatial reasoning, adherence to game rules, creativity, and accuracy in interpreting instructions(Mandi, Jain, and Song 2024). These tasks range in complexity, from simple builds to intricate designs with functional components, providing a comprehensive assessment of the agent's architectural and creative abilities.

1. **Simple Wooden House**
   - **Description:** A basic 8x8 wooden house equipped with a door, window, bed, and crafting table inside.
   - **Skills Tested:** This task assesses the agent's ability to execute fundamental architectural planning and manage basic interior furnishing by correctly orienting and positioning multiple items within a confined space.
2. **Snow Pyramid**
   - **Description:** A pyramid structure made entirely of snow and ice blocks.
   - **Skills Tested:** This task primarily challenges the agent's creativity by requiring the use of unconventional building materials. Unlike typical pyramid builds using sand, the agent must design a pyramid with snow and ice—materials not traditionally associated with pyramids—without relying on prior construction knowledge or standard block associations.
3. **Village House (from Reference Image)**
   - **Description:** A small but structurally complex village house based on a reference image.
   - **Skills Tested:** This benchmark evaluates the agent's ability to interpret and replicate a design from a vi-

sual reference. The agent must pay attention to fine details, ensuring the final structure closely matches the reference. This task is crucial for assessing the agent's skill in translating visual information into a precise 3D construction, an essential ability for tasks requiring detailed replication of human-created designs.

4. **Watchtower with Redstone Lighting System**
   - **Description:** A tall watchtower featuring a Redstone-powered lighting system at the top, designed to illuminate automatically at night.
   - **Skills Tested:** This task tests the agent's ability to manage vertical space, implement functional Redstone mechanics, and interact dynamically with the environment (day-night cycle). Success here requires a solid understanding of wiring mechanisms and game rules; agents lacking this knowledge will struggle to build a functional lighting system.
5. **Two-Floor Mansion**
   - **Description:** A two-story mansion with intricate interior layouts, including room plans, a flower garden, and a chimney.
   - **Skills Tested:** This advanced task challenges the agent to handle a complex design with multiple floors and detailed interior layouts, demanding precise block placement and long-horizon planning abilities. The agent must interpret a visual blueprint to achieve a cohesive design that maintains functionality and aesthetic appeal across multiple spaces.

Evaluation Framework (VLM Evaluation)

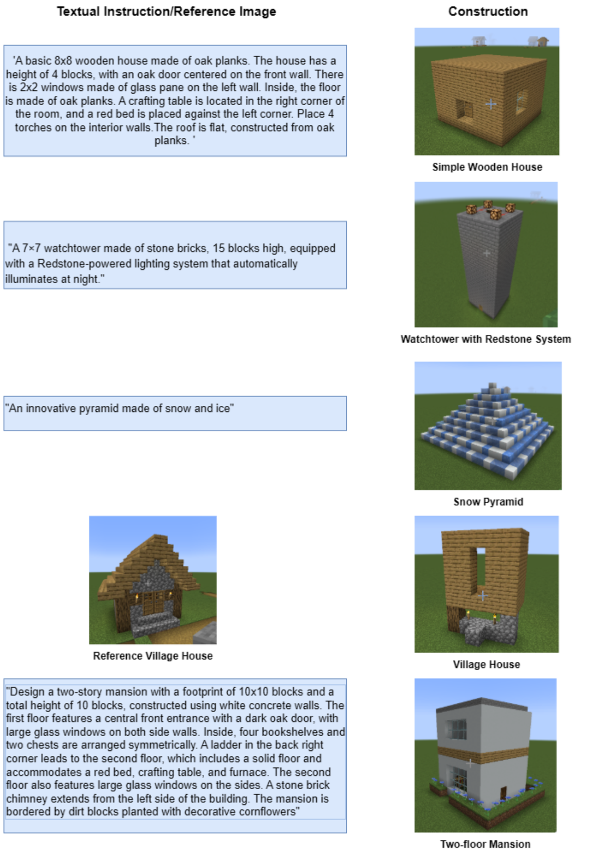
Quantitatively evaluating the aesthetics and accuracy of architectural constructions in Minecraft is a significant challenge, as item placement often requires dynamic weighting, leading to potential inconsistencies and a lack of objective measures for creativity and aesthetics. Traditional evaluation methods rely on human assessment, which, while thorough, is labor-intensive and can be influenced by subjective preferences(Baker et al. 2022). To address this, we leverage advanced Vision-Language Models (VLMs) such as GPT-4o, renowned for their capabilities in vision-language reasoning and interpretation, to automate the evaluation of our agent's constructed architectures(Zhang et al. 2023). A sample evaluation template, showcasing these criteria, is provided below:

Prompt for Evaluation

Your task is to evaluate the building across four key aspects:

1. **Correctness:** How accurately does the building adhere to the provided instructions, accounting for the inclusion of all specified components, block placements, and overall structure shape?
2. **Complexity:** How intricate and detailed is the structure?
3. **Creativity:** How unique and imaginative is the design?
4. **Functionality**: How well does the building serve its intended purpose or function?

Please provide a score (out of 10) for each of these aspects. Additionally, provide an overall total score based on the individual aspect ratings.
Instruction: {`INSTRUCTION`}
Image of the building: {`IMAGE`}



'A basic 8x8 wooden house made of oak planks. The house has a height of 4 blocks, with an oak door centered on the front wall. There is 2x2 windows made of glass pane on the left wall. Inside, the floor is made of oak planks. A crafting table is located in the right corner of the room, and a red bed is placed against the left corner. Place 4 torches on the interior walls.The roof is flat, constructed from oak planks. '

"A 7×7 watchtower made of stone bricks, 15 blocks high, equipped with a Redstone-powered lighting system that automatically illuminates at night."

"An innovative pyramid made of snow and ice"

"Design a two-story mansion with a footprint of 10x10 blocks and a total height of 10 blocks, constructed using white concrete walls. The first floor features a central front entrance with a dark oak door, with large glass windows on both side walls. Inside, four bookshelves and two chests are arranged symmetrically. A ladder in the back right corner leads to the second floor, which includes a solid floor and accommodates a red bed, crafting table, and furnace. The second floor also features large glass windows on the sides. A stone brick chimney extends from the left side of the building. The mansion is bordered by dirt blocks planted with decorative cornflowers"

Figure 3: Benchmark set structures constructed by our APT agent based on provided textual instructions or visual references. Some descriptions are highly detailed and lengthy, specifying precise item placements to rigorously evaluate the LLM's ability to imagine and reason spatially.

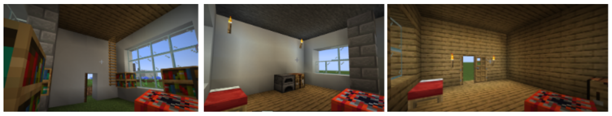
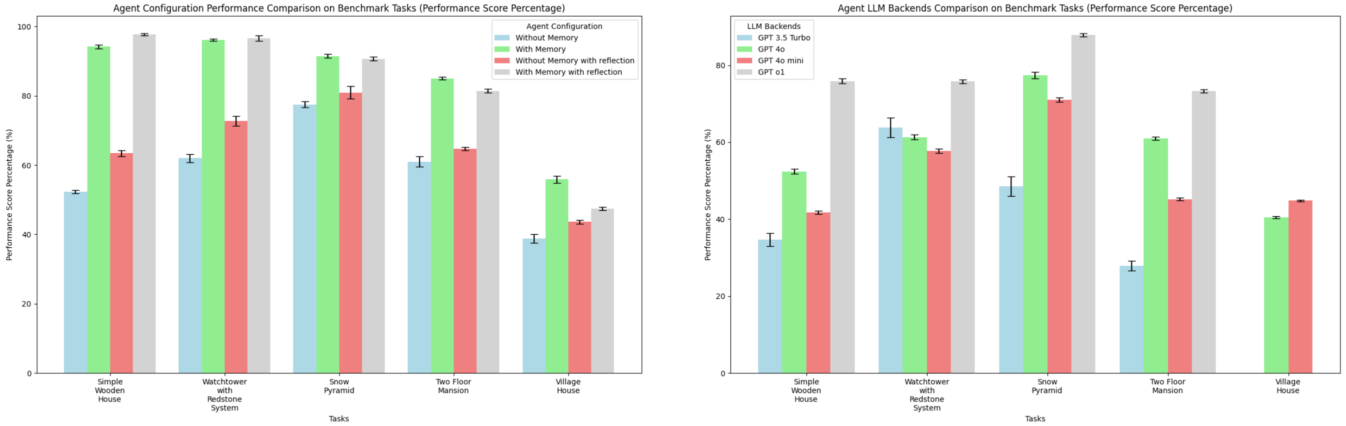


Figure 4: Interior views, from left to right: the first floor of the Two-floor Mansion, the second floor of the same mansion, and the interior room of the Simple Wooden House.



(a) Comparison of performance score percentages of APT agent on benchmark tasks using different configurations, with variations in the use of memory and reflection modules.

(b) Comparison of performance score percentages of APT agent on benchmark tasks using differentLLM backends.

Figure 5: Performance score percentages based on agent configurations (left) and LLM backends (right).

Evaluation Metrics

As shown in the prompt for evaluation, our evaluation framework is based on four primary metrics: Correctness, Creativity, Complexity, and Functionality. Not all metrics apply uniformly to every task, as each construction task is designed to test specific skill sets. For example, the ”Simple Wooden House” task emphasizes the agent’s ability to accurately follow instructions, so it is primarily assessed on Correctness. Evaluating it on Complexity would be unfair, as the structure itself is designed to be straightforward and simple.

Results and Analysis

With the integration of memory and self-reflection modules, the APT agent with GPT-4o backend achieved notable performance scores across most tasks: 97.8% on the Simple Wooden House, 96.7% on the Watchtower with Redstone Lighting System, 90.7% on the Snow Pyramid, and 81.4% on the Two-Floor Mansion. As depicted in Figure 3 and Figure 4, the agent’s creations for these four tasks demonstrate a commendable level of adherence to the provided instructions. However, performance on the Village House with Reference Image task was significantly lower, with an average score of 47.5%. As depicted in Figure 5a, adding memory and reflection modules yielded only minimal improvements for this task.

As shown in Figure 3, the construction of ”Village House” by our APT agent replicates only the individual items depicted in the reference, including the use of torches, cobblestone, oak wood, and oak planks. However, it fails to capture the three-dimensional structure of the reference image. This task required the agent to interpret a reference image accurately, suggesting that visual fidelity and detailed reproduction remain challenging for LLM-based agents, especially in zero-shot or few-shot scenarios. This indicates that reasoning from images to descriptions and then to blueprints is still a complex hurdle for LLMs, regardless of the configuration used.

Ablation Experiment on Agent with Memory and Self-reflection

We conducted ablation experiments to systematically compare various agent configurations (with and without memory and reflection) using a GPT-4o backend, assessing the individual and combined contributions of each module to overall performance in Figure 5a. The results demonstrate that the memory module significantly enhances the APT agent’s performance, with an average increase of 47.3% in performance scores when memory is enabled. Even with the addition of the reflection module, the APT agent’s performance still saw a substantial boost, with a combined increase of 26.24%. In contrast, without memory, the implementation of reflection alone improved performance by 12.8%, suggesting that while reflection yields moderate gains(Huang et al. 2024), memory remains the more influential factor in enhancing agent capabilities.

To evaluate robustness, we calculated the standard deviation of overall scores across all test tasks, each assessed over 10 trials. Standard deviation here reflects consistency, with larger values indicating less robust performance. Notably, it is evident from Figure 5a that configurations lacking memory (either with or without reflection) exhibited higher standard deviation, underscoring the variability of LLM-generated plans and reasoning paths even with the temperature set to zero, which was intended to produce more predictable outputs. In contrast, implementing RAG with memory reduced variability, enabling the model to generate more stable and deterministic plans.

Agent Workflow with Different LLM Backend

As depicted in Figure 5b, We evaluated the APT agent’s performance across five construction tasks using various LLM backends: GPT-3.5 Turbo, GPT-4o, GPT-4o Mini, and

GPT-o1. Each backend demonstrated unique capabilities, with significant performance differences across tasks. Notably, GPT-o1 achieved the highest overall average score of 78.22%, compared to GPT-3.5 Turbo at 43.75%, GPT-4o at 63.02%, and GPT-4o Mini at 53.94%. This outcome suggests that GPT-o1 exhibits superior reasoning abilities, particularly in handling complex spatial structures and accurately interpreting detailed textual instructions.

In Figure 5b, we observed that GPT-3.5 Turbo struggled more with tasks such as the ”Simple Wooden House” and ”Two-Floor Mansion,” both of which required understanding intricate spatial configurations and processing extended textual instructions. Moreover, because GPT-3.5 Turbo and GPT-o1 lack multimodal capabilities, the “Village House” task was only tested on GPT-4o and GPT-4o Mini. Both models performed poorly on this visually-driven task, underscoring current limitations in LLMs for tasks that demand high visual fidelity and nuanced spatial reasoning.

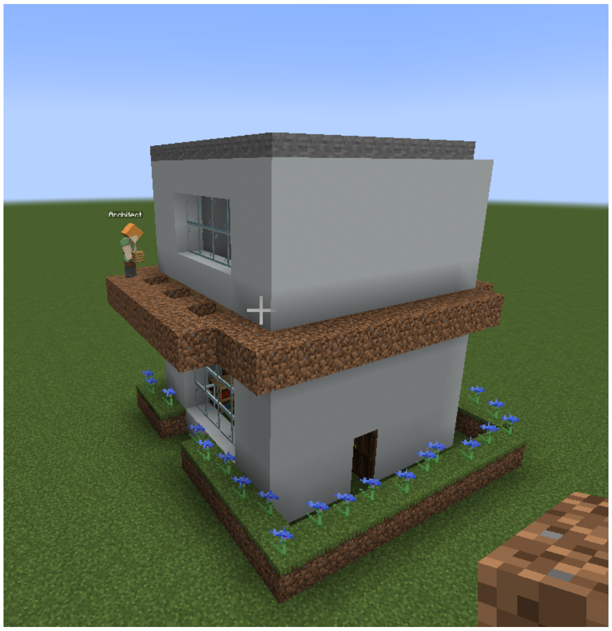


Figure 6: Our APT agent demonstrates emergent abilities in constructing scaffolding as a part of the blueprint execution plan.

Scaffolding ability

One unexpected finding in agent behavior was that some blueprint layout code generated by our chain-of-thought modules included scaffolding as part of the construction sequence, as shown in Figure 6. In Minecraft, where agents are restricted from flying, scaffolding (e.g., stacking dirt or temporary blocks) is commonly used by players to reach higher floors and build efficiently. As observed in This behavior in our APT agent emerged without any explicit prompt for scaffolding, suggesting that the LLM inferred this technique while ”imagining” an optimal construction sequence. This phenomenon is intriguing as it highlights the LLM’s potential capability for implicit reasoning and common sense knowledge—recognizing and incorporating practical construction strategies from inferred context rather than direct instructions(Zhao, Lee, and Hsu 2023).

Correlation Type	Coefficient
Pearson	0.988
Spearman	0.903

Table 1: Correlation coefficients between Human and Machine evaluation. Pearson’s coefficient measures strength of linear correlation, while Spearman’s coefficient measures monotonic correlation based on ranks.

Consistency between Human Evaluation and AI Evaluation

To ensure that the evaluation of the VLM aligns closely with human judgment, we invited 22 participants to manually assess the structures created by our agents. Recognizing the critical influence of the memory component on the agent’s performance, we focused the evaluation on structures generated by agents using the GPT-4o backend, comparing outputs both with and without memory. The reflection component was excluded from this assessment, as the memory component plays a more substantial role in shaping the overall performance of the agent.

To evaluate the consistency and alignment between the VLM evaluation and human judgment, we utilized the Pearson correlation coefficient and Spearman correlation coefficient, which are standard metrics for assessing the strength and nature of relationships between two sets of data (Schober, Boer, and Schwarte 2018). The results in Table 1 show a Pearson correlation coefficient of 0.988 and a Spearman correlation coefficient of 0.903, indicating strong alignment and reliability between human assessments and the VLM evaluation. These findings validate the VLM evaluation process as a dependable alternative to human judgment for assessing agent-generated structures.

Limitations and Future Directions

One limitation of our study is the constrained memory capacity of the agent. Expanding the memory pool with a broader and more diverse set of building experiences could improve retrieval efficiency and boost the agent’s performance and success rates in constructing complex structures(Wang et al. 2023b). To address data scarcity, text-based prompt generators could be utilized to produce creative and detailed building instructions, enriching the memory pool with varied examples. Furthermore, integrating image-to-3D modeling tools, such as Tripo AI, could allow the memory to store voxel-based blueprints, enabling the agent to reason spatially and replicate intricate structures from visual inputs.

Another challenge stems from the limited range of primitive actions implemented in the current framework. This restriction prevents the agent from replicating more complex player behaviors, such as activating blocks, pouring liquids, or handling intricate mechanisms. Enhancing the primitive action set would enable the agent to undertake a broader range of construction tasks, such as the creation of functional semi-automated farms. Integrating downstream text-to-behavior controllers and policy execution frameworks, such as STEVE-1 or Voyager(Lifshitz et al. 2023; Wang et al. 2024), could further optimize task execution and expand the agent's behavioral repertoire.

Lastly, our findings indicate that the LLM-driven agent struggles to reason directly from visual references to produce accurate descriptive instructions and map these into precise blueprint layouts. This limitation highlights an opportunity for improvement through fine-tuning LLMs with training datasets that pair visual references with generated blueprints(Wei et al. 2022a). Such advancements could significantly enhance the agent's ability to perform spatial reasoning and visualization tasks, capabilities not innately present in current LLMs but vital for more complex applications.

Conclusion

In this paper, we propose APT, an LLM-driven agent framework capable of constructing complex structures by leveraging the inherent reasoning abilities of large language models (LLMs) in both textual and spatial contexts. This framework is the first of its kind to integrate memory and reflection components into structure-building agents. While previous work has primarily focused on skill acquisition and task progression within the technology tree, the domain of structure building—a behavior highly characteristic of real player interactions in Minecraft—remains relatively unexplored. Our APT agent is capable of following extensive descriptive instructions to construct structures with detailed internal designs and arranged item layouts. Our findings demonstrate that the memory module significantly enhances the performance of few-shot and zero-shot learning agents. Conversely, the reflection module has shown limited impact. However, further advancements in cognitive framework for reflection(Wang et al. 2023a) and the incorporation of enhanced computer vision capabilities could improve its effectiveness(Shinn et al. 2023; Yao et al. 2023).

Additionally, we created a benchmarking dataset to evaluate agents' abilities across creativity and spatial reasoning tasks. This benchmark provides a valuable tool for researchers to test diverse skill sets within this domain. Lastly, the agent's unexpected use of scaffolding—a technique widely employed in both real-world construction and survival-mode Minecraft gameplay—raises questions about the boundaries of emergent reasoning in LLMs and their ability to generalize human-like problem-solving techniques. Future research may also consider scaffolding modules for subroutine planning, particularly in advancing the construction of more complex and sophisticated structures.

References

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Gopalakrishnan, K.; Hausman, K.; Herzog, A.; Ho, D.; Hsu, J.; Ibarz, J.; Ichter, B.; Irpan, A.; Jang, E.; Ruano, R. M. J.; Jeffrey, K.; Jesmonth, S.; Joshi, N. J.; Julian, R. C.; Kalashnikov, D.; Kuang, Y.; Lee, K.-H.; Levine, S.; Lu, Y.; Luu, L.; Parada, C.; Pastor, P.; Quiambao, J.; Rao, K.; Rettinghouse, J.; Reyes, D. M.; Sermanet, P.; Sievers, N.; Tan, C.; Toshev, A.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Xu, S.; and Yan, M. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning*.

Baker, B.; Akkaya, I.; Zhokov, 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. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *Advances in Neural Information Processing Systems*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Chen, Z.; Zhou, K.; Zhang, B.; Gong, Z.; Zhao, X.; and Wen, J.-R. 2023. ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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*.

Huang, J.; Chen, X.; Mishra, S.; Zheng, H. S.; Yu, A. W.; Song, X.; and Zhou, D. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

Lifshitz, S.; Paster, K.; Chan, H.; Ba, J.; and McIlraith, S. 2023. STEVE-1: A Generative Model for Text-to-Behavior in Minecraft. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Mandi, Z.; Jain, S.; and Song, S. 2024. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 286–299.

OpenAI. 2023. GPT-4 Technical Report. Technical report, OpenAI. Accessed: 2024-11-18.

OpenAI. 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/. Accessed: 2024-11-10.

PrismarineJS. 2024. PrismarineJS/mineflayer. https://github.com/PrismarineJS/mineflayer. Accessed: 2024-04-15.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2021. High-Resolution Image Synthesis with Latent Diffusion Models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10674–10685.

Schober, P.; Boer, C.; and Schwarte, L. A. 2018. Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia*, 126(5): 1763–1768.

Shinn, N.; Cassano, F.; Berman, E.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*.

Wang, Z.; Cai, S.; Chen, G.; Liu, A.; Ma, X.; and Liang, Y. 2023a. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wang, Z.; Cai, S.; Liu, A.; Ma, X.; and Liang, Y. 2023b. JARVIS-1: Open-world Multi-task Agents with Memory-Augmented Multimodal Language Models. In *Second Agent Learning in Open-Endedness Workshop*.

Wei, J.; Bosma, M.; Zhao, V.; Guu, K.; Yu, A. W.; Lester, B.; Du, N.; Dai, A. M.; and Le, Q. V. 2022a. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; brian ichter; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022b. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *Advances in Neural Information Processing Systems*.

Xie, H.; Yao, H.; Sun, X.; Zhou, S.; and Zhang, S. 2019. Pix2Vox: Context-Aware 3D Reconstruction From Single and Multi-View Images. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2690–2698.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Yu, X.; Fu, J.; Deng, R.; and Han, W. 2024. MineLand: Simulating Large-Scale Multi-Agent Interactions with Limited Multimodal Senses and Physical Needs. arXiv:2403.19267.

Yuan, H.; Zhang, C.; Wang, H.; Xie, F.; Cai, P.; Dong, H.; and Lu, Z. 2023. Skill Reinforcement Learning and Planning for Open-World Long-Horizon Tasks. arXiv:2303.16563.

Zhang, C.; Cai, P.; Fu, Y.; Yuan, H.; and Lu, Z. 2023. Creative Agents: Empowering Agents with Imagination for Creative Tasks. arXiv:2312.02519.

Zhao, Z.; Lee, W. S.; and Hsu, D. 2023. Large Language Models as Commonsense Knowledge for Large-Scale Task Planning. In *RSS 2023 Workshop on Learning for Task and Motion Planning*.

Appendix

This section provides the implementation details of our Chain-of-Thought (CoT) LLM framework and detailed examples of the prompts.

Overview

To fully utilize the potential of GPT-based LLMs for imagining and planning blueprints with spatial reasoning and precise item placements, we employed GPT-4o, GPT-4o Mini, GPT-3.5 Turbo, and GPT-O1. Through prompt engineering and a Chain-of-Thought (CoT) framework, we aimed to enhance the agent's ability to generate structured plans and execute them effectively.

Layout Synopsis Generation Module

Layout Synopsis Prompt

Please translate the structure in the provided text with the following Minecraft details:

1. **Components and Positioning:** List all individual elements (e.g., blocks, materials, windows, doors, etc.) used in the structure and describe the position of each component relative to the entire structure.
2. **Dimensional Layout:** Provide the overall dimensions of the structure (length, width, height).
3. **Description:** Summarize the purpose and design of the structure (e.g., a house, tower, etc.), and outline the most logical construction sequence, taking into account how building certain parts first could obstruct access to other areas.

Please ensure the description is clear, precise, and professional, making it easy to recreate the structure programmatically.
Here is the provided text description: `{text}`

Blueprint Generation Module

Following the Layout Synopsis, the Blueprint Generation Module generates executable Python code to construct the layout within the Minecraft environment. This stage integrates a highly structured prompt to guide the LLM in producing code efficiently and dynamically.

Blueprint System Prompt

You are an expert in both Minecraft and Python coding. Your task is to generate Python code that creates layouts for Minecraft structures as a list of tuples. The structure layout should be represented in the following way:

- Each tuple contains:
  1. The block type (e.g., `'oak_planks'`, `'glass_pane'`, `'oak_door'`).
  2. The exact 3D position of the block, repre-

sented by a `vec3` object with `x`, `y`, and `z` coordinates.

The layout should follow this format:

```
[
    ('block_type', start_pos.offset
        (x, y, z)),
    ('block_type', start_pos.offset
        (x, y, z)),
    ('block_type', start_pos.offset
        (x, y, z))
]
```

Important Notes:

- You do not need to manually define every block's coordinates. Instead, provide efficient and reusable code that generates the layout dynamically, based on the starting position.
- The variable storing the layout must be named `layout`.
- Do not append `'air'` to the layout.

# Always start the code with this:

```
start_pos = self.bot.entity.
    position.floor()
```

# Layout generation code that returns the layout
...
# The output always ends with the layout being passed into the following method:

```
self.actions.buildStructure(layout,
     mode='creative')
```

For the user's input, we provide the retrieved plan from our RAG memory pool with the highest similarity score, along with the layout synopsis from the first module.

Blueprint User Prompt

Here is a possibly relevant plan from past experiences that were effective under their contexts (could be None). If the task described isn't relevant, don't reference it. If it is highly relevant, then reference it: retrievedPlans
Please provide the code for this structure: structure.

```
1. Do not import mineflayer or
    vec3
2. Do not miss any components
Ensure the generated code is
    properly indented and
    formatted as a complete
    Python script.
```

Only include the code and output the code into a compact JSON format on a single line without whitespace. The key is 'code' and the value is the actual code.

Reflections Module

Once the agent completes the visual inspection of the constructed structure, the captured screenshots are seamlessly integrated into the self-reflection chain. This process combines the screenshots with the original structure description and the previously failed code. The prompt in the self-reflection chain generates not only the corrected code but also a comprehensive analysis of what was incorrect or suboptimal in the initial code. This analysis includes detailed insights into the generated structure, identification of potential errors, and recommendations for improvement, ensuring iterative enhancement and accuracy in subsequent builds.

Self Reflection Prompt

You are an expert in both Minecraft structure generation and Python coding. Below is the current imperfect blueprint code used to generate a Minecraft structure, along with a description of the intended structure and an image of what was generated by the current code.
Structure Description: structure
Image: Image
Current Blueprint Code: blueprint
Task: Generate the following features:

1. **Reflection:** Analyze why the current blueprint code does not successfully generate the structure as described. Compare the structure description with both the image and the code itself. Issues may arise either from discrepancies in the visual appearance of the generated structure compared to the description, or from errors in the code's syntax that prevent it from running. Keep this concise.
2. **Code:** Provide an improved, optimized version of the blueprint code that accurately aligns with the structure description and resolves any issues in the current code.

```
# Always start the code with this:
start_pos = self.bot.entity.
    position.floor()
# layout generation code that
    returns the layout
# ...
# The output always ends with the
    layout being passed into the
    following method
self.actions.buildStructure(layout,
     mode='creative')
```

Ensure the generated code is properly indented and formatted as a complete Python script. Include the code and reasoning into a compact JSON format on a single line without whitespace where we have two keys "reflection" and "code" and the value is the corresponding output.