Metagent-P: A Neuro-Symbolic Planning Agent with Metacognition for Open Worlds

Yanfang Zhou[1*] **Yuntao Liu**[1*] **Xiaodong Li**[2*] **Yongqiang Zhao**[3] **Xintong Wang**[4]
Jinlong Tian[2] **Zhenyu Li**[1] **Xinhai Xu**[1†]
[1]Academy of Military Sciences
[2]National University of Defense Technology
[3]Peking University
[4]Department of Informatics, Universität Hamburg
yanfangzhou08@gmail.com, xuxinhai@nudt.edu.cn

Abstract

The challenge of developing agents capable of open-world planning remains fundamental to artificial general intelligence (AGI). While large language models (LLMs) have made progress with their vast world knowledge, their limitations in perception, memory, and reliable reasoning still hinder LLM-based agents from achieving human-level performance in long-term tasks. Drawing inspiration from human cognitive-metacognitive collaboration, we propose **Metagent-P**, integrating the world knowledge of LLMs, the symbolic reasoning capabilities of cognitive architectures, and the self-reflection characteristic of metacognition to construct a "planning-verification-execution-reflection" framework. Metagent-P improves experience utilization through multimodal memory integration. It uses a neural-symbolic hierarchical representation structure to ensure the plan's reasoning correctness in advance. Finally, it actively adapts the agent to dynamic environments through monitoring, evaluation, and regulation mechanisms. Experimental results show Metagent-P significantly outperforms current state-of-the-art methods in Minecraft. In long-term tasks, Metagent-P reduces the average replanning counts by **34%** and exceeds the average human success rate by **18.96%**. Additionally, Metagent-P also demonstrates self-evolution through step-by-step open-world exploration.

1 Introduction

Planning (Russell and Norvig, 2016) in open-world environments remains a key challenge in artificial intelligence. Recent advances in Large Language Models (LLMs) (Achiam et al., 2023; Guo et al., 2024) show promising potential through their world knowledge and language processing capabilities in open-world planning (BAAI, 2023; Wang et al., 2023, 2024; Qin et al., 2024). However, LLMs struggle with long-term tasks due to limited perception, memory, and reliable reasoning (Venkit et al., 2022; Tang et al., 2023), failing to match human-like planning and adaptation.

Human cognition theory suggests a solution through dual cognitive-metacognitive mechanisms. The cognitive domain (Laird et al., 2017) primarily handles task decomposition and action planning, while the metacognition domain (Flavell, 1979) monitors and optimizes these processes, enabling adaptive planning in changing environments.

Cognitive Architectures (CAs) (Anderson, 2009; Laird, 2019) simulate human cognition through semantic memory (general knowledge), episodic memory (past experiences), and procedural memory (skills knowledge stored as condition-action rules). Procedural memory drives CAs' core symbolic reasoning by matching rules with current states and inferring actions to generate reliable plans (Mininger and Laird, 2022). However, their dependence on pre-encoded domain knowledge limits knowledge acquisition and generalization in open-world environments (Lieto et al., 2018). Moreover, with increasing task complexity and environmental uncertainty, basic cognitive capabilities become insufficient (Cox, 2005). Human higher-order metacognition (Flavell, 1979) can address this by monitoring, evaluating, and regulating cognitive processes, enhancing adaptability by detecting failures, and adjusting strategies for more flexible performance in uncertain environments.

Based on our analysis (Table 1), LLM-based agents face two key challenges:

1. Insufficient Cognition. Existing agents focus on semantic reasoning but neglect symbolic reasoning based on procedural rules, leading to reasoning errors in long-term planning.

2. Insufficient Metacognition. Existing agents directly employing LLMs alone cannot accurately identify planning deviations in the dynamic envi-

[*] Equal contribution.
[†] Corresponding author.

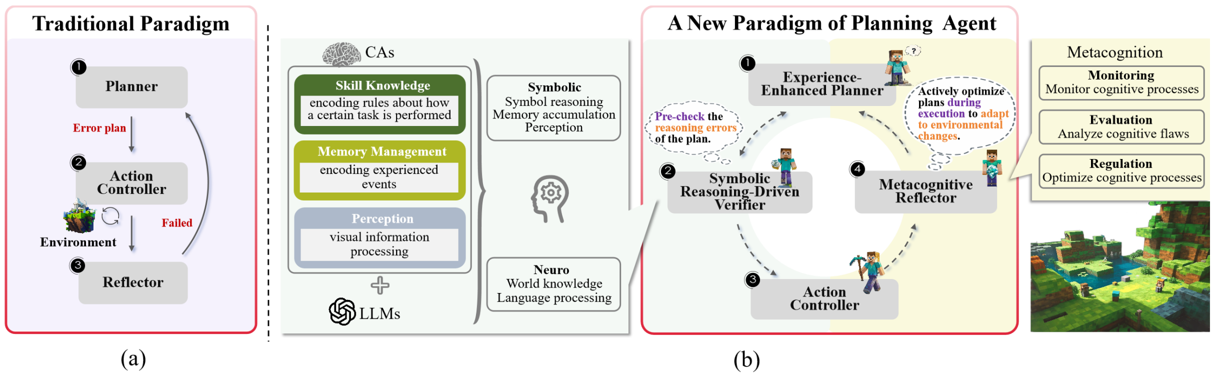

Traditional Paradigm
Planner
Error plan
Action Controller
Environment
Failed
Reflector
(a)
CAs
Skill Knowledge
encoding rules about how a certain task is performed
Memory Management
encoding experienced events
Perception
visual information processing
LLMs
Symbolic
Symbol reasoning
Memory accumulation
Perception
Neuro
World knowledge
Language processing
A New Paradigm of Planning Agent
Experience-Enhanced Planner
Pre-check the reasoning errors of the plan.
Actively optimize plans during execution to adapt to environmental changes.
Symbolic Reasoning-Driven Verifier
Metacognitive Reflector
Action Controller
Metacognition
Monitoring
Monitor cognitive processes
Evaluation
Analyze cognitive flaws
Regulation
Optimize cognitive processes
(b)

Figure 1: **Comparison of Metagent-P and Traditional LLM-based Agent.** **(a)** Traditional methods use a *"planning-execution-reflection"* framework. **(b)** Inspired by cognitive theory, Metagent-P constructs a *"planning-verification-execution-reflection"* framework. It combines CAs' strengths in symbolic reasoning, perception, and experience with LLMs' world knowledge, plus metacognitive abilities, to achieve human-like planning.

ronments or provide reliable corrective feedback. Consequently, they cannot do adaptive planning like humans do.

Feature	CAs	LLMs	Human
Symbolic reasoning	++	-+	++
Perception	++	-	++
Memory accumulation	++	-	++
Language processing	-+	++	-+
World knowledge	-+	++	-+
Self-reflection (metacognition)	-	-+	++

Table 1: ++ Fully supported, (-+) Partially supported, (-) Rarely (or not) supported.

To address these limitations, we propose Metagent-P, a neuro-symbolic planning agent with metacognition, consisting of four components: planner, verifier, controller, and reflector. As shown in Figure 1, it combines the world knowledge of LLMs, the symbolic reasoning capabilities of CAs, and the self-reflection characteristic of metacognition, creating a "planning-verification-execution-reflection" framework for open-world tasks.

To improve the agent's situation awareness and experience utilization, we propose an **Experience-Enhanced Planner** with multimodal memory that encodes both successful and failed cases. Successful cases guide task decomposition, while failed cases inform the reflector's decision-making.

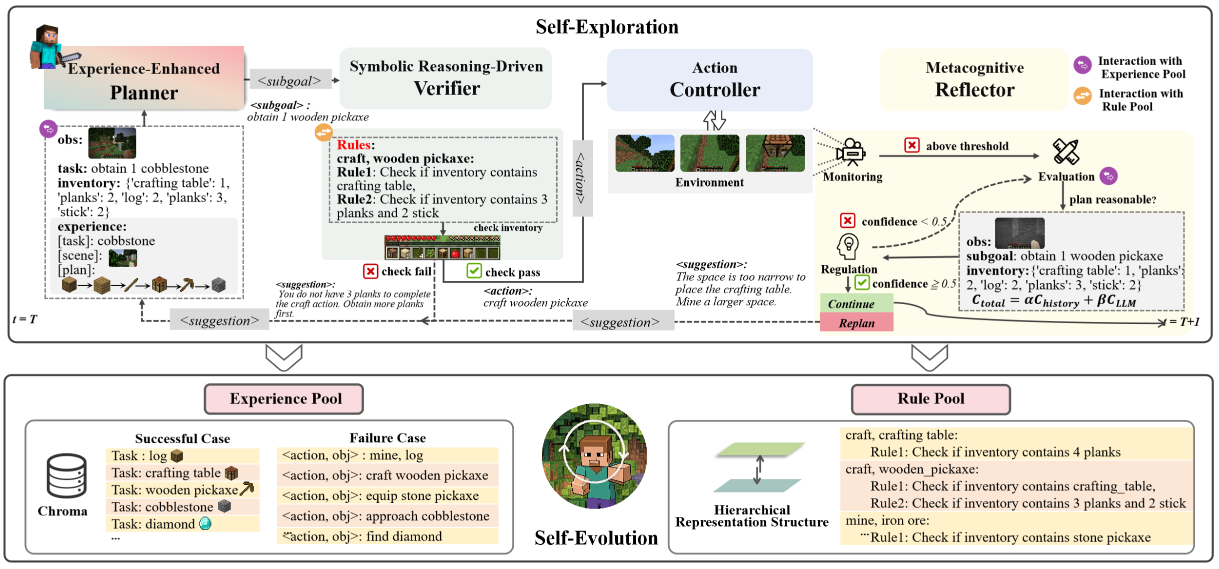
To preemptively detect reasoning errors and improve efficiency, we design a **Symbol-Reasoning Driven Verifier** with a neuro-symbolic knowledge hierarchical representation structure. The top layer encodes explicit rules for reliable reasoning, while the bottom layer embeds LLMs for implicit world knowledge. Through novel knowledge representation, it enhances verifier reasoning. The verifier achieves autonomous rule learning via bidirectional top-down guidance and bottom-up learning, without any manual rule definition.

We design a **Metacognitive Reflector** inspired by human metacognition that enhances the agent's adaptability through three steps: (1) Monitoring: Detects execution anomalies and dynamically triggers evaluation. (2) Evaluation: Similar to human self-examination, it evaluates plan rationality with quantitative scoring, by considering current observation and past experience. (3) Regulation: Optimizes plans based on evaluation feedback to enable agent adaptation to dynamic environments.

We evaluate Metagent-P in Minecraft, a popular open-world game environment, showing significantly outperforms current state-of-the-art methods. In long-term tasks, it has a **2 to 25 times** performance improvement, reduces the average re-planning counts by **34%**, and exceeds the human success rate by **18.96%**. Through step-by-step exploration, Metagent-P demonstrates self-evolution by actively adapting to dynamic environments.

The main contributions of this paper are:

- Metagent-P integrates LLMs' world knowledge, CAs' symbolic reasoning, and metacognition's self-reflection to enable human-like open-world planning.
- **Symbolic Reasoning-Driven Verifier** uses a neural-symbolic structure to enhance knowledge representation and reasoning, enabling autonomous rule learning. It detects reasoning errors early and boosts planning efficiency.
- **Metacognitive Reflector** enables human-like actively adaptive planning via quantitative



Self-Exploration
Experience-Enhanced Planner
<subgoal>
Symbolic Reasoning-Driven Verifier
Action Controller
Metacognitive Reflector
Interaction with Experience Pool
Interaction with Rule Pool
<subgoal> : obtain 1 wooden pickaxe
obs:
task: obtain 1 cobblestone
inventory: {'crafting table': 1, 'planks': 2, 'log': 2, 'planks': 3, 'stick': 2}
experience:
[task]: cobbstone
[scene]:
[plan]:
Rules:
craft, wooden pickaxe:
Rule1: Check if inventory contains crafting table,
Rule2: Check if inventory contains 3 planks and 2 stick
check inventory
check fail
check pass
<action>
<action>: craft wooden pickaxe
<suggestion>: You do not have 3 planks to complete the craft action. Obtain more planks first.
Environment
Monitoring
above threshold
Evaluation
plan reasonable?
confidence < 0.5
Regulation
confidence ≧ 0.5
Continue
Replan
obs:
subgoal: obtain 1 wooden pickaxe
inventory: {'crafting table': 1, 'planks': 2, 'log': 2, 'planks': 3, 'stick': 2}
$C_{total} = \alpha C_{history} + \beta C_{LLM}$
<suggestion>: The space is too narrow to place the crafting table. Mine a larger space.
<suggestion>
<suggestion>
t = T
t = T+1
Experience Pool
Chroma
Successful Case
Task : log
Task: crafting table
Task: wooden pickaxe
Task: cobblestone
Task: diamond
...
Failure Case
<action, obj> : mine, log
<action, obj>: craft wooden pickaxe
<action, obj>: equip stone pickaxe
<action, obj>: approach cobblestone
<action, obj>: find diamond
Self-Evolution
Rule Pool
Hierarchical Representation Structure
craft, crafting table:
Rule1: Check if inventory contains 4 planks
craft, wooden_pickaxe:
Rule1: Check if inventory contains crafting_table,
Rule2: Check if inventory contains 3 planks and 2 stick
mine, iron ore:
...Rule1: Check if inventory contains stone pickaxe

Figure 2: **Overview of the Metagent-P framework.** Metagent-P consists of four components: *Planner*, *Verifier*, *Controller*, and *Reflector*. Given the task "obtain 1 cobblestone", the *Planner* decomposes the task into actions using multimodal memory. The *Verifier* checks the feasibility of actions, and the *Controller* executes them or sends them back for refinement. The *Reflector* monitors, evaluates and adapts plans based on observations and experience. Through exploration, Metagent-P enriches its rule and experience pools for self-evolution.

evaluation, facilitating agent self-evolution.

2 Method

In this section, we first give an overview of our Metagent-P. Next, we present detailed discussions of Metagent-P's key components: the Experience-Enhanced Planner, the Symbolic Reasoning-Driven Verifier, and the Metacognitive Reflector.

2.1 Overview

To develop an agent to complete long-term tasks in open-world environments like humans, we design the Metagent-P. As shown in Figure 2, our Metagent-P includes four major components. **The Experience-Enhanced Planner** breaks the high-level task into executable action sequences. **The Symbolic Reasoning Verifier** checks the plan's reasoning errors before task execution. **The Action Controller** converts each action in the sequence into executable commands, enabling the agent to interact with the environment (In this paper, we employ the Controller proposed by MP5 (Qin et al., 2024). Details of Controller can be found in Apendix C.2). **The Metacognitive Reflector** enables the agent to actively adapt the plan to environmental changes.

Why can Metagent-P solve long-term tasks? Metagent-P achieves long-term tasks mainly through its three interacting components. The Experience-Enhanced Planner leverages multimodal memory for situation-aware task decomposition and action generation. The Symbolic Reasoning Verifier pre-checks action feasibility to ensure correct reasoning and reliable planning. The Metacognitive Reflector triggers upon monitoring anomalies, quantitatively evaluates plans by integrating observation and experiences, and provides reliable feedback for plan refinement before failure occurs. This enables Metagent-P's adaptive planning capability for long-term task completion.

How does Metagent-P achieve self-evolution? Metagent-P achieves self-evolution through cognitive shaping and adaptive optimization (For details, see Section 3.4.) During cognitive shaping, it abstracts implicit knowledge into symbolic rules and records successful and failed cases to enrich its rule and experience pools, continuously enhancing its planning capabilities through step-by-step exploration. In adaptive optimization, the Reflector's active plan adaptation handles environmental disturbances, ensuring stable planning performance in uncertain environments.

2.2 Experience-Enhanced Planner

Inspired by the memory management and perception modules of CAs, Metagent-P employs LLMs to design a multimodal experience-enhanced planner, providing richer references for in-context learning. The formulation is as follows:

$$\begin{aligned} T = \{g_1, g_2, \ldots, g_n\} &= \mathit{decompose}(E, T, h; \Theta, \mathcal{P}), \\ \{a_{i1}, a_{i2}, \ldots, a_{it}\} &= \mathit{plan}(E, g_i; \Theta, \mathcal{P}). \end{aligned} \tag{1}$$

Where E denotes the environment and T is the task, h is the historical experience. At time step t, the agent takes action a_{it} to achieve subgoal g_i. The LLM's parameters and prompts are represented by Θ and $\mathcal{P}$ respectively.

We build a multimodal experience pool by encoding both successful and failed cases at different granularity levels. Successful cases guide the planner's task decomposition, while failed cases provide data for quantitative analysis by the reflector.

Successful Cases. Each experience encodes key elements (*task, scene graph, plan, inventory*) to preserve decision paths and context. We use MineClip (Fan et al., 2022) instead of CLIP (Radford et al., 2021) for *scene graph* encoding due to its better instruction-following in Minecraft.

Failed Cases. These are encoded at a fine-grained level focusing on the subgoal. The reflector retrieves relevant failed cases as references to prevent repeated mistakes. Each case encodes key elements:

$$(\ G, A,\ S, suggestion\ )\,. \tag{2}$$

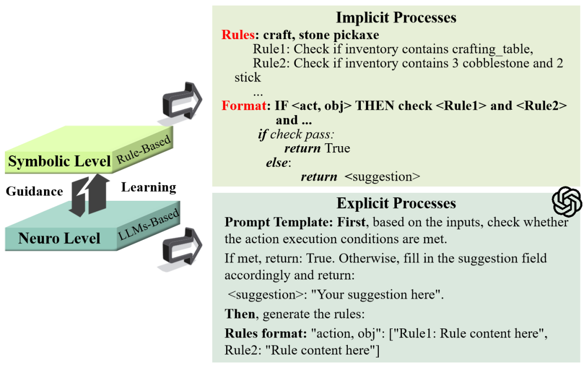
Where G is the failed subgoal, A is the corresponding action sequence, S is the failure-time scene graph, and $suggestion$ provides optimization recommendations.

2.3 Symbolic Reasoning-Driven Verifier

Similar to human procedural memory, the verifier stores skill knowledge in rule-based formats, driving symbolic reasoning to pre-check the reasoning errors of the plan and guide reliable replanning. While CAs need manual rules and LLMs reasoning lack reliability, our hierarchical neural-symbolic structure addresses these limitations. The two-layer design enables dynamic interaction between symbolic knowledge and neural knowledge representations. This structure enhances the verifier's knowledge representation and reasoning, improving planning efficiency.

2.3.1 Hierarchical Representation Structure

The structure of the verifier is illustrated in Figure 3 has two layers: symbolic representation for explicit "if-then" rules at the top, and LLM-based implicit knowledge at the bottom. The verifier has two functions:



Implicit Processes

Rules: craft, stone pickaxe
Rule1: Check if inventory contains crafting_table,
Rule2: Check if inventory contains 3 cobblestone and 2 stick
...

Format: IF <act, obj> THEN check <Rule1> and <Rule2> and ...

```
if check pass:
    return True
else:
    return <suggestion>
```

Explicit Processes

Prompt Template: First, based on the inputs, check whether the action execution conditions are met.

If met, return: True. Otherwise, fill in the suggestion field accordingly and return:

<suggestion>: "Your suggestion here".

Then, generate the rules:

Rules format: "action, obj": ["Rule1: Rule content here", Rule2: "Rule content here"]

Figure 3: **Hierarchical representation structure.** This neuro-symbolic structure enhances the verifier's knowledge representation and reasoning by combining explicit rules at the top layer with LLM-based implicit knowledge at the bottom. It enables autonomous rule through bidirectional design without any manual rule definition.

Action Feasibility Verification. Checks action prerequisites before execution. The verifier first uses symbolic rules for explicit checks, then LLMs for cases not covered by rules. This early detection helps avoid invalid environment interactions. The formulation is as follows:

$$f_{ij} = Verifier\,(\ a_{ij}, r_{ij}\ )\,. \tag{3}$$

Where f_{ij} denotes the verifier's feedback on action a_{ij}, r_{ij} represents the relevant rules.

Guiding Reliable Replanning. When actions are infeasible, the verifier provides specific feedback by referring to explicit rules to guide plan revision, reducing replanning counts and improving planning performance. The formulation is as follows:

$$P^* = \mathit{plan}\,(\ E, T, f_{ij}; \Theta, \mathcal{P}\ )\,. \tag{4}$$

2.3.2 Rule Learning and Evaluation Mechanism

Existing agents often rely on predefined rules or fine-tuning (Zeng et al., 2023) for rules learning, limiting generalization, and increasing computation. Our approach uses **top-down guidance** to convert rules into prompts guiding LLMs reasoning, and **bottom-up learning** to automatically extract symbolic rules from LLMs responses. The prompt can be seen in Appendix F.2. The bidirectional interaction mechanism, built upon a hierarchical representation structure, enables autonomous rule learning and eliminates the need for predefined rules, leading to improved generalization performance.

Then, we evaluate rule quality across three dimensions: common-sense consistency, condition clarity, and action feasibility. Rules that exceed a predefined quality threshold are incorporated into the rule pool for automatic expansion. More details can be found in Appendix F.3. The formulation is as follows:

$$eval\left(r\right) = w_c \cdot s_c\left(r\right) + w_p \cdot s_p\left(r\right) + w_a \cdot s_a\left(r\right). \quad (5)$$

Where s_c, s_p, and s_a represent common sense consistency, condition clarity, and action executability scores, respectively, with corresponding weights w_c, w_p, and w_a.

2.4 Metacognitive Reflector

In open-world environments, plans may fail during execution despite correct initial planning. This is mainly due to both LLMs and CAs struggling to fully model open-world uncertainties. Drawing from human metacognition, Metagent-P introduces a metacognitive reflector. Its mechanism can be seen in Figure 2. The reflector uses a three-step process of monitoring, evaluation, and regulation to enable adaptive planning. The formula is as follows:

$$\begin{aligned} s_i &= Reflector\left(\, E, g_i,\ h; \Theta, \mathcal{P}\,\right), \\ P^{*} &= plan\left(\, E, T, s_i; \Theta, \mathcal{P}\,\right). \end{aligned} \quad (6)$$

Where s_i denotes the reflector's suggestion on refining the subgoal g_i.

Monitoring. Following human brain patterns, not all cognitive activities trigger metacognitive evaluation (Lai, 2011). It triggers upon anomalies, such as excessive replanning counts or execution failures. Then the monitor collects environment and agent's information, helping analyze anomalies during evaluation.

Evaluation is similar to human self-examination. While LLMs may hallucinate (Varshney et al., 2023) and often show overconfidence in their responses (Lin et al.), we enhance the reliability of their confidence by extending the CoT-style prompting (Dohan et al., 2022). Traditionally, LLM's answer A is sampled from $p\left(\,A \,|\, Q,\ T,\ \theta\,\right)$, where Q is the query, T is thought chain, and θ represents model parameters. Instead of simple error reflection, we collect two component thoughts T_e and T_h, representing the current observation and historical experiences. The final answer is sampled from $p\left(\,A \,|\, Q,\ T_e,\ T_h,\ \theta\,\right)$. Additionally, we compute an integrated confidence score to evaluate planning rationality. If irrational, failed cases guide replanning; if rational, exploration continues.

$$\begin{aligned} confidence &= \alpha \cdot C_{history} + \beta \cdot C_{LLM}, \\ C_{\text{History}} &= Sim\left(v_1, v_2\right) = \frac{v_1 \cdot v_2}{\|v_1\|\,\|v_2\|}. \end{aligned} \quad (7)$$

Where $C_{history}$ measures the similarity between plans v_1 and v_2 using cosine similarity $[\,0,\ 1\,]$. C_{LLM} represents the LLM's verbalized confidence $[\,0,\ 1\,]$. α and β are weights where $\alpha + \beta = 1$.

Regulation. Metagent-P can self-correct and improve based on evaluation results. Specifically, when confidence exceeds 0.5, the evaluation conclusion is accepted. Otherwise, re-evaluation is triggered.

3 Experiment

3.1 Experimental Setup

3.1.1 Environment

The experiments are carried out in MineDojo (Fan et al., 2022), a Minecraft benchmark with various tasks and standardized evaluation. Compared to directly using multimodal large models, we utilize OpenAI's GPT-4-turbo (Achiam et al., 2023) and integrate MineClip (Fan et al., 2022) vision encoder for Minecraft visual parsing, which achieves better instruction-following capabilities (Qin et al., 2024). Text-embedding-ada-002 (OpenAI, 2022) API is used for efficient storage and retrieval.

3.1.2 Task Setting

We select 50 tasks from MineDojo (Fan et al., 2022) varying in duration and complexity, focusing on obtainable overworld items. Tasks are categorized into five difficulty levels (basic, easy, medium, hard, and complex) based on the minimum required subgoals. All Minecraft tasks information can be found in Appendix D.2

3.1.3 Evaluation Metrics

Metagent-P starts in survival mode with an empty inventory. We conducted each task 30 times with different seeds to simulate various scenarios. We design a specific action space for agent control. More details about the action space can be found in Appendix C.2. Two evaluation metrics are used:

Success Rate (SR): Measures the accuracy of agent planning.

Replanning Counts (RC): Indicates the number of times an agent recovers from failed states to eventually complete the task successfully. Lower CT values suggest stronger problem-solving capabilities and higher planning efficiency.

3.1.4 Baselines

We compare Metagent-P with three representative baselines in the experiments, please refer to Appendix D.1 for more details. **DEPS** (Wang et al., 2023), employs an LLM-based planner and reflector, but lacks visual information planning. **MP5** (Qin et al., 2024), combining visual information and using successful experiences to achieve reliable planning. **Huamn players** (Li et al.), ten experienced Minecraft players participated in the evaluation, with their average success rate for each task serving as the human-level baseline.

3.2 Main Results

Task	Metric	DEPS	MP5	Human	Metagent-P
Basic	SR	91.00	97.00	100	**99.67**
	RC	4.14	2.73	-	**0.2**
Easy	SR	80.33	84.67	100	**99.33**
	RC	11.32	7.41	-	**3.75**
Medium	SR	53.67	71.33	100	**88.33**
	RC	26.78	20.44	-	**9.63**
Hard	SR	19.20	43.00	86	**53.00**
	RC	36.81	26.72	-	**17.04**
Complex	SR	0.8	9.67	16.98	**20.20**
	RC	41.54	38.42	-	**26.33**

Table 2: **Main results of Metagent-P and baselines on Minecraft tasks.** We report the average success rate (SR)(%) and average replanning count (RC) on each task group, the full results can be found in the Appendix E.

Table 2 shows that Metagent-P significantly outperforms other LLM-based agents (DEPS and MP5) across all tasks. While achieving comparable performance to humans in basic and easy tasks with a nearly 100% success rate. Specifically, in complex tasks, Metagent-P achieves a 20.20% success rate, surpassing human performance by 18.96% and baselines DEPS and MP5 by 25 and 2 times respectively. Furthermore, Metagent-P requires 34% fewer re-planning times compared to baselines, highlighting its effectiveness and efficiency in long-term tasks.

Metagent-P's superior performance over human players in complex tasks but not in simple tasks is an intriguing finding. We believe this might be attributed to the following reasons: **(1)** In basic tasks with limited problem spaces and straightforward actions, human intuition and common sense enable efficient goal-to-action mapping. Conversely, Metagent-P's comprehensive process of task decomposition, symbolic reasoning, and plan verification may introduce overhead. **(2)** As tasks grow more complex, the problem space expands exponentially with longer planning horizons. While humans face cognitive limits in tracking actions and adapting to changes. However, Metagent-P's neural-symbolic framework with metacognitive reflection enables logically sound planning and dynamic environmental adaptation. **(3)** Additionally, Metagent-P achieves self-evolution through cognitive shaping and adaptive optimization (see Section 3.4.2), leading to more efficient and robust planning compared to humans' heuristic-based approaches and limited learning capacity.

3.3 Ablation Study

To evaluate Metagent-P components, we conducted ablation experiments on varying difficulty tasks. The results are shown in Table 3. The base model achieves an 80.33% success rate in basic task, showing the basic planning abilities of the LLM-based agent. Adding **Experience-Enhanced Planner** achieves the first success in the complex task, with a 3.33% success rate, demonstrating the value of multimodal experience in complex planning. The **Symbolic Reasoning-Driven Verifier** enhances performance across all tasks, particularly in complex task where it increases success rates 4 times and cuts replanning counts by 46%. This might be caused by identifying potential errors and providing guidance for efficient replanning. The full Metagent-P model with **Metacognitive Reflector** enhances performance on difficult tasks, particularly complex task, doubling the success rate to 33.33%, indicating its necessity for long-term tasks.

3.4 Exploring Self-Evolution Capabilities

Metagent-P's self-evolution capability distinguishes it from previous methods. We analyze it from two perspectives: cognitive shaping and adaptive optimization. **Cognitive shaping** helps agents both build knowledge through reusable rules and gain experience for a better understanding of their world. **Adaptive optimization** focuses on the agent's ability to dynamically adjust its plan in response to changing environments. This allows the agent to maintain strong performance, even when facing unexpected challenges.

Ablation Setting				SR (RC)				
Base	Planner	Verifier	Reflector	🟫	⛏	⛏	⛏	💎
✓				80.33(6.5)	63.33(10.2)	46.67(32.1)	16.67(44)	0.00(-)
✓	✓			93.33(2.2)	83.33(6.4)	63.33(21.4)	43.33(37.2)	3.33(44.9)
✓	✓	✓		100.00(1.5)	93.33(4.6)	83.33(13.3)	53.33(21.8)	16.00(24.2)
✓	✓	✓	✓	100.0(1.5)	96.67(4.5)	90.00(9.6)	56.67(18.7)	33.33(22.1)

Table 3: **Ablation study results.** We report the success rate (SR) and replanning count (RC) on each task, in the format of "SR (RC)". The base model includes the planner, controller, and reflector, with both the planner and reflector using only an LLM.

3.4.1 Cognitive Shaping

To study Metagent-P's cognitive shaping process, we test it in Minecraft technology tree exploration. As Table 4 shows, Metagent-P builds its rule and experience pools by abstracting knowledge into symbolic rules and recording successful and failed cases, enriching them with increased exploration tasks. This enrichment process demonstrates Metagent-P's cognitive shaping capabilities through exploration. Additionally, Figure 4 shows this cognitive shaping process improves planning performance, achieving higher success rates with fewer replanning counts.

Cognitive Shaping	Number of Exploratory Tasks				
	20	40	60	80	100
Rules Num.	54	109	134	178	216
Experiences Num.	26	57	102	157	224

Table 4: **Cognitive shaping study.** We analyze the growth of rule and experience pools as Metagent-P explores 100 tasks in the Minecraft technology tree.

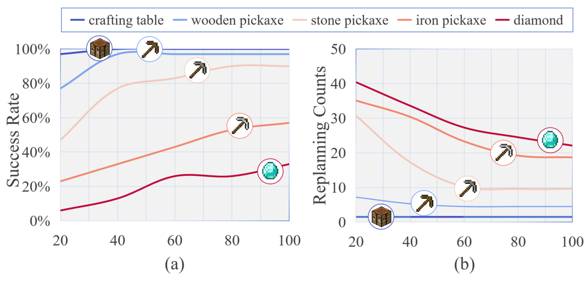


Figure 4: **Planning performance during cognitive shaping.** We track the planning success rates (a) and replanning counts (b) as Metagent-P explores 100 tasks in the Minecraft technology tree.

3.4.2 Adaptive Optimization

We evaluated how Metagent-P adapts to environmental changes. As is known, long-term tasks involve longer interaction with the environment, making them more susceptible to environmental

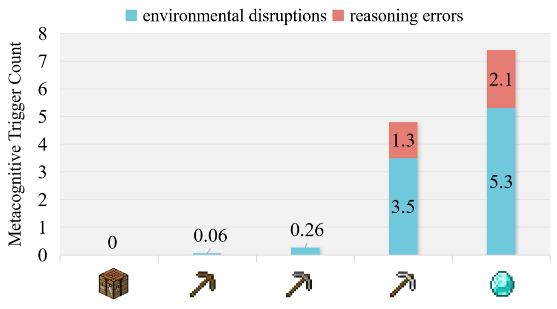


Figure 5: **Metacognitive trigger counts.** We tested each difficulty level 30 times, counting triggers from environmental disruptions and reasoning errors separately.

changes. Figure 5 shows more complex tasks trigger more metacognitive responses. This demonstrates the reflector's ability to actively adapt plans to environmental changes, notably enhancing performance on complex tasks. Furthermore, our analysis shows 77.0% of triggers come from environmental changes rather than reasoning errors (like failing to mine "cobblestone" ⬛ witout a "wooden pickaxe" ⛏), indicating the verifier could prevent most of the reasoning errors in advance, while the reflector mainly handles environmental disturbances.

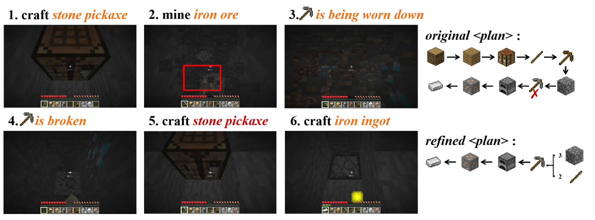


Figure 6: **Illustration of metacognition reflection mechanism.** In a "obtain 1 iron pickaxe" task, despite correct initial planning and completing the stone pickaxe subgoal, environmental disruption broke the tool. The reflector identifies this and adds a new stone pickaxe crafting subgoal, actively adapting to environmental disruptions before failure occurs.

We have shown a case of "obtain 1 iron pickaxe" ⛏ in Figure 6, monitor module triggers upon detecting execution anomaly (excessive replanning counts). Based on observations and experiences, the evaluation module identifies a broken tool and suggests: "stone pickaxe is broken, we need another one." The regulation module then adds a subgoal to re-craft the stone pickaxe ⛏. Notably, the subgoal had already been completed but got damaged due to environmental disruptions. The reflector actively adapts plans to environmental changes before failure occurs, further enhancing Metagent-P's flexibility.

4 Related Work

4.1 Planning with LLM-Based Agents

We summarise the differences of existing Minecraft agents in Appendix D.1. Early methods based on symbolic planning and reinforcement learning (Oh et al., 2017; Wichlacz et al., 2019) have struggled with complex tasks in Minecraft (Wikipedia contributors, 2024) due to modeling difficulties and sample inefficiency.

The rapid advancement of LLMs has enabled their use as planners and reflectors in Minecraft studies like DEPS (Wang et al., 2023) and Plan4mc (BAAI, 2023). However, their semantic reasoning rather than symbolic (Tang et al., 2023) poses challenges for consistency and reliability in multi-step reasoning (Dziri et al., 2024). Recent works like Jarvis-1 (Wang et al., 2024), Optimus-1 (Li et al.), and MP5 (Qin et al., 2024) have introduced external memory to address this. However, most methods overlook failed cases and lack effective experience encoding. Additionally, reflector design remains underexplored, as LLMs alone struggle to provide accurate feedback for replanning.

Our Metagent-P includes an Experience-Enhanced Planner that conducts multi-granular encoding of successful and failed experiences to enhance in-context learning. A Symbolic Reasoning-Driven Verifier that employs a neural-symbolic structure, combining LLMs' world knowledge with CAs' symbolic reasoning to effectively pre-check reasoning errors in plans. A Metacognitive Reflector to actively adapt to environmental disruptions.

4.2 Planning with CAs

Cognitive Architectures (CAs) Laird et al., 2017, such as Soar (Laird, 2019) and ACT-R (Anderson, 2009), include perception, motor, working memory, semantic memory, episodic memory and procedural memory. Among these, procedural memory stores condition-action rules and drives symbolic reasoning. By representing knowledge, actions, and states symbolically, CAs apply these rules to match current observations and infer appropriate actions, producing logical and reliable plans. Systems like Rosie built on Soar (Mininger and Laird, 2022) have demonstrated goal-oriented task learning, reasoning, and planning. However, due to their heavy reliance on domain-specific and task-tailored knowledge, planning with CAs has faced challenges in knowledge representation and scalability (Gunetti et al., 2013b,a; Lieto et al., 2018), making it difficult to accomplish open-world tasks effectively.

4.3 Metacognition

Metacognition, a higher-order cognitive capability (Wang et al., 2025), enables self-reflection and evaluation of cognitive processes (Flavell, 1979). In humans, it drives strategy optimization through continuous reflection on thought patterns. Recently, Integrating metacognitive abilities into AI has shown potential for enhancing autonomous learning and introspective reasoning. Some studies(Cox, 2005) have introduced introspection and self-management into cognitive systems, improving adaptability and robustness, while others (Conway-Smith and West, 2024) have used metacognition to boost self-directed learning in LLMs or designed metacognitive prompts (Wang and Zhao, 2023) to enhance comprehension. However, current designs still lack sophistication in handling dynamic, uncertain open-world environments.

5 Conclusion

We propose Metagent-P for open-world planning, consisting of four components: planner, verifier, controller, and reflector. It integrates the world knowledge of LLMs, the symbolic reasoning capabilities of cognitive architectures, and the self-reflection characteristic of metacognition, forming a "planning-verification-execution-reflection" framework. Experimental results show that Metagent-P outperforms current state-of-the-art agents across 50 Minecraft tasks. In long-term tasks, it improves performance by 2 to 25 times, reduces average replanning counts by 34%, and exceeds the average human success rate by 18.96%. Through step-by-step exploration, Metagent-P demonstrates self-evolution, achieving more reliable and efficient adaptive planning. Metagent-P's symbol-neural integration and metacognitive enhancement offer valuable insights for artificial general intelligence research and applications, bringing us closer to human intelligence.

6 Limitation

In Metagent-P, we aim to improve open-world planning through an Experience-Enhanced Planner, Symbolic Reasoning-Driven Verifier, and Metacognitive Reflector. However, the study has limitations: 1) Metagent-P is primarily designed for single-agent scenarios, with limited multi-agent collaboration; 2) We directly utilize the MP5 controller (Qin et al., 2024) enhances instruction-following, but it lacks low-level control due to insufficient video-action training data, limiting its ability to mimic human behavior more closely, please refer to the Appendix B for more details.

Future work will expand Metagent-P to multi-agent settings and develop an exploration learning mechanism for Metagent-P to autonomously collect and annotate video-action data, enabling the training of a more effective low-level action controller.

7 Ethics Statement

We propose Metagent-P to address open-world planning challenges. Our approach combines the world knowledge of LLMs, the symbolic reasoning capabilities of cognitive architectures, and the self-reflection characteristic of metacognition to solve long-term tasks and adaptive planning problems in open-world environments. Our experiments were conducted in MineDojo, an open-source research framework built on Minecraft that provides a rich, open-ended environment for developing and evaluating AI agents. MineDojo includes a comprehensive API, task datasets, and benchmarks for embodied AI research. The framework is released under the MIT license and built on Minecraft's official APIs, ensuring research reproducibility and ethical compliance. Additionally, we have submitted our code to facilitate future research and implementation of our methods.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

John R Anderson. 2009. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press.

PKU BAAI. 2023. Plan4MC: Skill Reinforcement Learning and Planning for Open-World Minecraft Tasks. *arXiv preprint arXiv:2303.16563*.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*.

Brendan Conway-Smith and Robert L West. 2024. Toward Autonomy: Metacognitive Learning for Enhanced AI Performance. In *Proceedings of the AAAI Symposium Series (AAAI)*.

Michael T Cox. 2005. Metacognition in Computation: A Selected Research Review. *Artificial intelligence*, 169(2):104–141.

David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A Saurous, Jascha Sohl-Dickstein, et al. 2022. Language Model Cascades. *arXiv preprint arXiv:2207.10342*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. 2024. Faith and fate: Limits of transformers on compositionality. *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*.

John H Flavell. 1979. Metacognition and Cognitive Monitoring: A New Area of Cognitive-Developmental Inquiry. *American psychologist*, 34(10):906.

Paolo Gunetti, Haydn Thompson, and Tony Dodd. 2013a. Autonomous Mission Management for UAVs Using Soar Intelligent Agents. *International Journal of Systems Science*, 44(5):831–852.

Paolo Gunetti, Haydn Thompson, and Tony Dodd. 2013b. Simulation of a Soar-Based Autonomous Mission Management System for Unmanned Aircraft. *Journal of Aerospace Information Systems*, 10(2):53–70.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196*.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering Diverse Domains through World Models. *arXiv preprint arXiv:2301.04104*.

Emily R Lai. 2011. Metacognition: A Literature Review. *Always learning: Pearson research report*, 24:1–40.

John E Laird. 2019. *The Soar Cognitive Architecture*. MIT press.

John E Laird, Christian Lebiere, and Paul S Rosenbloom. 2017. A Standard Model of the Mind: Toward a Common Computational Framework Across Artificial Intelligence, Cognitive Science, Neuroscience, and Robotics. *Ai Magazine*, 38(4):13–26.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid Multimodal Memory Empowered Agents Excel in Long-Horizon Tasks. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS)*.

Antonio Lieto, Christian Lebiere, and Alessandro Oltramari. 2018. The Knowledge Level in Cognitive Architectures: Current Limitations and Possible Developments. *Cognitive Systems Research*, 48:39–55.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching Models to Express Their Uncertainty in Words. *Transactions on Machine Learning Research*.

Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. 2022. Deep Learning, Reinforcement Learning, and World Models. *Neural Networks*, 152:267–275.

Aaron Mininger and John E Laird. 2022. A Demonstration of Compositional, Hierarchical Interactive Task Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. 2017. Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 2661–2670. PMLR.

OpenAI. 2022. New and improved embedding model. https://openai.com/index/new-and-improved-embedding-model/.

Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. 2024. MP5: A Multi-modal Open-ended Embodied System in Minecraft via Active Perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR.

Stuart J Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson.

Xiaojuan Tang, Zilong Zheng, Jiaqi Li, Fanxu Meng, Song-Chun Zhu, Yitao Liang, and Muhan Zhang. 2023. Large Language Models are In-Context Semantic Reasoners rather than Symbolic Reasoners. *arXiv preprint arXiv:2305.14825*.

Neeraj Varshney, Wenlin Yao, Hongming Zhang, Jianshu Chen, and Dong Yu. 2023. A Stitch in Time Saves Nine: Detecting and Mitigating Hallucinations of LLMs by Validating Low-Confidence Generation. *arXiv preprint arXiv:2307.03987*.

Pranav Narayanan Venkit, Mukund Srinath, and Shomir Wilson. 2022. A Study of Implicit Bias in Pretrained Language Models Against People with Disabilities. In *Proceedings of the 29th International Conference on Computational Linguistics (COLING)*.

Xintong Wang, Jingheng Pan, Liang Ding, Longqin Jiang, Longyue Wang, Xingshan Li, and Chris Biemann. 2025. Cogsteer: Cognition-inspired selective layer intervention for efficiently steering large language models. In *Findings of the Association for Computational Linguistics (ACL)*.

Yuqing Wang and Yun Zhao. 2023. Metacognitive Prompting Improves Understanding in Large Language Models. *arXiv preprint arXiv:2308.05342*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. 2024. JARVIS-1: Open-World Multi-task Agents with Memory-Augmented Multimodal Language Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Julia Wichlacz, Alvaro Torralba, and Jörg Hoffmann. 2019. Construction-Planning Models in Minecraft. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Wikipedia contributors. 2024. Minecraft. https://en.wikipedia.org/wiki/Minecraft.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. AgentTuning: Enabling Generalized Agent Abilities for LLMs. *arXiv preprint arXiv:2310.12823*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory. *arXiv preprint arXiv:2305.17144*.

A Challenges of the Open-World Environments

Open-world environments present significantly greater challenges for agent development compared to traditional controlled scenarios. While conventional environments feature limited scope, predictable dynamics, and a restricted task set, open worlds demand agents capable of handling diverse and complex challenges. In this section, we examine two fundamental challenges that emerged during our development of Metagent-P.

Challenge I: Task Complexity. Tasks in open-world environments are typically long-term and diverse. Take the task of "obtaining diamonds" as an example - it needs to be broken down into 11 subtasks, such as "crafting a wooden pickaxe", "mining stone blocks", and "crafting a stone pickaxe". These subtasks have complex logical relationships and sequential dependencies, which challenge the agent's planning capabilities. Therefore, agents need symbolic reasoning abilities to decompose complex goals and generate correct execution plans.

Moreover, tasks in open worlds are highly diverse. Agents need to continuously enrich their knowledge base and experience pool during exploration to improve their ability to handle various tasks. This process essentially shapes the agent's cognitive capabilities, enabling it to understand the intrinsic connections between tasks and develop general problem-solving strategies.

Challenge II: Environmental Disruption. Open-world environments are dynamic and full of various interfering factors. The open world is characterized by uncertainty and constant change: terrain may shift, weather systems affect visibility and movement, day-night cycles present different challenges, hostile entities may appear unexpectedly, and even other players' actions can influence the environmental state. These environmental disruptions create uncertainties that demand high adaptability from agents.

To address environmental disruptions, agents need metacognitive abilities similar to humans. Metacognition is the awareness and understanding of one's own cognitive processes, including environmental perception, self-behavior monitoring, and regulation. Through metacognition, agents can dynamically evaluate how environmental changes affect their current action plans, learn from failures, and adjust plans when facing new situations, demonstrating strong adaptive planning capabilities. This ability helps agents robustly complete tasks in complex and dynamic open worlds.

To address these challenges, our Metagent-P introduces four key components: an Experience-Enhanced Planner, a Symbolic Reasoning-Driven Verifier, an Action Controller, and a Metacognitive Reflector. It integrates the world knowledge of LLMs, the symbolic reasoning capabilities of cognitive architectures, and the self-reflection characteristic of metacognition, forming a "planning-verification-execution-reflection" interactive planning framework. Notably, during progressive open-world exploration, Metagent-P continuously shapes its cognition and adaptively optimizes its performance, achieving self-evolution.

B Limitation and Future Work

In the framework of Metagent-P, we are dedicated to leveraging the proposed Experience-Enhanced Planner, Symbolic Reasoning-Driven Verifier, and Metacognitive Reflector to enhance the agent's performance and efficiency in open-world planning. However, this study has several limitations: 1) Metagent-P currently focuses primarily on single-agent scenarios, with only preliminary support for multi-agent collaboration; 2) We directly employ the controller proposed by MP5 (Qin et al., 2024), which demonstrates improved instruction following capabilities and can effectively clarify Planner-generated actions such as equip, move, and craft. However, low-level commands (e.g., keyboard and mouse operations) more closely mimic human behavior, due to the current lack of high-quality video-action training data, we cannot achieve a more effective low-level action controller.

Future work will focus on extending the "planning-verification-execution-reflection" interactive planning framework to multi-agent scenarios. Additionally, we will design an exploration learning mechanism for Metagent-P to actively collect and automatically annotate high-quality video-action training data during world exploration, which will be used to train a low-level action controller with enhanced instruction-following capabilities.

C Environment Setting

C.1 Observation Space

To achieve embodied agent-like perception, we restrict environmental information access and imple-

Index	Action	Argument	Description
1	Find	Object	Navigate the environment to locate a target object
2	Approach	Object	Move towards a target object
3	Mine	Object, tool	Using a tool to break down a block or object
4	Craft/Smelt	Object, platform	Combine raw materials or items at a crafting table or furnace to create a new item
5	Attack	Object, tool	Attack entity
6	Equip	Object	Equip or hold a specific item or tool in the player's hand
7	Dig-down	Y-level, tool	Dig or excavate downwards to a specified y-coordinate using a tool
8	Dig-up	Tool	Dig or excavate upwards using a tool

Table 5: **The definition of the Action Space we use in MineDojo simulator.**

Agents	Planning components				Input	Memory			External knowledge
	Planner	Verifier	Controller	Reflector					
DEPS (Wang et al., 2023)	✓		✓	✓	T	Raw	T	Succ.	-
Plan4m (BAAI, 2023)	✓		✓		T	-	-	-	Pre-defined skill
GITM (Zhu et al., 2023)	✓		✓	✓	T	Raw	T	Succ.	-
Jarvis-1 (Wang et al., 2024)	✓		✓	✓	T+V	Raw	T+V	Succ.	-
MP5 (Qin et al., 2024)	✓		✓	✓	T+V	Voctor	T	Succ.	External train data
Metagent-P (our)	✓	✓	✓	✓	T+V	Voctor	T+V	Succ.+Fail	-

Table 6: **Minecraft Agents.**

ment a human-like observation system. Rather than using omniscient perception methods like LiDAR rays in GIMT (Zhu et al., 2023), our agent relies on egocentric RGB images to perceive its environment. The observation space is structured into two main components:

Perceptual observations:

Egocentric, Minecraft-style RGB images, 3x3x3 voxels encountered by the agent (object properties)

Status observations:

Relevant ancillary text information (health statistics, inventory details)

This dual observation system enables situation-aware planning.

C.2 Action Space

We directly employ the controller proposed by MP5 (Qin et al., 2024), which demonstrates improved instruction following capabilities and can effectively clarify Planner-generated action sequences in Table 5. For navigation tasks, such as the "find" action, the agent moves while continuously updating its orientation based on MineDojo's field of view data to locate and approach targets.

D Benchmark Suite

D.1 Minecraft Agents

While reinforcement learning (RL) (Matsuo et al., 2022) has emerged as the dominant approach for creating game-playing agents, its application to Minecraft has revealed significant limitations. RL agents face two major challenges: they demand extensive training (for instance, DreamerV3 (Hafner et al., 2023) requires approximately 30 million steps just to learn diamond collection) and struggle with task generalization (as evidenced by VPT (Baker et al., 2022), which needs separate agents for exploration and diamond mining tasks).

In this section, we summarise the differences between existing LLM-based agents. As shown in Table 6, traditional LLM-based approaches use a "planning-execution-reflection" framework, while Metagent-P advances the LLM-based agent planning paradigm by introducing a validator. The neural-symbolic hierarchical representation structure enhances the validator's reasoning capabilities through symbolic reasoning, enabling early detection of planner errors and avoiding unnecessary environmental interactions, thereby improving planning efficiency. Additionally, rule-based reliable feedback improves the planner's replanning accuracy. Experimental results demonstrate that this approach effectively enhances both planning success rate and efficiency. Furthermore, successful and failed experiences are stored separately and encoded at different granularities. This approach leverages successful experiences to guide task decomposition by the planner while incorporating failed cases into the reflector's design, enabling a more comprehensive utilization of Metagent-P's exploration experiences. Metagent-P operates without external data or predefined rules, instead evolving through progressive exploration. It continuously enriches its experience and rule repositories while leveraging metacognitive reflection mechanisms for adaptive planning, achieving self-evolution.

D.2 Minecraft Task Details

We conducted a systematic evaluation of Metagent-P using MineDojo tasks. First, we analyzed and categorized tasks based on their minimum required sub-goals (Table 7), acknowledging that the actual number may vary depending on initial conditions and biome types. Our experimental setup includes:

- 50 selected tasks arranged by minimum sub-goal complexity.
- Five difficulty levels: basic, easy, medium, hard, and complex.
- Task parameters: complexity level, name, minimum sub-goals, required tools, platforms, initial inventory, and maximum episode steps.
- Time limits ranging from 12,000 steps (basic) to 36,000 steps (complex).

E Full Experimental Results

We list the full results of each task below (Table 8), with details including task level, task name, success rate (SR), and average replanning counts (RC). To account for Minecraft's open-world variability, we performed 30 trials per task with randomized initial positions and environment seeds. All trials were conducted under survival mode with an empty initial inventory. Success was defined as target acquisition within the time limit, while agent death constituted failure.

F PROMPT DESIGN

F.1 Decision-Making Prompt

We show the prompt templates for the Experience-Enhanced Planner in Table 9.

F.2 Generate Rule Prompt

We show the prompt templates for Symbolic Reasoning-Driven Verifier in Table 10. Implement bottom-up automatic rule learning to extract explicit rules from LLMs' implicit knowledge, without requiring predefined knowledge.

F.3 Rule Evaluation Prompt

We show the prompt templates for rule evaluation in Table 11. To ensure the reliability of rules extracted from LLMs, we designed a structured prompt-based rule evaluation mechanism that considers three dimensions:

Common-Sense Consistency: whether the rule aligns with common sense and basic logic;

Condition Clarity: whether the rule's conditions are sufficiently explicit and complete;

Action Feasibility: whether the actions are clear and executable.

Given a rule extracted from LLM output: *"IF: iron ore discovered; THEN: mine with stone pickaxe"*, denoted as r, we apply the evaluation function $eval(r)$ across multiple dimensions. Rules that exceed a predefined quality threshold are incorporated into the rule pool for automatic expansion. (Rules scoring above 0.85 are considered high-quality.)

This structured prompt-based evaluation mechanism enables the validator to continuously learn and correct from LLM outputs, refining its rule base and improving generalization in open environments.

Task level	Task	Sub-goal Num.	Tools/Platform	Step Setting
Basic	mine log	1	-	12K
	mine sand	1	-	
	mine sapling	1	-	
	mine wheat seeds	1	-	
	mine dirt	1	-	
	mine grass	1	-	
	craft plank	2	-	
	craft stick	3	-	
	craft button	3	-	
	craft crafting table	3	-	
Easy	craft chest	4	crafting table	12K
	craft bowl	4	crafting table	
	craft boat	4	crafting table	
	craft wooden slab	5	crafting table	
	craft wooden pressure plate	5	crafting table	
	craft ladder	5	crafting table	
	craft barrel	5	crafting table	
	craft wooden axe	5	crafting table	
	craft wooden pickaxe	5	crafting table	
	craft wooden sword	5	crafting table	
Medium	mine cobblestone	6	wooden pickaxe	24K
	mine coal ore	7	wooden pickaxe	
	craft furnace	7	crafting table	
	craft lever	7	crafting table	
	craft stone pickaxe	7	crafting table	
	craft stone axe	7	crafting table	
	craft stone hoe	7	crafting table	
	craft stone shovel	7	crafting table	
	craft stone sword	7	crafting table	
	mine iron ore	8	stone pickaxe	
Hard	smelt glass	9	furnace	36K
	smelt iron ingot	10	furnace	
	craft iron bars	11	crafting table	
	craft carpentry table	11	crafting table	
	craft iron pickaxe	11	crafting table	
	craft iron door	11	crafting table	
	craft iron trapdoor	11	crafting table	
	craft rail	11	crafting table	
	craft cauldron	11	crafting table	
Complex	obtain diamond	12	iron pickaxe	36K
	mine redstone	12	iron pickaxe	
	craft dropper	13	crafting table	
	craft redstone torch	13	crafting table	
	craft compass	13	crafting table	
	craft clock	13	crafting table	
	craft piston	13	crafting table	
	craft diamond pickaxe	13	crafting table	
	craft diamond sword	13	crafting table	
	craft raw gold block	13	crafting table	

Table 7: **The details of all the tasks.**

Task level	Task name	SR	RC
Basic	mine log	100.00	0.0
	mine sand	96.67	0.3
	mine sapling	100.00	0.0
	mine wheat seeds	100.00	0.0
	mine dirt	100.00	0.0
	mine grass	100.00	0.0
	craft plank	100.00	0.0
	craft stick	100.00	0.0
	craft button	100.00	0.2
	craft crafting table	100.00	1.5
Easy	craft chest	100.00	2.1
	craft bowl	100.00	2.4
	craft boat	100.00	2.4
	craft wooden slab	100.00	4.6
	craft wooden pressure plate	100.00	4.5
	craft ladder	100.00	4.1
	craft barrel	100.00	4.3
	craft wooden axe	96.67	4.7
	craft wooden pickaxe	96.67	4.5
	craft wooden sword	100	3.9
Medium	mine cobblestone	93.33	7.6
	mine coal ore	86.67	9.5
	craft furnace	90.00	10.9
	craft lever	86.67	9.9
	craft stone pickaxe	90.00	9.6
	craft stone axe	86.67	9.7
	craft stone hoe	90.00	9.4
	craft stone shovel	90.00	8.9
	craft stone sword	86.67	9.4
	mine iron ore	83.33	11.4
Hard	smelt glass	66.67	13.0
	smelt iron ingot	66.67	16.4
	craft iron bars	56.67	15.8
	craft carpentry table	53.33	17.5
	craft iron pickaxe	56.67	18.7
	craft iron door	50.00	16.1
	craft iron trapdoor	43.33	17.6
	craft rail	46.67	19.3
	craft cauldron	46.67	17.6
	craft minecart	43.33	18.4
Complex	obtain diamond	33.33	22.1
	mind redstone	33.33	26.9
	craft dropper	23.33	27.6
	craft redstone torch	20.00	24.3
	craft compass	23.33	27.5
	craft clock	16.00	25.9
	craft piston	16.00	26.7
	craft diamond pickaxe	13.33	28.2
	craft diamond sword	10.00	27.4
	craft raw gold block	13.33	26.7

Table 8: **Full experimental results of Metagent-P.**

Decision-Making Prompt

You are a helpful planner in Minecraft, and good at planning workflows to complete tasks. I will give you a task, for which you need to conceive a plan, and then create a workflow composed of a sequence of various actions to complete this task.
I will give you the following information:
task information:
task: The name of the task.
quantity: The required quantity for the task.
material: The necessary materials for achieving the task in your inventory.
tool: The primary tool necessary for this task is, for instance, a wooden pickaxe.
platform: The crafting station or block that is necessary for this task, for instance, a crafting table or a furnace.
current environment information: MineClip-encoded scene graph.
inventory: a dict representing the inventory, whose keys are the names of the objects and the values are their quantities.
The actions which can compose a workflow are as follows:
find (obj): used to find objects including blocks and creatures.
-obj: a string, the object to find, like "log", "cobblestone".
approach (obj): move close to a visible object including blocks and creatures.
-obj: a string, the object to move close to, like "log", "cobblestone", "iron ingot" and "pig".
craft (obj, materials, platform): craft the object with the materials and platform.
-obj: a dict, whose key is the name of the object and value is the object quantity, like "crafting table": 1.
materials: a dict, whose keys are the names of the materials and values are the quantities, like "planks": 4.
platform: a string, the platform used for crafting, like "furnace" and "crafting table".
mine (obj, tool): mine 1 object with the tool.
-obj: a string, which the object to mine, like "log".
-tool: a string, the tool used for mining, like "wooden pickaxe".
attack (obj, tool): fight 1 object with the tool within range
-obj: a string, which the object to mine, like "pig" and "zombie".
-tool: a string, the tool used for mining, like "wooden sword".
equip (obj): equip the object from the inventory.
-obj: a string, the object to equip, like "wooden pickaxe".
digdown (y_level, tool): dig down to the y-level with the tool.
-y_level: a number, the y-level to dig down to.
-tool: a string, the tool used for digging down, like "wooden pickaxe".
digup (tool): dig up to the ground from underground.
-tool: a string, the tool used for digging down, like "wooden pickaxe". Set to null if without any tool.
apply (obj, tool): apply the tool on the object.
-obj: a string, the object to apply to.
-tool: a string, the tool used to apply, like "bucket" and "shears".
RESPONSE FORMAT:
"workflow": [
"times": "the number of times the actions will perform", "actions": ["name": "action name", "args": "arg name": value, ...],
"times": "the number of times the actions will perform", "actions": ["name": "action name", "args": "arg name": value, ...]]
Ensure the response can be parsed by Python json.loads, e.g.: no trailing commas, no single quotes, etc.
Examples:
USER:
task information:
- task: cobblestone
- quantity: 4
- material: None
- tool: wooden pickaxe
- platform: None
current environment information: MineClip-encoded scene graph
inventory: "cobblestone": 2, "wooden pickaxe": 1
SYSTEM:
{ "workflow": [
{"times": "1", "actions": [{"name": "equip", "args": {"obj": "wooden pickaxe"} }] },
{"times": "1", "actions": [{"name": "digdown", "args": {"y_level":58, "tool": "wooden pickaxe"} }] },
{"times": "4", "actions": [{"name": "find", "args": {"obj": "cobblestone"} },
{"name": "approach", "args": {"obj": "cobblestone"} },
{"name": "mine", "args": {"obj": "cobblestone", "tool": "wooden pickaxe"} }] }] }

USER:

My information is as follows:
- The task is:
{task_information}
- The current environment:
{current_environment_information}
- The current inventory: {inventory}
-Based on experience: {episodic_memory}
-The reflection is: {feedback}
Please plan for the task.

Table 9: **Decision-Making Prompt.**

Generate Rule Prompt

You are well-versed in all the rules of Minecraft. Please complete the following two tasks:
Given the inputs of action and inventory, determine whether the action satisfies the execution constraints.
If it satisfies the constraints, return:
"return": {"success": true,
"suggestion": ""}
Otherwise, based on your analysis, fill in the suggestion field accordingly and return:
"return": {"success": false,
"suggestion": "Your suggestion here"}
Examples:
USER:
action: {"name": "craft", "args": {"obj": {"stone pickaxe": 1}, "materials": {"cobblestone": 3, "stick": 2}, "platform": "crafting table"}}
inventory: {"crafting table": 1, "stick": 2}
SYSTEM:
{"return": {
"success": false,
"suggestion": "Mine or Craft enough cobblestone first." },
"craft, stone pickaxe": ["Rule1: Check if inventory contains crafting table", "Rule2: Check if inventory contains 3 cobblestone and 2 stick"]}

Table 10: **Generate Rule Prompt.**

Rule Evaluation Prompt

Please evaluate this rule across three dimensions, providing a score from 0 to 1 for each:
1. Common-Sense Consistency: Does this rule align with Minecraft's mechanics and logic?
2. Condition Clarity: Is the trigger condition-specific and unambiguous?
3. Action Feasibility: Is the action executable, considering any prerequisites?
For each dimension provide a numerical score
Example:
USER:
Rule: IF iron ore is discovered; THEN mine with stone pickaxe
SYSTEM:
Common-Sense Consistency (w_c = 0.3)
Condition Clarity (w_p = 0.3)
Action Feasibility (w_a= 0.4)

Table 11: **Rule Evaluation Prompt.**