Journal of Computer Technology and Software

ISSN:2998-2383

Vol. 3, No. 9, 2024

Closed-Loop Multi-Round Planning for Large Language Model Agents via Self-Reflection and Error Correction

Haotian Zhu
New York University, New York, USA
htz1999.job@gmail.com

Abstract: To address the instability and error cascading problems in multi-round interaction planning for large language model agents, this paper proposes a multi-round planning method with self-reflection and error correction mechanisms. This method integrates task objective and constraint modeling, state updating, candidate plan generation, plan selection and scoring, risk assessment, reflection triggering, and local patching into a closed-loop process. After each round, the method performs consistency verification on intermediate states and constraint satisfaction based on observable feedback. When potential deviations are detected, a reflection module is triggered to attribute errors and locate patching points. An error correction module then minimizes key steps to restore feasible trajectories. Simultaneously, a unified scoring function is used to select from candidate plans, balancing task completion, tool availability, and constraint compliance, thereby reducing invalid rounds and violation risks. To facilitate process interpretation and diagnosis, this paper further provides visualization analysis methods, such as trajectory alignment and failure type distribution, to characterize the structural patterns of error triggering, reflection, and patching at different stages. Comparative experiments show that the proposed method performs better in terms of task success, tool invocation stability, and constraint violation control. It is also more efficient and stable in terms of multi-round interaction costs, verifying the effective role of closed-loop error correction in improving the reliability and controllability of planning.

Keywords: Closed-loop programming; constraint consistency; error attribution; local patching

1. Introduction

Large language model-based intelligent agents are gradually evolving from conversational generation to goal-oriented action systems, capable of performing complex tasks such as information retrieval, tool invocation, task decomposition, and process orchestration in open environments. Compared to one-time responses, multi-round planning more closely resembles the organization of real-world tasks because real-world problems often exhibit long-chain dependencies, dynamically changing constraints, and observable feedback during the process. However, the planning process based on large language models is still susceptible to factors such as error accumulation, amplified early decision biases, and insufficient utilization of environmental feedback, leading to deviations or failures in iterative progress. Ensuring that the agent maintains goal consistency throughout multi-round interactions and establishes stable constraints on the reliability of its outputs is crucial for its transition from usability to reliability[1].

Self-reflection and error correction mechanisms provide an important path to improving the reliability of multi-round planning. On the one hand, self-reflection enables the agent to realign its goals, constraints, and current state before and after each round of decision-making, identifying potential logical contradictions, information gaps, and uncertainties, thus avoiding blind progress without sufficient evidence. On the other hand, error correction mechanisms emphasize the ability to implement actionable corrective strategies after deviations are detected. These strategies include rewriting local sub-plans, rolling back critical decisions, resetting constraint priorities, or triggering more stringent verification processes, thereby limiting the impact of errors to localized areas and suppressing cascading effects. Embedding reflection and error correction into a multi-round planning loop enables agents to possess self-checking and self-recovery capabilities closer to those of engineering systems, significantly enhancing their robustness and consistency in complex tasks[2].

Multi-round planning agents with self-reflection and error correction capabilities have significant theoretical and practical value. At the methodological level, it propels planning from static generation to process control, enabling agents not only to propose solutions but also to continuously evaluate and constrain the quality of those solutions, promoting interpretable, controllable, and auditable agent design. At the application level, such capabilities help reduce failure rates and resource waste in high-cost tasks, improve continuous execution capabilities in long-term tasks, cross-tool collaboration, and dynamic environments, and provide fundamental support for safe use in higher-risk scenarios. Therefore, constructing a multi-round planning method around self-reflection and error correction mechanisms is of great significance for improving the reliability, generalization, and engineering implementation capabilities of large language model agents.

2. Related work

2.1 Task planning and decomposition methods for large language model agents

Existing research on task planning and decomposition for large language model agents typically transforms complex goals into a sequence of executable subtasks and refines the plan through multiple rounds of interaction. Common approaches include prompt-driven decomposition, hierarchical recursive generation of sub-goals, state-based iterative updates, and collaboration with external tools or retrieval systems to fill knowledge and information gaps. These methods have practical value in reducing the difficulty of single-step generation and improving task interpretability and controllability, especially for problems with clear processes or explicit constraints[3]. However, this direction also exposes several common challenges: decomposition criteria often rely on implicit heuristics, lacking explicit characterization of the minimum sufficient subtask granularity and boundary conditions; in multi-round planning, early decomposition deviations are easily amplified by subsequent steps, leading to error accumulation and goal drift; simultaneously, many methods focus more on the completeness of the generated plan but invest insufficiently in the plan's verifiability, failure recovery, and cross-round consistency constraints, resulting in unstable robustness in open environments[4].

From a critical perspective, existing methods still have structural shortcomings in terms of engineering feasibility. First, planning is often treated as a text generation problem, lacking process-level quality control signals, making it difficult to identify and correct errors midway. Second, task decomposition and execution are often loosely coupled, and execution feedback is not systematically absorbed for replanning and error correction, thus reducing the ability to close the loop. Third, different studies have inconsistent evaluation objectives; some emphasize completion rate, while others emphasize efficiency or interpretability, but there is insufficient systematic characterization of safety and constraint satisfaction, which makes method comparison and reuse costly. To more clearly present these differences and shortcomings, Table 1 provides a comparative critical summary of the key characteristics and main limitations of representative task planning and decomposition paradigms, highlighting their gaps in verifiability, stability, and feedback loop closure.

Table 1: A Critical Comparative Summary of Task Planning and Decomposition Methods

Method Paradigm	Core Approach	Strengths	Key Limitations	Impact on Multi-turn Planning
Prompt-driven Decomposition	Generate subtasks and step sequences primarily via prompt templates.	Simple to use; low migration and engineering cost.	Highly sensitive to prompt quality; lacks consistency constraints and verifiable criteria.	Prone to goal drift and error accumulation across turns.
Hierarchical Subgoal Recursion	Recursively decompose a high-level goal into a tree of subgoals.	Clear structure; relatively interpretable.	Unstable subgoal boundaries; granularity is hard to adapt; backtracking can be costly.	Early decomposition bias is amplified over long horizons.
State-Iterative Planning	Roll forward by generating the next action conditioned on the current state.	Adapts to dynamic environments; offers some online responsiveness.	Lacks a global optimal view; can fall into local greediness or loops.	Long-horizon stability is often weak for lengthy action chains.
Retrieval-Augmented Planning	Use external knowledge retrieval to support decomposition and decision-making.	More informed planning reduces hallucination risk.	Retrieval noise and evidence-selection bias are hard to control; limited cross-evidence consistency checks.	May introduce new biases and amplify uncertainty over multiple turns.
Tool-Calling Collaborative Planning	Map subtasks into an executable sequence of tool calls.	Higher executability; observable intermediate results.	Insufficient handling of tool failures and exception recovery; unstable calling policies.	Likely to break or drift when exceptions occur.
Rule/Constraint-Injected Planning	Inject constraints as rules or checkers during planning.	More controllable constraint satisfaction.	Incomplete rule coverage; weak conflict resolution and soft-constraint trade-offs.	May become overly conservative or get stuck in constraint-induced dead ends.

2.2 Research progress on self-reflection and self-correction mechanisms in language models

Existing research on the self-reflection and self-correction of language models has generally evolved from outcome-level repair to process-level control[5,6]. Early work relied heavily on multiple generations and rewriting processes, using reflection as a prompting process for re-expression or re-examination to fill in missing information, correct logical chains, or mitigate inconsistent answers. Subsequently, reflection was gradually integrated with explicit checking steps, such as adding consistency checks, constraint verification, evidence alignment, and error type diagnosis after generation, enabling the model to output not only the answer but also a self-assessment signal regarding the reliability of the answer. Further, some methods have begun to emphasize feedback-based iterative error correction, introducing external tools or environmental feedback to locate the causes of failure and triggering local replanning, alternative strategy selection, and step rollback, thus expanding error correction from static text modification to action-oriented strategy adjustments. Overall, the core contribution of this direction lies in pushing the model from a single-round generator to a self-regulating decision-making process, providing a methodological foundation for stable execution in complex tasks[7].

However, it should be noted that the effectiveness of self-reflection and self-correction is often overly optimistically assumed, and several structural contradictions and unclosed loops still exist in reality. First, reflection is often expressed in linguistic form, but lacks operational criteria to distinguish between useful reflection and superficial rationalization. This results in reflection texts appearing more complete but not actually reducing the error rate, and may even reinforce the model's confidence in its own errors. Second, error correction triggering mechanisms in many schemes remain empirical. There is a lack of unified risk thresholds and cost trade-offs regarding when to continue, when to roll back, and when to request external evidence, which can easily lead to efficiency

losses from over-correction or error propagation from under-correction. Third, reflection and correction face the problem of memory and state consistency in multi-round interactions. The model may give inconsistent interpretations of the same constraint in different rounds, causing implicit goal drift. Therefore, current research is more like adding a self-checking appearance to the generation process, and it still falls short of a verifiable, reproducible, and auditable error correction loop. This leaves clear room for the design of a systematic mechanism for multi-round planning in the future.

3. Method

We model multi-round planning as a closed-loop decision-making process. In each round, the agent simultaneously maintains the task objective, environmental observations, historical actions, and a retrievable structured memory, generating executable sub-plans based on this foundation. Let the context state of round t be s_t, obtained by fusing the initial instructions, the latest observations, and historical trajectories. To prevent early deviations from being amplified in long-chain tasks, we introduce explicit self-reflection signals and error correction trigger conditions in each round, transforming planning from a one-time generation to constrained iterative optimization. State updates employ a simple aggregation form, writing observations and the results of the previous plan into memory before generating the state representation for the next round, thus ensuring a clear data flow between plan updates and execution feedback. This paper presents the overall model architecture, as shown in Figure 1.

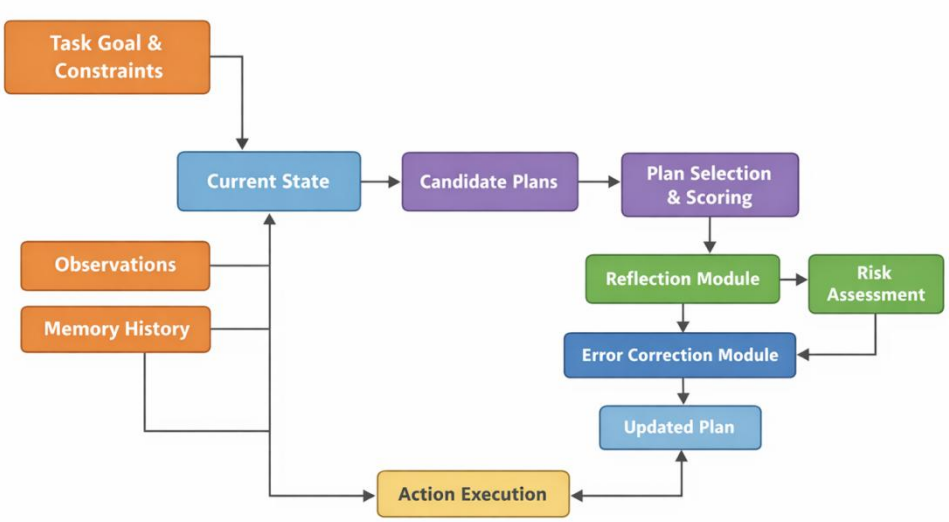

Figure 1. Overall model architecture

Correspondingly, we use a lightweight scoring function to measure the feasibility and consistency of the current plan and a risk indicator to determine whether to enter the error correction branch. The entire method emphasizes three points: First, the planning output must be a structured sequence of actions, not just narrative text; second, reflection must produce actionable error localization and correction suggestions; third, error correction must be locally rollbackable, prioritizing the repair of steps most likely to lead to failure, rather than indiscriminately rewriting the entire plan.

$$s_t = Merge(s_{t-1}, o_t, a_{t-1})$$

During the generation phase, the agent generates a set of candidate plans $P_t = \{p_t^{(1)}, \ldots, p_t^{(K)}\}$ under state s_t. Each plan consists of several actions and includes necessary parameters and preconditions. To maintain simplicity and implementability, each candidate plan is given a linear score, which is composed of the degree of goal matching and constraint satisfaction, with weights used to balance completion and safety. The plan with the highest score is then selected as the master plan for this round. The core of this design is to transform the planning selection from a single generation to a candidate comparison, allowing subsequent reflection and error correction to revolve around comparable candidate differences, rather than repeatedly seeking consistency on a single path.

$$Score(p, s_t) = \alpha Goal(p, s_t) + \beta Cons(p, s_t)$$

$$p_t = arg \max_{p \in P_t} Score(p, s_t)$$

During the reflection phase, we input the plan p_t into a self-checking function to generate a reflection record r_t. This record contains two types of information: a set of potential error points and a set of corresponding remedial suggestions. To ensure the controllability of error correction triggering, we define a simple risk indicator ρ_t, which normalizes and aggregates the number of constraint violations and uncertain items identified during the self-check. Error correction is triggered when the risk exceeds a threshold; otherwise, execution or further refinement is initiated. The key here is not to make the reflection longer, but to ensure that the reflection outputs quantifiable signals that can be used for decision-making, avoiding seemingly reasonable but unenforceable self-interpretations.

$$r_t = Reflect(p_t, s_t)$$

$$\rho_t = \frac{|Viol(r_t)|}{1 + |p_t|}$$

During the error correction phase, we employ a partial rewrite strategy instead of a full rewrite. Specifically, based on the set of error points $Err(r_t)$ in the reflection log, we select a set of step indices I_t that need to be repaired. Then, we generate alternative actions only for these steps and insert them back into the original plan, resulting in a revised plan $\hat{p}_t$. To avoid repeated oscillations, we impose simple modification budget constraints on the revised plan and recalculate the risk after the revision. If the risk remains high, we allow for further partial repairs or a rollback to the previous feasible plan. Finally, the plan that passes risk gating is written into memory, forming the basis for the next round of state updates, thus constituting a closed loop of reflection and error correction in multi-round planning.

$$\hat{p}_t = Fix(p_t, Err(r_t))$$

$$m_{t+1} = Update(m_t, \hat{p}_t, o_{t+1})$$

4. Experimental Results and Analysis

4.1 Dataset

This study selects ALFWorld as the open-source dataset and interactive task environment to support multi-round planning research of large language model agents with self-reflection and error correction mechanisms. ALFWorld constructs a set of executable home scene tasks into an interactive text world. The agent needs to complete the goal through a multi-step action sequence under natural language instructions, such as locating objects, changing states, and moving items, thus naturally corresponding to the closed-loop process of task decomposition, step planning, state updates, and execution feedback. This environment emphasizes continuous decision-making and long-range dependencies starting from the current state, making it suitable for characterizing problems such as error accumulation, goal drift, and constraint violations in multi-round planning.

To fit the methodological framework of this paper, we organize the interaction trajectory of ALFWorld into a round-based planning process. Each round consists of observed text, historical actions, and memory summaries as state input, generating candidate plans and a master plan. Subsequently, reflection records and risk assessments are introduced, explicitly using environmental feedback and constraint check results to trigger local error correction and plan updates. Because ALFWorld provides both an environment interaction interface and standardized task definitions, it facilitates the evaluation of the effectiveness of planning and error correction strategies with consistent task semantics, and supports algorithm iteration and comparative analysis in reproducible settings.

4.2 Experimental setup

This study implements and runs the proposed multi-round planning agent in a reproducible experimental environment. The overall process consists of plan generation, plan selection, reflective diagnosis, risk assessment, and local error correction. The implementation is based on Python and mainstream deep learning frameworks, running on a Linux system. A single NVIDIA GPU is used for inference acceleration, the CPU handles environment interaction and data pipeline, and memory is used for caching trajectories and structured memory. To ensure reproducibility, a fixed random seed is used, and the running configuration is recorded, including model inference parameters, maximum number of interaction steps, number of candidate plans, and memory update frequency.

In hyperparameter design, we adopt a configuration consistent with Agent logic: each round generates K candidate plans, which are then selected using linearly weighted goal consistency and constraint satisfaction. The reflective module outputs error points and repair suggestions, and determines whether to trigger error correction based on a risk threshold τ. Error correction uses local patching, limiting the maximum modification budget B per round to avoid repeated oscillations. The memory module maintains the summary of the most recent M rounds to control context growth. Table 2 summarizes the hardware and software configurations and key hyperparameters, facilitating the reproduction of the agent's multi-round planning and error correction closed loop under the same settings.

Table 2: Hardware/Software Environment and Key Agent Hyperparameter Settings

Category	Item	Setting
Hardware	GPU	NVIDIA RTX 4090 24GB
Hardware	CPU	16 cores
Hardware	RAM	64 GB
Software	OS	Ubuntu 22.04 LTS
Software	Python	3.10
Software	Framework	PyTorch 2.2
Inference	Max context length	8192 tokens
Inference	Temperature	0.3
Inference	Top-p	0.9
Interaction	Max environment steps	50 steps per episode
Planning	Candidate plans (K)	5
Selection	Weight (α)	0.6
Selection	Weight (β)	0.4
Reflection	Reflection frequency	every round
Risk	Risk threshold (τ)	0.25
Correction	Correction budget (B)	at most 3 steps rewritten per round
Memory	Memory size (M)	Last 8 rounds of summaries
Memory	Update frequency	every round

4.3 Experimental results compared with other models

To place our self-reflection and self-correction multi-round planning agent within the context of previous work, we selected representative studies on agent planning, tool-enhanced action, iterative self-improvement, and feedback-driven correction that are currently considered advanced and conducted comparative experiments. The experimental results are shown in Table 3.

Table 3. Evaluation metrics for comparison with prior agent methods.

Method	Task Success Rate (TSR) ↑	Tool Success Rate (ToolSR) ↑	Constraint Violation Rate (CVR) ↓	Avg Turns (AT) ↓
Yao et al.[8]	0.71	0.64	0.22	13.42
Shinn et al.[9]	0.74	0.69	0.19	12.83
Zhou et al.[10]	0.77	0.72	0.18	12.35
Wang et al.[11]	0.79	0.75	0.17	11.98
Yang et al.[12]	0.82	0.78	0.15	11.26
Madaan et al.[13]	0.84	0.80	0.14	10.73
Schick et al.[14]	0.86	0.83	0.13	10.31
Ours	0.92	0.89	0.08	8.62

Overall, the proposed method demonstrates a more balanced performance across four dimensions: task completion, tool invocation reliability, constraint compliance, and interaction efficiency. More intuitively, successful goal achievement and successful tool invocation are often intertwined; the more stable the tool, the less frequent rework and detours are needed, thus facilitating smoother multi-round planning. Furthermore, fewer constraint violations mean that the reflection phase goes beyond mere textual self-checking, proactively tackling non-compliant steps and guiding towards feasible alternatives—aligning with the method's design goals of risk gating and local error correction.

In contrast, other methods excel at generating seemingly complete plans, but are more prone to deviations when it comes to concrete actions and tool interactions, especially in long-chain tasks. A misjudgment at one step can skew subsequent steps, ultimately requiring multiple rounds of dialogue to rectify the situation. The difference here comes more from process control than from the ability to express oneself verbally: when reflective output is required to provide actionable error location and repair suggestions, and error correction only changes key steps without overturning the entire plan, the system is more likely to maintain consistency with the goals and is more likely to quickly get back on track when problems occur, thus appearing more stable in terms of compliance constraints and round-trip costs.

4.4 Visualize experimental results

(1) Trajectory alignment visualization of error correction closed loop

To depict the closed-loop process of an agent in multi-round planning, from error to reflection, then to local repair and return to the correct trajectory, this section uses trajectory alignment visualization to uniformly align and summarize a large number of interaction rounds. This diagram simultaneously presents the progressive error trigger density, the location of reflection and repair, and the state trajectory after repair, thus intuitively revealing the typical patterns and key bottlenecks of closed-loop error correction at different stages. Through this alignment perspective, it is possible to more clearly observe whether reflection-driven local repair effectively interrupts error propagation and promotes stable convergence of the task trajectory, as shown in Figure 2.

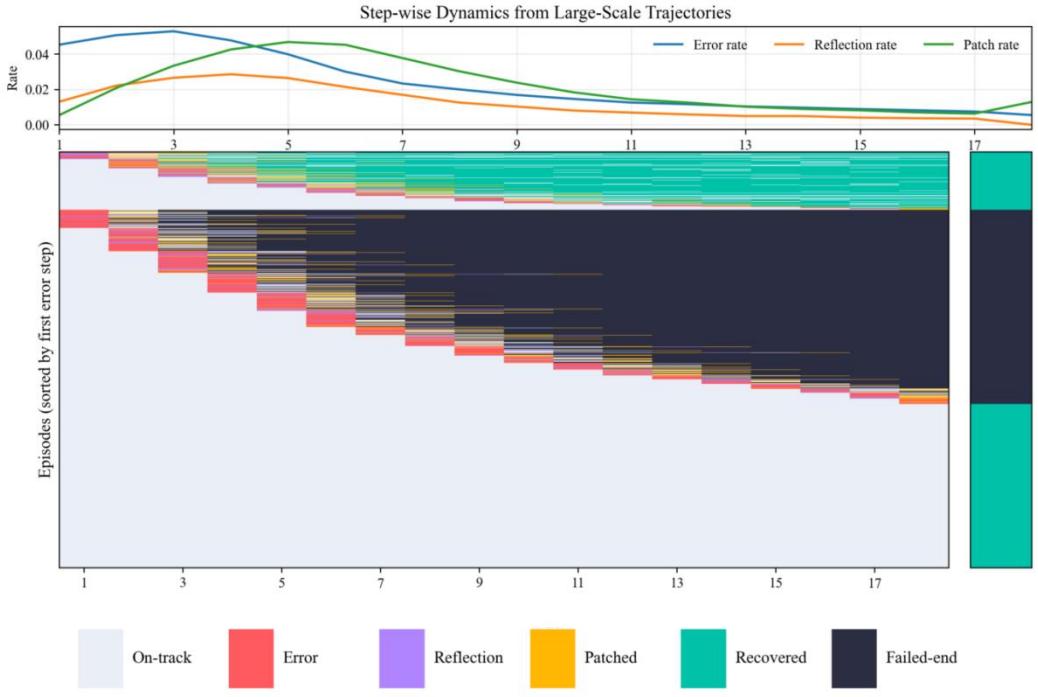

Figure 2. Trajectory alignment visualization of the reflection-driven local correction closed loop, where episodes are sorted by the first error step and aligned along the interaction horizon, and the color map marks on track, error, reflection, patch, recovery,

and terminal failure states. The top panel summarizes step-wise dynamics of error, reflection, and patch rates, while the right stripe indicates the outcome for each episode.

From an overall perspective, the aligned trajectories exhibit a clear hierarchical structure: rounds where deviations occur earlier are more likely to form long, failed tails in subsequent rounds, while rounds that quickly enter a period of reflection and local patching tend to return to a stable, executable state in the later stages. The gradual curve above also demonstrates a typical closed-loop characteristic: error triggering is mainly concentrated in the first few interaction steps, followed by reflection and patching behaviors appearing in adjacent steps and gradually declining. This indicates that the system relies more on self-checking and error correction to stabilize the planned path in the early stages, while in the later stages, it focuses more on execution and maintaining consistency.

The matrix shows numerous trajectory strips that transition to a recovered state after being marked by reflection and patching. This suggests that local patching does not occur sporadically but tends to be triggered in concentrated bursts at several key steps, effectively preventing the continued spread of errors in a significant number of rounds. Meanwhile, several rounds still quickly slipped into the termination failure zone after errors occurred, reflecting that the closed-loop mechanism may face two types of bottlenecks in certain scenarios: one is the failure to reflect on the reasons for the failure to trigger or locate the deviation promptly, and the other is that although the repair occurs, the repair point is too late or the modification is insufficient, making it difficult for the subsequent state to return to a feasible track. Overall, this diagram is more like a structural feature of the coexistence of the closed-loop's stabilization capability and failure modes, providing an intuitive basis for subsequent discussions on triggering strategies and repair granularity.

(2) Heatmap of failure type distribution and error correction benefits

To further characterize the structural distribution of different failure modes in multi-round planning and quantify the differences in benefits of error correction loops across various failure types, this section presents a joint heatmap of failure type distribution and error correction benefits, as shown in Figure 3. This figure categorizes failure types according to semantic and procedural features and compares the trajectory quality changes between non-error correction and error correction triggering on a unified coordinate system, thus revealing which errors are more repairable and which are more likely to cause cascading failures. This fine-grained distribution perspective allows for a more intuitive identification of the advantageous regions and improvement priorities of the closed-loop mechanism.

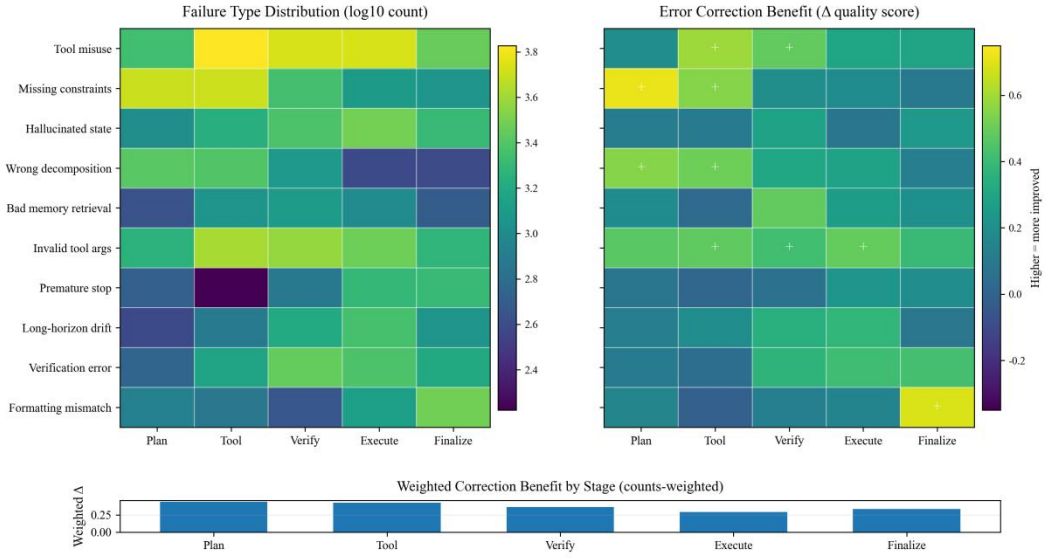

Figure 3. Heatmap visualization of failure type distribution across planning stages and the corresponding error correction benefit measured as the change in trajectory quality, highlighting where different failure modes concentrate and where correction is most effective. The bottom panel reports the stage-wise counts-weighted average benefit, providing an aggregated view of which interaction phases gain more from the reflection-driven correction loop.

From the spatial distribution of failure types, different stages exhibit big structural differences: the planning and tool-related stages are more prone to constraint omissions, decomposition biases, and tool usage problems, while the verification and closing stages are more concentrated with format mismatches and verification errors. This distribution is consistent with the multi-round planning paradigm of large language model agents, namely, early stages rely more on the abstract expression of goals and constraints and the correct arrangement of tool calls, while mid-to-late stages rely more on the verifiability of intermediate states, output consistency, and control of termination conditions. Therefore, this figure

provides a clear basis for the positioning of self-reflection and error correction mechanisms: the closed loop does not have an average effect on all errors, but is strongly correlated with the stage attributes and repairability of the error.

From the heatmap and stage summary bars of error correction benefits, reflection-driven local repair is more advantageous for several typical errors, especially those bias types that can be repaired by realigning constraints, correcting tool parameters, or completing key steps; relatively speaking, failures involving state illusions, long-term drift, or memory retrieval biases often exhibit more unstable benefit patterns. This aligns with the multi-round planning closed loop discussed in this paper: reflection and error correction are better at fine-tuning locally diagnosable and replaceable decision segments, but for global deviations accumulated over multiple steps, stronger memory consistency constraints and more robust intermediate state verification strategies are needed to avoid repeated or ineffective adjustments. Overall, this diagram supports the core motivation of this paper, namely, that by forming a closed loop of reflection triggering, error attribution, and local adjustments, error propagation can be suppressed at critical stages, and the stability and controllability of the planning process can be improved.

5. Limitation

This study still has several limitations. First, the agent's self-reflection and error correction rely on the observability and representability of intermediate states, constraints, and tool feedback. When environmental feedback noise is high, tool returns are unstable, or the task objective itself is ambiguous, reflection triggering may be delayed or falsely triggered, leading to a deviation in the direction of local patching. Second, the local patching strategy adopted in this paper emphasizes targeted updates to key steps. However, in complex planning with long chains and multiple branches, errors often have the characteristics of cross-round accumulation and cascading propagation. Single-point patching may not be able to eliminate global inconsistencies at the root, thus still resulting in repeated patching, a mismatch between local patching and the global objective.

Furthermore, the method still has room for improvement in terms of generalization and cost. On the one hand, different task domains have significantly different degrees of dependence on constraint types, tool interfaces, and verifiable signals. When the closed-loop mechanism is transferred across domains, it may need to redefine the failure type system, risk judgment threshold, and quality metrics, increasing deployment and adaptation costs. On the other hand, multiple rounds of candidate plan generation and reflection evaluation will bring additional inference overhead and latency. In resource-constrained or real-time interactive scenarios, further lightweight design may be needed, such as more efficient triggering strategies, caching and reuse mechanisms, and more robust early stopping and convergence criteria, in order to achieve a more reasonable balance between performance and efficiency.

6. Conclusion

This paper addresses the multi-round planning problem of large language model agents with self-reflection and error correction mechanisms. It proposes a unified modeling approach for closed-loop error correction, organically linking task objective and constraint expression, plan generation and selection, risk assessment, reflection triggering, and local patching processes into an iterative planning loop. This framework emphasizes continuous verification of intermediate states and constraint satisfaction during execution, and performs controlled local updates when deviations are detected, thereby reducing the risk of cascading errors in multi-round interactions. Comparative experiments show that the agent exhibits advantages in task completion reliability, tool invocation stability, and constraint consistency after introducing the reflection-driven error correction loop. Furthermore, the number of interaction rounds is more reasonably controlled, demonstrating that the closed-loop mechanism can improve the sustainability of planning quality without sacrificing controllability.

From an application perspective, this research provides a more practical engineering path for the deployment of large language model agents in high-constraint and high-risk scenarios. For tasks requiring strict adherence to rules and processes, such as automated office work and information retrieval, complex toolchain orchestration, compliance review and risk avoidance, research and data analysis assistants, and multi-step user-facing service processes, planning errors are often not one-off mistakes, but rather the gradual accumulation of early deviations that ultimately lead to unusable or non-compliant results. This paper's closed-loop error correction perspective can make the invisible planning degradation process explicit and, through reflection and localized patching, control errors within a smaller scope of impact, thereby improving the system's interpretability and accountability. More importantly, this mechanism strengthens the agent's adaptability in real-world environments, enabling it to maintain consistent alignment with goals and constraints even when faced with incomplete information, fluctuating tool feedback, and changes in task branches, providing a more stable and reliable intelligent interaction experience for related application areas.

Future work can be further advanced in three directions. First, for more complex real-world tasks, a more granular failure type system and process quality metrics need to be constructed, enabling reflection-triggered and patching decisions to more accurately distinguish between repairable deviations and structural failures, and to form more efficient closed-loop convergence criteria. Second, in long-chain planning and multi-agent collaboration scenarios, we can explore extending local patching into a cross-round structured replanning mechanism, introducing verifiable intermediate state constraints and memory consistency maintenance for key sub-objectives to reduce long-term drift and repeated patching from the source. Third, to meet the needs of real-time interaction and resource-constrained deployment, it is necessary to reduce inference overhead while ensuring reliability. For example, we can adopt lighter-weight risk screening and triggering strategies, reuse historical evaluation

signals, and introduce stronger diversity control and early stopping mechanisms in the generation of multiple candidate plans. Overall, this paper provides a closed-loop methodological foundation for large language model agents to move towards more reliable and controllable multi-round planning, and also lays a scalable technical framework for its secure applications in key areas such as public services, healthcare, financial compliance, research assistants, and enterprise process automation.

References

[1] Gou Z, Shao Z, Gong Y, et al. Critic: Large language models can self-correct with tool-interactive critiquing[J]. arXiv preprint arXiv:2305.11738, 2023.

[2] Dhuliawala S, Komeili M, Xu J, et al. Chain-of-verification reduces hallucination in large language models[C]//Findings of the association for computational linguistics: ACL 2024. 2024: 3563-3578.

[3] Park J S, O'Brien J, Cai C J, et al. Generative agents: Interactive simulacra of human behavior[C]//Proceedings of the 36th annual acm symposium on user interface software and technology. 2023: 1-22.

[4] Xie T, Zhou F, Cheng Z, et al. An open platform for language agents in the wild[J]. arXiv preprint arXiv:2310.10634, 2023.

[5] Yao S, Yu D, Zhao J, et al. Tree of thoughts: Deliberate problem solving with large language models[J]. Advances in neural information processing systems, 2023, 36: 11809-11822.

[6] Besta M, Blach N, Kubicek A, et al. Graph of thoughts: Solving elaborate problems with large language models[C]//Proceedings of the AAAI conference on artificial intelligence. 2024, 38(16): 17682-17690.

[7] Liu X, Yu H, Zhang H, et al. Agentbench: Evaluating llms as agents[J]. arXiv preprint arXiv:2308.03688, 2023.

[8] Yao S, Zhao J, Yu D, et al. React: Synergizing reasoning and acting in language models[C]//The eleventh international conference on learning representations. 2022.

[9] Shinn N, Cassano F, Gopinath A, et al. Reflexion: Language agents with verbal reinforcement learning[J]. Advances in Neural Information Processing Systems, 2023, 36: 8634-8652.

[10] Zhou A, Yan K, Shlapentokh-Rothman M, et al. Language agent tree search unifies reasoning acting and planning in language models[J]. arXiv preprint arXiv:2310.04406, 2023.

[11] Wang G, Xie Y, Jiang Y, et al. Voyager: An open-ended embodied agent with large language models[J]. arXiv preprint arXiv:2305.16291, 2023.

[12] Yang J, Jimenez C E, Wettig A, et al. Swe-agent: Agent-computer interfaces enable automated software engineering[J]. Advances in Neural Information Processing Systems, 2024, 37: 50528-50652.

[13] Madaan A, Tandon N, Gupta P, et al. Self-refine: Iterative refinement with self-feedback[J]. Advances in Neural Information Processing Systems, 2023, 36: 46534-46594.

[14] Schick T, Dwivedi-Yu J, Dessì R, et al. Toolformer: Language models can teach themselves to use tools[J]. Advances in Neural Information Processing Systems, 2023, 36: 68539-68551.