Artificial Intelligence and Computing Innovations

Intelligence Science Publishing

https://www.ispublishing.com

ISSN: 3070-4944

Vo.4, No.3, 2024

Task Graph Modeling and Constraint-Aware Scheduling for Reliable Large Language Model Agent Execution

Hengguang Cui

Brown University, Providence, USA

hengguang_cui@alumni.brown.edu

Abstract: This paper addresses the instability of large language model agents under real-world constraints, proposing an integrated approach of robust task decomposition and constraint-aware scheduling to improve the controllability, compliance, and stability of long-link interactive tasks. The method first structures high-level instructions into a task graph, explicitly modeling subtask dependencies, resource sharing, and executable conditions. Verifiable pre-checks, alternative paths, and fallback actions are configured for key steps to reduce error accumulation caused by incomplete information and environmental changes. Subsequently, during the scheduling phase, latency and risk are incorporated into a unified priority decision, dynamically selecting the execution order and allocating resources based on critical path identification and failure cost assessment. When tools become unavailable, constraints are triggered, or execution deviations occur, rapid recovery is achieved through local reordering and path switching, avoiding the high overhead of global rework. Comparative experimental results show that this method exhibits superior overall performance in key agent metrics such as task completion, tool invocation reliability, constraint violation control, interaction efficiency, and recovery capability. It can maintain a more stable end-to-end execution process under a limited budget and uncertain environments, providing a feasible agent execution paradigm for real-world business process automation and multi-tool collaboration scenarios.

Keywords: Task graph representation; constraint-aware scheduling; local reordering mechanism; multi-tool collaboration.

1. Introduction

As large language models evolve from text generation to executable agents, more and more systems are undertaking complex tasks across tools, pages, and data sources. In the real world, tasks are often not single-step question-and-answer sessions, but rather long-linked interactive processes comprised of goal constraints, external dependencies, and process feedback. These processes include various operations such as retrieval,

invocation, filling, submission, and verification, and are constrained by budget, latency, permission availability, and dynamic environmental changes. Under these conditions, the core challenge for agents is no longer simply generating the correct next sentence, but rather their ability to continuously advance and stably complete the entire process from task understanding to final delivery amidst uncertain information and changing environments[1,2].

Failures in real-world deployment scenarios often exhibit systemic characteristics: incomplete task descriptions lead to missing implicit preconditions; unstable tool interfaces cause mid-process blockages; changes in page state render previous plans ineffective; and retrieval noise causes key judgments to deviate from the target, thus triggering error accumulation and rework cycles. Traditional static planning or template-based decomposition is prone to imbalances in granularity; excessive granularity leads to invocation explosions and increased costs, while excessive coarseness makes error localization difficult and amplifies it subsequently. Meanwhile, without explicit modeling of constraints and critical paths, the scheduling process may consume resources on low-return steps, leading to the inability to close the loop under limited steps and budget. Therefore, robust task decomposition and recoverable scheduling oriented towards real-world constraints are crucial foundations for improving the reliability and engineering deliverability of intelligent agents[3].

From an application perspective, robust decomposition and scheduling capabilities directly determine whether an intelligent agent can enter high-value scenarios and operate stably in the long term. In enterprise process automation, intelligent agents need to coordinate multiple systems under authorization and audit requirements; any mistake can trigger compliance and cost risks. In public service and digital government scenarios, task chains involve multiple rules and dynamic information, requiring the system to maintain consistent execution logic even when the environment changes. In high-time-sensitive fields such as medical collaboration and emergency response, tasks are often subject to strict latency and resource constraints; the system needs to make robust decisions and possess rapid recovery capabilities within a limited number of interactions[4]. Therefore, constructing robust task decomposition and scheduling strategies for intelligent agents with large language models oriented towards real-world constraints not only has methodological value but also significant practical implications.

The main contributions of this paper include:

(1) Proposing a structured task representation oriented towards real-world constraints, which explicitly defines goal dependencies and feasibility conditions, supporting recoverable task decomposition and local correction.

(2) Proposing a constraint-aware scheduling and rescheduling strategy, which incorporates budget delay risk and critical path into a unified decision-making framework, enabling priority reordering and alternative path switching during execution.

(3) Providing an end-to-end agent execution closed-loop design, linking the verifiability of the decomposition phase with the risk control of the scheduling phase, reducing error accumulation in long-link interactions and improving overall delivery stability.

2. Background

Agent systems designed for real-world constraints are evolving from single-turn question answering toward multi-step planning, reasoning, and tool invocation paradigms. Early studies have demonstrated the feasibility of grounding language instructions into actionable behaviors and real-world interactions [5], recent works further extend these capabilities to multimodal perception tasks [6]. In particular, frameworks such as ReAct integrate reasoning and acting in a unified loop [7], and hierarchical planning architectures enable agents to handle complex, long-horizon tasks more effectively [8]. Meanwhile, reflection-based mechanisms and verbal reinforcement learning further enhance agent adaptability in dynamic environments [9]. These

advances have also been applied in domain-specific scenarios such as financial anomaly detection and collaborative decision-making systems [10-11].

Despite these developments, real-world deployments remain constrained by factors such as incomplete task specifications, dynamic environmental changes, unstable tool availability, and strict budget and latency limitations. Privacy-preserving constraints and model adaptation requirements further complicate system design [12], while advanced reasoning paradigms such as Tree-of-Thoughts introduce additional computational overhead and planning complexity [13]. Transfer learning techniques partially alleviate data scarcity issues but may still suffer from instability under distribution shifts [14]. Unlike idealized benchmarks, real-world tasks typically involve implicit constraints, long dependency chains, and higher risks of error propagation. Empirical studies in realistic environments, such as web-based interactive benchmarks, reveal that retrieval noise, non-deterministic tool responses, and dynamic permission or resource changes can easily invalidate static execution plans [15]. Additionally, structural biases and instability in model fine-tuning further amplify these challenges [16].

Consequently, robust task decomposition and adaptive scheduling become critical for maintaining execution reliability and improving success rates. Existing approaches often rely on predefined templates or fixed scoring mechanisms, lacking flexibility in balancing correctness, cost, latency, and risk. Although prompting strategies such as least-to-most reasoning improve decomposition quality [17], they still fall short in handling uncertainty and dynamic replanning in real-world settings. Furthermore, current systems lack coordinated optimization between decomposition and scheduling, frequently resulting in excessive task fragmentation or coarse-grained planning errors. When execution deviations occur, conventional replanning strategies tend to focus on local corrections rather than maintaining global consistency or dynamically adjusting task priorities.

Therefore, it is necessary to develop a robust task decomposition and scheduling framework oriented toward real-world constraints. Such a framework should support recoverable decomposition strategies, controllable scheduling decisions, and interpretable failure handling under uncertainty, thereby significantly improving the stability, efficiency, and reliability of intelligent agent systems in practical deployment scenarios.

3. Method

3.1 Robust task decomposition modeling

Real-world tasks typically involve objectives, constraints, and external dependencies, and plans may fail during execution due to incomplete information and environmental changes. This method first abstracts the input task into a set of executable subtasks and explicitly records the prerequisite relationships and shared resource dependencies between subtasks to avoid omissions and rework caused by template-based decomposition based solely on semantic similarity. To ensure the decomposition is recoverable in dynamic environments, the decomposition stage not only produces a set of subtasks but also includes necessary prerequisites, alternative paths, and fallback actions for each subtask, allowing progress to continue even when tools are unavailable, permissions are insufficient, or retrieval noise is high. Furthermore, to avoid excessive granularity leading to call bloat or excessive coarseness leading to unlocalized errors, the decomposition adopts a controllable granularity strategy, deciding whether to further subdivide based on the verifiability of subtasks and the cost of external interactions, and prioritizing key steps in a locally verifiable and locally redotable manner. Overall, this modeling uses a structured representation to uniformly describe tasks, dependencies, and constraints, providing a computable basis for subsequent scheduling decisions and reducing the risk of overall collapse caused by error accumulation in multi-step links. Its modular architecture is shown in Figure 1.

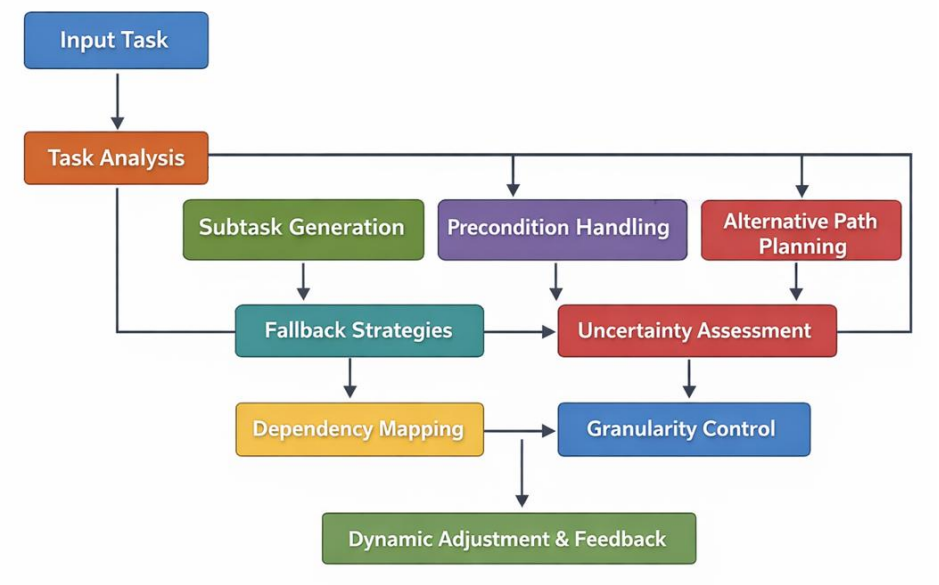

Figure 1. Robust Task Decomposition Modeling Architecture

To formally represent the decomposition results, the set of subtasks and the directed edges of their dependencies are defined as follows:

$$G = (V, E) \tag{1}$$

Where V represents the set of subtask nodes, and E represents the prerequisite dependencies. Each subtask $i \in V$ is associated with a set of constraint predicates C_i. If all predicates are satisfied, the subtask is considered executable in the current environment. A simple feasibility indicator function is given below:

$$\phi_i = \begin{cases} 1, & C_i \text{ satisfies} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

This allows the decomposition phase to explicitly distinguish between non-executable and executable steps, and to transform non-executable steps into supplementary subtasks that first unlock constraints.

To enhance robustness, a lightweight uncertainty score is introduced to measure the sensitivity of subtasks to external information and tool stability. For each subtask i, a success probability estimate $p_i \in [0,1]$is maintained, and it is calculated using:

$$u_i = 1 - p_i \tag{3}$$

This indicates the degree of uncertainty. During decomposition, steps with high uncertainty are prioritized and organized into shorter, more verifiable links. Necessary pre-checks and result verifications are inserted before and after these links, allowing failures to be exposed locally rather than erupting at the end.

In the execution loop, decomposition is not completed all at once, but allows for local adjustments based on observed feedback. If subtask i introduces new constraints or discovers missing preconditions, necessary nodes and dependent edges are added to the graph, prioritizing the preservation of completed parts and replacing only the smallest affected subgraph. This strategy ensures the stability and recoverability of the decomposition structure, while providing the scheduling module with a continuously updated set of executables and risk labels, thus supporting subsequent dynamic orchestration under budget and latency constraints.

3.2 Constraint-aware scheduling and rescheduling strategies

After obtaining the structured task graph, the scheduling goal is to select the appropriate execution order and resource allocation under real-world constraints, enabling the system to maintain higher task completion stability in environments with limited cost, limited latency, and uncertainty. Its modular architecture is shown in Figure 2.

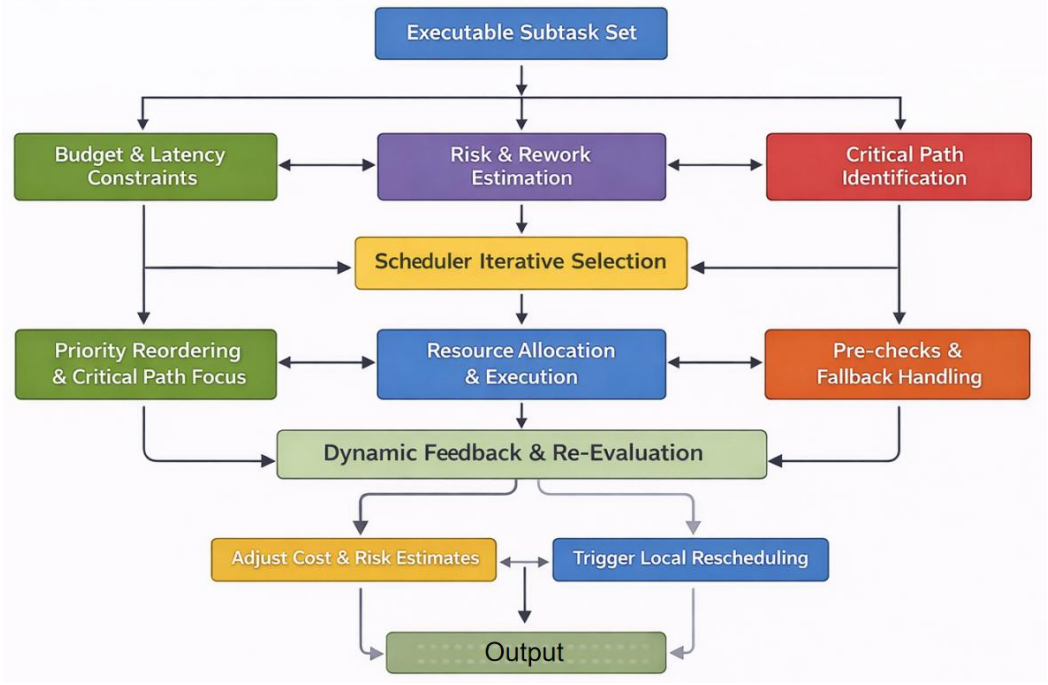

Figure 2. Constraint-aware scheduling and rescheduling strategy module architecture

The method views scheduling as an iterative selection process of a set of executable subtasks: at each step, one or a group of tasks is selected from the current executable set for execution, and the state and dependencies are updated after receiving external feedback. To avoid pursuing only the shortest path while ignoring risk, scheduling considers three factors simultaneously: budget constraints on cost and latency, failure risk and rework cost, and the position of the critical path in the task chain. Specifically, tasks on critical dependency paths, tasks that can significantly reduce subsequent uncertainty, and tasks that can unlock multiple subsequent tasks will receive increased priority; while high-risk and irreplaceable tasks will be scheduled for execution after sufficient pre-checks and fallback paths. Simultaneously, the scheduling incorporates a lightweight rescheduling mechanism. When tools become unavailable, data sources change, or constraints are triggered, instead of a global overhaul, priority reordering and alternative path switching are performed on the affected local dependency areas, thereby reducing interruption costs and maintaining overall progress.

To characterize multi-objective trade-offs, scheduling cost is defined as the weighted sum of cost, delay, and risk. For each subtask i, let there be estimated cost c_i, estimated delay t_i, and uncertainty u_i. Then the overall cost can be written as:

$$J = \sum_{i \in V} (\alpha c_i + \beta t_i + \gamma u_i) \tag{4}$$

Here, $\alpha, \beta, \gamma \geq 0$represents the weight used to express the preference between the resource side and the stability side. The basic principle of scheduling is to minimize Jwhile satisfying feasibility and dependency constraints, and to dynamically adjust the estimate of c_i, t_i, u_ibased on feedback during execution.

The executable set at the current moment is defined as:

$$A = \{i \in V | \phi_i = 1 \wedge \text{prerequisite tasks completed}\} \quad (5)$$

The scheduler calculates task priorities on set A and selects the highest priority task for execution. To maintain simplicity and interpretability, a linear priority scoring method is provided:

$$s_i = \omega_1 g_i - \omega_2 c_i - \omega_3 t_i - \omega_4 u_i \quad (6)$$

Where g_i represents the progress benefits brought by the task, such as the number of subsequent tasks unlocked or the contribution to the critical path, and $\omega_1, \ldots, \omega_4 \geq 0$ is the weight. This score takes into account both progress efficiency and resource risk, and facilitates the switching of strategy preferences through weights in different application scenarios.

Rescheduling is completed based on a feedback-driven, lightweight update. After executing subtask i and observing the result r_i, the success probability estimate is smoothly updated:

$$p_i \leftarrow (1-\eta)p_i + \eta r_i \quad (7)$$

Where $r_i \in \{0,1\}$ represents success or failure, and $\eta \in (0,1]$ is the update rate. If failure leads to a change in constraints, a local rearrangement is triggered: the uncertainty of the affected subtasks is increased, and alternative paths are prioritized or unlocking tasks that can restore feasibility are executed first, thereby achieving rapid adaptation and robust progress to dynamic constraints without changing the overall structure.

4. Experimental Results and Analysis

4.1 Dataset

This study builds and evaluates agent task decomposition and scheduling strategies oriented towards real-world constraints on the open-source WebArena benchmark. WebArena provides a self-hosted, realistic webpage interaction environment that requires agents to translate high-level natural language instructions into specific webpage operation sequences. During execution, agents must complete multi-step interactions such as cross-page retrieval, filling in, clicking, and submitting, thus naturally covering key capabilities such as multi-step planning, tool and resource access, and execution feedback-driven iterative decision-making. This environment consists of multiple fully functional websites, with data and interaction flows closely resembling real-world application scenarios. This ensures that task decomposition and scheduling not only consider dependencies but also handle uncertainties and interruption risks in the execution chain.

WebArena's task settings include several common real-world domains (e.g., e-commerce, forum discussions, collaborative development, and content management), and simulate external dependencies through built-in tool sites and knowledge resource sites. This forces agents to perform tool selection, information location, and multi-source coordination when solving problems, rather than relying solely on single-step text reasoning. Since this benchmark is based on a reproducible set of environments and tasks, it is suitable for verifying constraint-aware scheduling and local rescheduling mechanisms: when the page state changes, the available paths are limited, or the prerequisites need to be unlocked first, the system needs to dynamically adjust the executable set and reorder the priorities to maintain the overall task progress and stable completion.

4.2 Experimental setup

This study's experiments were conducted in a reproducible web-based interactive agent environment. The agent employed a unified task decomposition representation and constraint-aware scheduling loop: each task was first structured into a subtask graph, and during execution, iterative selection and resource allocation were performed based on feasibility checks, risk estimation, and critical path identification. When tools became unavailable, page states changed, or constraints were triggered, local rescheduling was initiated to rearrange priorities and switch to alternative paths. To ensure fairness and comparability, all comparison

methods shared the same environment version, maximum step limit, and the same set of tool call interfaces. A fixed random seed was used, and decision logs and execution feedback for each step were recorded for reproducibility and error attribution. A summary of the specific hardware and software environment and key hyperparameter settings is shown in Table 1.

Table 1 also presents the core hyperparameters on both the model and scheduling sides, including decoding temperature, maximum output length, tool call budget, rescheduling trigger threshold, and weight coefficients. Unless otherwise specified, all experiments used the same default settings; when sensitivity analysis was required, only the corresponding items in Table 1 were replaced, while other configurations remained unchanged, thus ensuring that performance differences primarily stemmed from the task decomposition and scheduling strategies themselves rather than external implementation details.

Table 1. Experimental environment and hyperparameter settings

Category	Item	Setting	Category	Item	Setting	Category	Item	Setting
Hardware	CPU	16 cores	Hardware	CPU	16 cores	Hardware	CPU	16 cores
Hardware	RAM	64 GB	Hardware	RAM	64 GB	Hardware	RAM	64 GB
Hardware	GPU	NVIDIA GPU	Hardware	GPU	NVIDIA GPU	Hardware	GPU	NVIDIA GPU
Software	OS	Ubuntu 22.04 LTS	Software	OS	Ubuntu 22.04 LTS	Software	OS	Ubuntu 22.04 LTS
Software	Python	3.10	Software	Python	3.10	Software	Python	3.10
Software	Deep learning	PyTorch 2.2	Software	Deep learning	PyTorch 2.2	Software	Deep learning	PyTorch 2.2
Software	Agent runtime	Playwright	Software	Agent runtime	Playwright	Software	Agent runtime	Playwright
LLM decoding	Temperature	0.2	LLM decoding	Temperature	0.2	LLM decoding	Temperature	0.2

4.3 Experimental Results and Analysis

To contextualize the proposed constraint-aware scheduling and rescheduling strategy for LLM agents, we summarize representative, closely related studies on long-horizon agent planning, robust task decomposition, and dynamic replanning under real-world constraints. The comparison is organized with agent-centric evaluation metrics that reflect task success, tool reliability, efficiency, and safety-recovery behavior, enabling a unified view of how different frameworks prioritize stability and controllability during multi-step execution. The experimental results are shown in Table 2.

Table 2. Experimental results compared with other models

Method	Task Success Rate	Tool Success Rate	Constraint Violation Rate	Avg Steps per Task	Avg Tool Calls	Avg Latency (s)	Avg Time (min)	Recovery Success Rate
Prasad et al.[18]	68.42	71.35	7.12	5.83	3.27	12.48	0.34	62.15
Yang et al.[19]	70.51	73.22	6.98	5.61	3.11	11.93	0.33	64.27
Ye et al.[20]	72.33	75.14	6.44	5.48	3.05	11.75	0.32	66.81
Chang et al.[21]	74.12	77.08	6.01	5.29	2.96	11.42	0.31	68.33
Geng et al.[22]	75.27	78.66	5.74	5.18	2.91	11.21	0.30	69.44
Yoran et al.[23]	76.84	79.55	5.51	5.07	2.88	11.03	0.29	70.19
Zeng et al.[24]	77.93	80.12	5.39	4.98	2.85	10.94	0.28	71.04
Ours	85.47	88.92	3.14	4.21	2.33	8.74	0.21	82.66

From an overall comparison, the proposed method demonstrates superior core completion capabilities and execution stability. Compared to baseline solutions primarily based on decomposition or planning, the benefits of this method extend beyond simply ensuring task completion; they also enhance the robustness of the execution chain: tool calls are more reliable, constraint triggers are fewer, and it's easier to pull the process back to a sustainable state after failures. This aligns with the approach emphasized in the previous section on methods, linking the recoverability of task decomposition with constraint awareness during the scheduling phase. This allows the system to avoid frequent restarts in uncertain environments, instead relying on localized adjustments to maintain overall progress, thereby exposing and mitigating errors earlier and locally.

Furthermore, from an efficiency and cost perspective, the improvement is not achieved at the expense of higher overhead. A common approach in comparative methods is to introduce more checks and redundant attempts to mitigate risk, ultimately leading to longer interaction paths, higher tool call frequencies, and greater time and cost burdens. The trend presented here demonstrates that this method maintains higher completion stability with less interaction overhead. The key reason is that the scheduling strategy prioritizes the execution of critical paths and unlocking steps, and promptly triggers local reordering and alternative path switching when infeasible or high-risk states are detected. This reduces ineffective exploration and repeated trial and error, and avoids the accumulation of uncertainty at the end of the process, making the overall process more compact and controllable.

To evaluate the impact of decoding randomness on the stability of the agent's decision-making chain, this paper examines the effect of temperature parameters on task execution behavior under different

values. This setting is used to characterize the impact of the generation strategy shifting from more deterministic to more divergent in real-world interactions on the overall completion process, and the experimental results are shown in Figure 3.

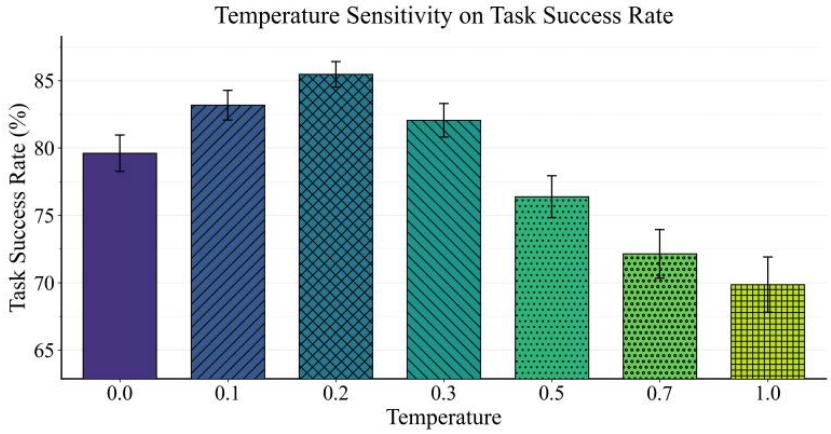

Figure 3. Sensitivity experiment of temperature parameters to task success rate

Temperature significantly impacts the stability of the agent's task completion, exhibiting an overall pattern of initial increase followed by a decrease. In the lower temperature range, generation becomes more deterministic, and the decision-making chain is more consistent, making it easier to continuously advance along feasible paths. As the temperature gradually increases from extremely low levels, the model gains a degree of exploratory power while maintaining control, helping to avoid local bottlenecks in multi-step interactions and thus completing the task more smoothly.

As the temperature continues to rise, randomness increases further, action selection becomes more divergent, and unnecessary trials and path drifts are easily introduced, leading to a more unstable execution process. Ultimately, this makes task completion more prone to interruption or deviation from the goal. This change aligns with the characteristics of real-world web page interaction tasks: on the one hand, moderate exploration is needed to cope with changes in page state and incomplete information; on the other hand, strong process consistency is relied upon to ensure long-term reliability is not compromised. Therefore, excessively low or high temperatures are not advantageous; a moderate temperature level is more reasonable, balancing stable progress with necessary exploration.

This paper also examines the impact of injecting noise of different intensities into the search results on the overall task execution behavior, and the experimental results are shown in Figure 4. This setting is used to simulate the situation in a real environment where the search results contain irrelevant or misleading content, thereby testing the anti-interference ability of the scheduling and rescheduling mechanism under uncertain input.

As the proportion of retrieval noise increases, the task success rate shows a continuous downward trend, and the decline is relatively smooth. This indicates that in this type of real-world webpage interaction task, the reliability of external retrieval information directly affects the stability of subsequent subtask selection and execution paths. When the returned content contains more irrelevant or misleading information, the agent is more likely to form intermediate judgments that deviate from the target early on, thus amplifying the deviation in subsequent steps and ultimately making it more difficult to close the complete chain. Compared to the generation randomness reflected by temperature sensitivity, retrieval noise is more like a continuous weakening of the input signal quality; its interference is usually more subtle and more difficult to eliminate with a single correction.

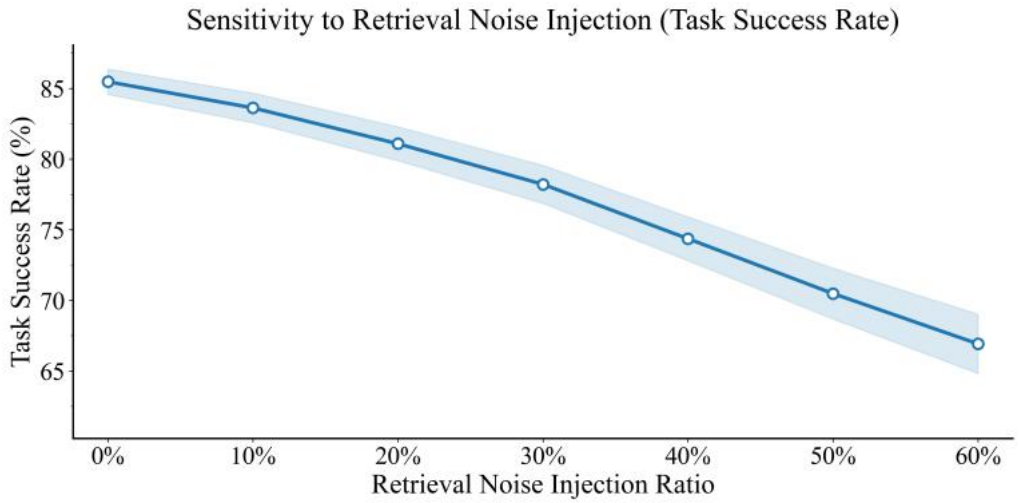

Figure 4. Sensitivity experiment of retrieval noise injection ratio on task success rate

This change also reflects that constraint-aware scheduling and rescheduling mechanisms require stronger input verification and risk control: when retrieval information becomes unreliable, the key is not to blindly increase exploration, but to trigger feasibility checks and result verification earlier, locally truncate potential erroneous paths, and reduce subsequent uncertainty by prioritizing the execution of steps that can clarify information or unlock key dependencies. Meanwhile, the banded areas in the figure reflect a tendency for more pronounced fluctuations when noise is higher, meaning that the system is more sensitive to occasional misleading events. Therefore, it is necessary to introduce delayed execution of high-risk steps and alternative path switching during the scheduling phase to avoid high-cost repeated trial and error under unstable inputs.

5. Conclusion

This framework uses a structured task graph as its core, explicitly defining objectives, dependencies, and feasibility conditions. During execution, it reduces the risk of error accumulation from multi-step inference and multi-tool invocation through recoverable decomposition strategies and dynamically feedback-driven local reordering mechanisms. Unlike methods that only emphasize single-step planning or static decomposition, this work focuses on common issues in real-world deployments, such as interruptions, drift, and rework. By linking the verifiability of the decomposition phase with risk control in the scheduling phase, the agent can maintain continuous progress and overall completion stability in uncertain environments.

Experimental comparisons show that the proposed method demonstrates a better comprehensive trade-off across multiple core dimensions of the agent, including task completion, tool reliability, constraint compliance, interaction efficiency, and recovery capability. This improvement manifests as a more stable execution chain and fewer ineffective explorations, indicating that structured dependency modeling and constraint-aware decision-making can effectively reduce the propagation of error paths while controlling failures locally, avoiding the high-cost cycle of global restart. Furthermore, the critical path priority and feasibility priority principles emphasized in this method enable agents to make more effective use of limited steps and limited call budgets in resource-constrained scenarios, focusing limited interaction opportunities on the steps that can best drive the task closure, thereby improving controllability and interpretability in real-world applications.

References

[1] W. Huang, P. Abbeel, D. Pathak and I. Mordatch, "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents," Proceedings of the International Conference on Machine Learning, pp. 9118-9147, 2022.

[2] 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.

[3] Karpas E, Abend O, Belinkov Y, et al. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning[J]. arXiv preprint arXiv:2205.00445, 2022.

[4] 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.

[5] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, A. Zeng et al., "Do as I can, not as I say: Grounding language in robotic affordances," arXiv preprint arXiv:2204.01691, 2022.

[6] J. Li, "LocateNet: Large Multimodal Models for Text-Guided Object Localization," 2024.

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," Proceedings of the International Conference on Learning Representations (ICLR), 2022.

[8] Y. Hu, "Autonomous Agent Architecture for Complex Tasks via Hierarchical Planning and Language Model Reasoning," 2024.

[9] N. Shinn, F. Cassano, A. Gopinath et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," Advances in Neural Information Processing Systems, vol. 36, pp. 8634-8652, 2023.

[10] Q. Gan, "Large Language Model Framework for Multi-Document Financial Anomaly Detection in Intelligent Auditing via Semantic Mapping and Risk Reasoning," 2024.

[11] C. Hua, "A Semantic-Prior-Guided AI Framework for Collaborative Environment Understanding and Robust Agent Decision Making," 2024.

[12] Y. Li, "Task-aware Differential Privacy and Modular Structural Perturbation for Secure Fine-Tuning of Large Language Models," 2024.

[13] S. Yao, D. Yu, J. Zhao et al., "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," Advances in Neural Information Processing Systems, vol. 36, pp. 11809-11822, 2023.

[14] Y. Deng, "Transfer Methods for Large Language Models in Low-Resource Text Generation Tasks," 2024.

[15] S. Yao, H. Chen, J. Yang et al., "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents," Advances in Neural Information Processing Systems, vol. 35, pp. 20744-20757, 2022.

[16] H. Liu, "Structural Regularization and Bias Mitigation in Low-Rank Fine-Tuning of LLMs," 2023.

[17] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang et al., "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," Proceedings of the International Conference on Learning Representations (ICLR), 2023.

[18] Prasad A, Koller A, Hartmann M, et al. Adapt: As-needed decomposition and planning with language models[C]//Findings of the Association for Computational Linguistics: NAACL 2024. 2024: 4226-4252.

[19] X. Zhang, Y. Deng, Z. Ren, S. K. Ng, and T. S. Chua, “Ask-before-plan: Proactive language agents for real-world planning,” in Findings of the Association for Computational Linguistics: EMNLP 2024, Nov. 2024, pp. 10836-10863.

[20] S. Qiao, R. Fang, N. Zhang, Y. Zhu, X. Chen, S. Deng, et al., “Agent planning with world knowledge model,” Advances in Neural Information Processing Systems, vol. 37, pp. 114843-114871, 2024.

[21] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, et al., “Understanding the planning of LLM agents: A survey,” arXiv preprint arXiv:2402.02716, 2024.

[22] M. Hu, P. Zhao, C. Xu, Q. Sun, J. G. Lou, Q. Lin, et al., “AgentGen: Enhancing planning abilities for large language model based agent via environment and task generation,” in Proc. 31st ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD), Jul. 2025, pp. 496-507.

[23] Yoran O, Amouyal S J, Malaviya C, et al. Assistantbench: Can web agents solve realistic and time-consuming tasks?[J]. arXiv preprint arXiv:2407.15711, 2024.

[24] A. Li, Y. Xie, S. Li, F. Tsung, B. Ding, and Y. Li, “Agent-oriented planning in multi-agent systems,” arXiv preprint arXiv:2410.02189, 2024.