Journal of Computer Technology and Software
ISSN:2998-2383
Vol. 4, No. 7, 2025

Safe Tool Use in LLM Agents via Reliability-Aware Planning and Conservative Action Gating

Zhiwei Luo[1*], Ya-Wen Hsu[2], Ruxsara Qeyum[2]
[1]University of Georgia, Athens, USA
[2]University of Southern California, Los Angeles, USA
*Corresponding author: Zhiwei Luo; eniacsimon@gmail.com

Abstract: This paper focuses on the task execution problem of large language model agents under tool uncertainty conditions, proposing a unified strategy framework of risk-aware planning and conservative execution to improve stability, controllability, and consistency in multi-tool collaborative scenarios. The method models the agent interaction process as a belief-based sequential decision-making process, using a belief update mechanism to integrate task state and tool observations, and employing an uncertainty metric to characterize the current cognitive confidence level. Simultaneously, it introduces tool reliability assessment to characterize the availability and consistency of tool invocation online, thus explicitly incorporating risk penalties into the planning objective and prompting the strategy to achieve an adjustable trade-off between benefits and risks. At the execution level, the framework employs a safety constraint and action gating mechanism, jointly using reliability thresholds and uncertainty thresholds as trigger conditions for tool invocation, and activating a backoff action when constraints are not met to limit error propagation and out-of-bounds operations. Comparative experiments show that this method exhibits a more balanced advantage in terms of accuracy, task completion, tool invocation success rate, and constraint violation control, verifying the effectiveness of incorporating risk modeling into planning and co-designing with conservative execution, providing an achievable technical path for the reliable deployment of tool-enhanced agents in complex toolchain environments.

Keywords: Risk perception decision-making; tool reliability modeling; security gating mechanisms; belief state reasoning

1. Introduction

In recent years, large language model-driven agents have been rapidly deployed in scenarios such as question answering, information verification, data analysis, and automated workflows. Their core capabilities have expanded from single-round generation to goal-oriented continuous decision-making and collaborative invocation of multiple tools[1]. Unlike traditional models that reason only within the text space, LLM agents need to perform actions in an external environment, such as selecting tools, organizing parameters, interpreting returned results, and updating plans accordingly, thus connecting language understanding with the action loop. Because this capability spans internal model reasoning and external system interaction, the reliability of the agent no longer depends solely on the quality of language generation but also heavily relies on the availability, stability, and verifiability of the toolchain. This makes tool uncertainty a key bottleneck affecting the security and efficiency of actual deployments[2].

Tool uncertainty is multi-source and insidious, including behavioral drift caused by interface timeouts, permission changes, and version upgrades, as well as noise and format inconsistencies in retrieval and database returns, and error propagation when multiple tools are combined[3]. More complexly, agents often make tool selections and sequential operations with incomplete information. Overconfidence in tool capabilities or output reliability can lead to a chain reaction of flawed planning, repeated calls, resource waste, and even risky operations. For high-impact tasks, a single deviation can be amplified into a systemic failure. Therefore, uncertainty must be treated as a first-class citizen in the planning process, with risk signals explicitly injected into the decision-making chain. This enables agents to maintain controllable, explainable, and auditable behavioral boundaries when facing an unstable external world[4].

To address this challenge, constructing risk-aware planning and conservative execution strategies oriented towards tool uncertainty is crucial[5]. On one hand, it helps improve the robustness of agents in real-world production environments, enabling them to proactively degrade, converge behavior, and maintain task continuity when tools fail, outputs conflict, or evidence is insufficient, thereby reducing the costs of unforeseen failures. On the other hand, risk awareness and conservative execution provide mechanisms to support safety and compliance. By quantifying uncertainty and imposing decision constraints, it achieves more transparent risk exposure, more consistent execution strategies, and clearer responsibility boundaries. Ultimately, this type of strategy will drive LLM Agents from usable to trustworthy, making them more suitable for long-term stable operation in complex

business processes and demanding scenarios, and providing a transferable methodological foundation for the design of intelligent systems for real-world interactions.

2. Related Work

Existing research primarily focuses on two paths to improve the reliable interaction and task completion capabilities of LLM agents. The first type of work concentrates on tool-enhanced reasoning and action frameworks. By combining capabilities such as external retrieval, code execution, database queries, and application interface calls into a unified decision loop, the model can perform plan generation, tool selection, result integration, and state updates across multiple tasks. Related methods typically improve stability through instruction following, task decomposition, reflective error correction, and memory mechanisms, and reduce execution bias caused by unstructured output through structured prompts, function call constraints, and execution trajectory recording[6]. The second type of work emphasizes improvements in planning strategies for complex environments, such as hierarchical planning and tree-search reasoning. These methods decompose long-term goals into executable sub-goals and adjust paths iteratively to reduce local optima and misoperations, while improving task completion rates and generalization capabilities under multi-tool dependencies.

Another important research direction revolves around the design of mechanisms for risk control and safe execution. Much work focuses on factual consistency, verifiability, and output reliability, introducing techniques such as self-checking and cross-validation, evidence alignment and conflict resolution, confidence estimation, and calibration to attempt to identify potential errors before or after decision generation. These techniques aim to reduce the probability of inappropriate actions through constraint decoding, rule filtering, or strategy optimization[7]. Simultaneously, some research focuses on execution-level protection, such as hierarchical access control, sandbox isolation, budget constraints, and rollback mechanisms, to control the scope and cost of tool invocation and prevent error propagation. While these approaches have made progress in improving controllability, a systematic, unified perspective and transferable strategic paradigm are still lacking regarding how to explicitly model the instability of the tool itself and the uncertainty of its output during the planning phase, and how to drive a more conservative execution pace and decision boundaries.

3. Method

In agent-based tasks targeting real-world toolchains, abstracting a single solution process into state-driven sequential decision-making makes it easier to characterize the sources and transmission paths of uncertainty. The overall algorithm flow is presented here, as shown in Figure 1.

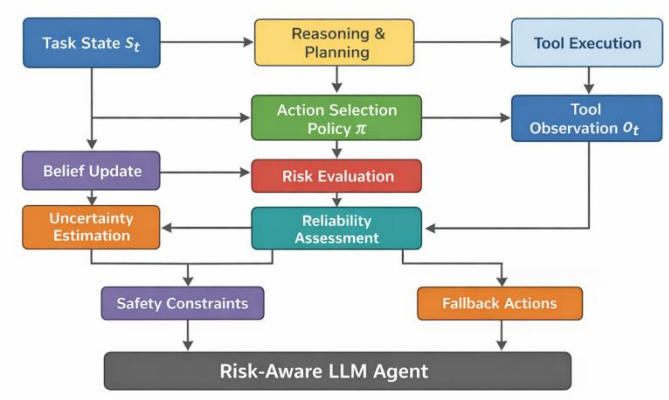

Figure 1. Algorithm overall process architecture

Let the task state be s_t, and the available actions include natural language inference actions and tool invocation actions, denoted as $a_t \in A$. Tool invocation returns an observation o_t and triggers a state transition. Due to issues such as tool timeouts, permission changes, return noise, and semantic drift, both transitions and observations carry uncertainty. Therefore, we use the belief state $b_t(s)$ to represent the posterior distribution of the true state, replacing single-point states for planning. The planning objective not only pursues task gains but also needs to explicitly penalize the risk exposure caused by uncertainty. Using a risk-sensitive comprehensive objective to select the action sequence π is more in line with deployment requirements, for example:

$$\pi^* = arg\max_{\pi} E[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)] - \lambda R(\pi)$$

In this model, $r(s_t, a_t)$ represents immediate returns, γ is a discount factor, $R(\pi)$ represents risk, and λ controls the degree of conservatism. The key to this model is incorporating instrumental uncertainty into the construction of $R(\pi)$, allowing the planning phase to favor more robust and verifiable action paths, rather than reactively correcting mistakes after failure.

To quantify tool uncertainty, a reliability parameter ρ_k is introduced for each tool k, characterizing the probability that the tool returns a usable and consistent result in the current context. The reliability can be updated online via the execution log; a simple exponential sliding update can reflect recent drift without adding excessive overhead.

$$\rho_k^{(t)} = (1-\alpha)\rho_k^{(t-1)} + \alpha\, 1(tool\ k\ success\ at\ t)$$

Where α is the update rate and $1(\bullet)$ is the indicator function. To couple reliability with planning, the risk term can be defined as the cumulative penalty for the risk of tool invocation in the selected action. For example, the risk of each invocation can be approximated as $(1-\rho_{k(a_t)})$, thereby prompting the strategy to prioritize the more stable class

among multiple feasible tools, or to tend to switch to a lower-risk alternative action when evidence is insufficient.

During execution, the agent's perception of the environment needs continuous correction based on tool observations. Let the observation returned by the tool be $P(o_t)$, the observation model be $P(o_t/s_t, a_t)$, and the transition model be $P(s_t/s_{t-1}, a_{t-1})$. Then, the belief update adopts a standard Bayesian filtering approach:

$$b_t(s) = \eta P(o_t/s, a_t) \sum_{s^*} P(s/s^*, a_{t-1}) b_{t-1}(s^*)$$

Where η is the normalization constant. To reflect the uncertainty of the tool, the observation likelihood can be modulated by the reliability, for example, by converting signals such as anomalous formats, conflicting evidence, and low confidence into a smaller $P(o_t/s, a_t)$, thereby naturally reducing the dependence on the observation at the belief level and avoiding the amplification of erroneous observations in subsequent planning.

Conservative implementation strategies are put into practice through explicit safety constraints and action gating, limiting high-risk actions to those with sufficient evidence and meeting reliability standards. Let $\mathcal{U}$ be the set of unsafe events, requiring that the probability of unsafety under the sense of belief does not exceed a threshold δ:

$$\sum_{s \in \mathcal{U}} b_t(s) \le \delta$$

Furthermore, a reliability threshold and secondary verification trigger condition are set for tool calls. Execution is only allowed if both the reliability and current uncertainty simultaneously meet the constraints; otherwise, a more conservative approach is adopted, such as switching to an interpretable alternative tool, adding consistency checks, or directly requesting supplementary information. A simple and implementable gating rule is as follows:

$$a_t = \begin{cases} execute\ tool\ k & \rho_k \ge \tau \ \bigwedge \ H(b_t) \le \kappa \\ fallback\ action, otherwise \end{cases}$$

Where $H(b_t)$ represents belief entropy, used to measure current uncertainty, and τ and κ are thresholds. By incorporating both reliability and uncertainty into execution gating, the strategy can automatically converge behavioral boundaries when there are fluctuations in the toolchain, conflicting evidence, or ambiguous states, reflecting the consistent synergy between risk perception at the planning level and conservatism at the execution level.

4. Experimental Results and Analysis

4.1 Dataset Introduction

This paper uses ToolBench as the research dataset. ToolBench is an open-source benchmark dataset built for large language model tool usage capabilities. Its core consists of a large-scale tool set and corresponding task instructions, covering common tool-based scenarios such as information retrieval, computation and data processing, text and structured information manipulation, and multi-step combined invocation. The dataset typically provides structured descriptions of tools, such as tool name, function description, parameter constraints, return fields, and invocation examples, and is accompanied by numerous natural language tasks. This requires agents to select tools, construct parameters, plan the invocation sequence, and integrate results after understanding the objective, making it suitable for characterizing closed-loop behavior from reasoning to execution.

To align with the theme of risk-aware planning for tool uncertainty, this paper, based on the task and tool descriptions in ToolBench, treats the tool invocation process as an interaction with noisy observations and potential failures. It records the execution status and availability signals of the returned content for each invocation, thus constructing training samples that can be used to model reliability and uncertainty. Specifically, the task text and tool specifications serve as planning inputs, while tool returns and execution status serve as observation inputs. This supports the construction of key variables needed for risk items and conservative execution gating, including scenarios such as failure or timeout, abnormal return format, information conflicts, and unstable results. This approach preserves the reproducibility of open-source datasets while enabling the systematic study of robust planning and safe execution strategies under tool uncertainty without relying on proprietary toolchains.

4.2 Experimental setup

To ensure reproducibility and fair comparison, this paper completes all training and evaluation processes in a unified hardware and software environment, fixing the random seed and recording key dependency versions. The overall implementation is based on Python and PyTorch, with the same runtime configuration used for both inference and execution to avoid numerical and behavioral differences caused by different environments. A summary of specific hardware, software, and major hyperparameter configurations is shown in Table 1.

Table 1. Hardware and software environment and main training hyperparameter settings

Item	Setting
GPU	NVIDIA RTX 4090 24GB
CPU	Intel Xeon (≥16 cores)
RAM	64 GB
OS	Ubuntu 22.04 LTS
Python	3.10
PyTorch	2.2
CUDA	12.1
Mixed Precision	Enabled (FP16)
Batch Size	8
Training Epochs	30
Optimizer	AdamW
Learning Rate	1e-4
Weight Decay	1e-2
Gradient Clipping	1.0
Discount Factor γ	0.99
Risk Weight λ	0.5
Reliability Update Rate α	0.1
Reliability Threshold τ	0.7
Belief Uncertainty Threshold κ	1.0

Safety Probability Bound δ	0.05
Max Tool Calls per Episode	8
Tool Timeout	10 s
Random Seed	42

In terms of training settings, the agent adopts a trajectory-based offline learning paradigm, constructing training samples from task instructions, tool specifications, call sequences, and tool returns. The planning side uses risk-sensitive targets to drive policy updates, while the execution side introduces reliability and uncertainty gating, and sets safety thresholds and fallback strategies to constrain high-risk calls. During the evaluation phase, the tool call budget and timeout limits are consistent with those used in training, ensuring that different methods are compared under the same resource constraints.

4.3 Experimental Results and Analysis

To facilitate a systematic comparison of existing methods from the perspective of tool uncertainty and safe execution, this paper selects representative works related to tool augmentation agents, planning and decision-making, and safety constraints, and summarizes their methodological characteristics and performance under a unified evaluation index system. Table 2 summarizes the comparison methods and the proposed method on four indicators.

Table 2. Experimental results compared with other models

Method	Accuracy	Task Success (%)	Tool Success (%)	Constraint Viol. (%)
Han et al.[8]	72.14	68.32	64.90	11.28
Doshi et al.[9]	74.83	70.27	66.41	10.52
Qin et al.[10]	76.92	73.18	69.85	9.74
Zhou et al.[11]	78.31	74.92	71.44	9.11
Xu et al.[12]	79.24	76.88	73.52	8.64
Patil et al.[13]	80.37	77.43	74.81	8.22
Schick et al.[14]	81.12	78.59	75.66	7.94
Ours	88.93	86.44	84.27	4.11

The experimental results show that the baseline methods exhibit relatively consistent gradient improvements across the four metrics, indicating that capability enhancements in tool-calling tasks are often chain-like: more stable planning leads to easier task completion, fewer tool calls result in fewer errors, and ultimately, violations are more easily suppressed. The last row of the table shows a significant hierarchical difference compared to the other methods, particularly in the two execution-related metrics, manifested as more stable tool completion and higher task closure completion. This difference typically means that the method is not only stronger at the generation level but, more importantly, handles the uncertainties of external tools more effectively, allowing the strategy to maintain consistency and controllability when encountering unreliable returns, formatting anomalies, or unclear states.

Another noteworthy point is that the change in the violation rate is not simply synchronized with the first three metrics, but rather seems to be independently lowered by a conservative execution mechanism: when the system can identify risk signals earlier and trigger rollbacks or degradations, violations decrease more significantly, which in turn reduces invalid calls and error propagation. In other words, the advantage is not just about increasing accuracy, but about minimizing the cost of errors, manifested as fewer out-of-bounds behaviors, more stable tool interactions, and smoother task convergence paths. Unlike the incremental improvements brought about by stronger reasoning or larger models alone, these results are more like structural gains resulting from the combined effect of risk perception at the planning stage and gating constraints at the execution stage.

The reliability threshold determines when an agent is allowed to continue executing tool calls. Too low a threshold may amplify the impact of unreliable tool outputs, while too high a threshold may excessively trigger conservative backoff, affecting overall decision-making efficiency. To examine the impact of this gating strength on model stability, a sensitivity analysis was designed below, using changes in accuracy to characterize its effect. The experimental results are shown in Figure 2.

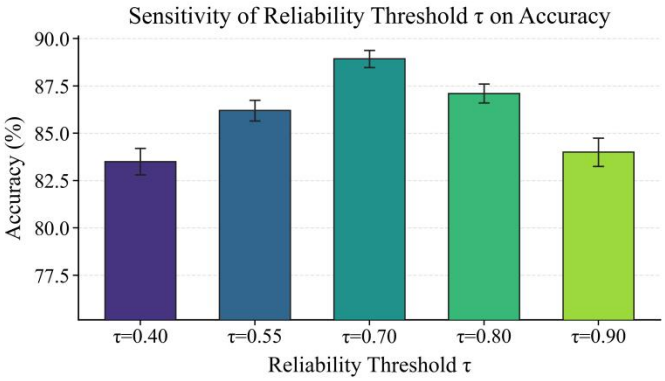

Figure 2. Sensitivity experiment of the reliability threshold to accuracy

A moderate reliability threshold tends to yield better accuracy, while accuracy declines when the threshold is set too low or too high. This suggests that gating strength is neither necessarily better for the loser nor the more conservative the approach; rather, it's about balancing two types of risks: one is allowing noise and misleading information from unreliable calls to occur, and the other is excessive blocking that prevents valuable tool information from entering the decision-making process promptly, thus affecting the consistency of overall judgment.

Furthermore, this unimodal variation aligns with the intuitive mechanisms of risk perception and conservative execution. With a low threshold, the system is more likely to treat unstable returns as valid evidence, and errors propagate more quickly to subsequent steps, leading to skewed decisions. With a high threshold, while the system is more cautious, it triggers rollbacks and degradation actions more frequently; insufficient information leads to planning relying more on incomplete state estimates, further impacting the final judgment. A threshold in the middle range seems to allow reliability assessment and uncertainty control to work in tandem, filtering out obviously unstable calls without overly

narrowing the execution space, resulting in more stable overall performance.

5. Conclusion

This paper addresses the real-world deployment challenges of LLM Agents under tool uncertainty. Centering on risk-aware planning and conservative execution, it constructs an integrated strategy framework encompassing state belief updates, uncertainty characterization, reliability assessment, and safety constraints and fallback mechanisms. By treating tool invocation as a potentially failed and noisy interaction and explicitly introducing risk terms and gating constraints at the decision-making level, the method can more stably maintain action boundaries in complex toolchains and multi-step tasks, reducing the cascading impact of unreliable observations on subsequent inference and execution. Comparative experiments show that this strategy demonstrates a more balanced improvement in task closure completion, tool invocation stability, and constraint compliance. This indicates that in tool-centric agent applications, simply relying on stronger generation or longer inference is insufficient to guarantee controllability and trustworthiness; introducing interpretable risk modeling and execution constraints is a key path to driving the engineering implementation of agents.

Looking to the future, the concepts of risk-aware planning and conservative execution are expected to be further extended to a wider range of demanding scenarios, such as automated operation and maintenance and data pipelines, financial compliance and audit assistance, medical documentation and clinical decision support, government and legal retrieval and verification, and enterprise-level workflow orchestration. These areas share common characteristics: complex toolchains, limited fault tolerance, and high costs of errors. Future work can be deepened in three directions: First, building more granular tool reliability modeling and calibration mechanisms, enabling reliability to better adapt to changes in context, input distribution, and system version. Second, extending security constraints from single-step gating to global constraints and budget management at the long-term planning level, ensuring agents maintain consistent risk control strategies throughout long tasks. Third, introducing more standardized log and audit interfaces to structure and accumulate risk signals, decision-making basis, and rollback reasons, forming a traceable, replayable, and compliant agent execution chain. With advancements in these directions, the framework proposed in this paper is expected to provide a more general and trustworthy interaction paradigm for tool-enhanced agents, promoting their large-scale application and long-term stable operation in critical business processes.

References

[1] 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.

[2] Liu H, Dou Z Y, Wang Y, et al. Uncertainty Calibration for Tool-Using Language Agents[C]//Findings of the Association for Computational Linguistics: EMNLP 2024. 2024: 16781-16805.

[3] Luo W, Dai S, Liu X, et al. Agrail: A lifelong agent guardrail with effective and adaptive safety detection[J]. arXiv preprint arXiv:2502.11448, 2025.

[4] Xiang Z, Zheng L, Li Y, et al. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning[J]. arXiv preprint arXiv:2406.09187, 2024.

[5] Xie Y, Yuan Y, Wang W, et al. ToolSafety: A Comprehensive Dataset for Enhancing Safety in LLM-Based Agent Tool Invocations[C]//Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing. 2025: 14146-14167.

[6] Ye J, Li S, Li G, et al. Toolsword: Unveiling safety issues of large language models in tool learning across three stages[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024: 2181-2211.

[7] Zhang H, Huang J, Mei K, et al. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents[J]. arXiv preprint arXiv:2410.02644, 2024.

[8] Han J, Buntine W, Shareghi E. Towards uncertainty-aware language agent[J]. arXiv preprint arXiv:2401.14016, 2024.

[9] Ruan Y, Dong H, Wang A, et al. Identifying the Risks of LM Agents with an LM-Emulated Sandbox[C]//The Twelfth International Conference on Learning Representations. 2024.

[10] Qin Y, Liang S, Ye Y, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis[J]. arXiv preprint arXiv:2307.16789, 2023.

[11] 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.

[12] Xu B, Peng Z, Lei B, et al. Rewoo: Decoupling reasoning from observations for efficient augmented language models[J]. arXiv preprint arXiv:2305.18323, 2023.

[13] Patil S G, Zhang T, Wang X, et al. Gorilla: Large language model connected with massive apis[J]. Advances in Neural Information Processing Systems, 2024, 37: 126544-126565.

[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.