Future Technology
Open Access Journal
ISSN 2832-0379

February 2027| Volume 06 | Issue 01 | Pages 93-104

Journal homepage: https://fupubco.com/futech

https://doi.org/10.55670/fpll.futech.6.1.8

Future Publishing LLC

Article

Meta-learning-driven agent architecture optimization for autonomous self-correcting agents

Zifa Liu*

College Of Artificial Intelligence, Lishui University, Lishui, Zhejiang, China, 323000

ARTICLE INFO

Article history:
Received 05 June 2026
Received in revised form
30 October 2026
Accepted 24 November 2026

Keywords:
LLM agents, Meta-learning, Self-correction,
Procedural memory,
Prompt–workflow optimization,
Risk-aware control

*Corresponding author
Email address:
zfliuedu@phc.ac.cn

DOI: 10.55670/fpll.futech.6.1.8

ABSTRACT

Large language model agents must be able to recover from task failures, accumulate effective experience, and maintain an appropriate balance in the frequency of human intervention. This study proposes MetaSCA, a meta-learning-based self-correcting agent. Based on a frozen Qwen2.5-7B-Instruct model, MetaSCA employs a lightweight controller, prompt and workflow optimization, and procedural memory to enable agents to adjust their execution processes based on task feedback. Furthermore, the system assesses the task's confidence and potential risks based on current execution results and then selects the next processing method accordingly. According to the evaluation scheme in the research, MetaSCA achieved a task success rate of 54.6 ± 1.7% in five experiments on WebArena, higher than the 48.8 ± 2.0% achieved by Promptbreeder. The ratios of MetaSCA for the two indicators of manual intervention and constraint violation were 15.2 +/- 1.4% and 6.2 +/- 1.4. Those of Promptbreeder were 13.9 +/- 1.5% and 9.2 +/- 1.0%. Later ablation experiments and perturbation analysis examine the effects of the correction strategies, memory mechanisms, and manual intervention on the system performance. Appendix A presents the analysis of the risk calibration and intervention selection decision rules.

1. Introduction

1.1 Background and motivation

Large language models (LLMs) are increasingly used as agents that pursue goals through interaction with external environments. Such intelligent systems obtain information, perform reasoning, and invoke tools based on the task status, and then combine the returned results to determine the next action. Task planning, tool usage, experience memory, and feedback processing all affect task completion [1, 2]. During long-term task execution, the agent may encounter tool failures, information shortages, or environmental changes. Some operations may also require authorization. Improper handling may lead to irreparable consequences. Therefore, the agent must be able to recover from errors and determine when human intervention is needed. Previous studies have explored these issues from different perspectives. SelfCorrect-Agent uses the agent's execution trajectory to summarize experience and improve subsequent performance [3]. Relevant reviews summarized agent optimization as two methods: adjusting model parameters or improving task performance by modifying prompts and retrieving external information [4]. Research on agent controllability mainly focuses on protection mechanisms, execution supervision, and human regulation [5]. These studies also raise an interesting question: Under the condition that the core structure remains unchanged, how can the agent combine error correction, workflow adjustment, and experience reuse, and reasonably decide when human intervention is needed? Experience reuse has formed a relatively mature research direction. Agent Workflow Memory organizes the experiences formed during task execution into reusable workflows. ExpeL extracts useful practices from past experiences. Voyager saves the learned skills as executable programs. AutoManual collates the operation methods and rules specific to certain environments [6-9]. MetaSCA further explores how to combine these reusable correction experiences with task-adaptive controllers and autonomy gates, enabling agents to adjust their execution methods and human intervention requirements based on task conditions. The research uses three indicators to evaluate system performance: task completion rate, human intervention rate, and constraint violation rate. MetaSCA adds a lightweight meta-controller around a frozen LLM. The controller chooses whether to continue, retry, replan, switch tools, roll back, or escalate. A prompt–workflow optimizer converts feedback into candidate execution changes. Validated corrections are retained in procedural memory, while risk-aware control determines whether a proposed action can proceed. The

study evaluates this arrangement through task success rate (TSR), human-intervention rate (HIR), and constraint-violation rate (CVR).

1.2 Research objectives and contributions

This study mainly focuses on the following aspects: Firstly, when the main large language model remains unchanged, select the appropriate correction operation based on the task execution situation; Secondly, save the verified workflow corrections as reusable experiences; Thirdly, evaluate the system performance from three aspects: task completion rate, manual intervention rate, and constraint violation rate. Based on this, this study proposes a controller-based correction method that selects six correction operations based on execution feedback. At the same time, the system can adjust the prompt content and workflow, and save the verified effective methods through the memory module for subsequent tasks. In addition, the study also designs an autonomy gating mechanism to determine whether the agent continues to complete the task independently, takes corrective measures, or requests human intervention based on the execution situation.

2. Related works

2.1 Reasoning and acting architectures

ReAct integrates reasoning with environmental interaction, enabling the agent to decide subsequent actions based on the observed results [10]. Regarding research on tool learning and agent architecture, it further discusses the roles of task planning, tool selection, and feedback processing in agent operation [11,12]. These studies provide a foundation for MetaSCA's execution process, enabling it to balance error correction, workflow adjustment, and risk control during task execution.

2.2 Reflection and self-correction mechanisms

Reflexion records language reflection in situational memory, helping the agent improve subsequent attempts. Self-Refine utilizes the same LLM to repeatedly generate feedback and modify the output based on the feedback [13, 14]. SAGE combines reflection reasoning with memory management, while FuseMind further introduces a prediction mechanism [15,16]. These studies show that agents can improve subsequent performance by leveraging execution feedback and past experience. MetaSCA further focuses on selecting appropriate corrective actions based on feedback and adjusting the execution process accordingly. Procedural memory is also an important research direction for experience reuse. Agent Workflow Memory saves the extracted workflow for reference in subsequent tasks. ExpeL summarizes useful insights and examples from existing experience, Voyager saves verified executable skills, and AutoManual organizes operation rules based on the interaction process [6-9]. Based on these studies, MetaSCA combines experience retrieval with an adaptive correction controller and determines whether manual intervention is needed based on task risk.

2.3 Automatic prompt and workflow optimization

Promptbreeder improves prompts through evaluation and mutation [17]. Symbol learning expands the optimization scope to language models and agent components [18], Prompt Alchemy mainly studies prompt optimization in code generation tasks [19], and GAAPO combines multiple prompt generation strategies in evolutionary search [20]. These studies all attempt to use feedback to improve prompts and instructions. MetaSCA extends the optimization scope to prompt content and execution rules. Modifications that perform well after evaluation will be retained as a reference for subsequent adjustments to the execution strategy.

2.4 Meta-learning and feedback adaptation

Model-agnostic meta-learning (MAML) uses a small number of gradient updates to find an appropriate initial parameter state, enabling the model to adapt more quickly to new tasks [21]. MUSE, on the other hand, focuses on self-assessment and behavior regulation in unfamiliar environments, exploring how the model can adjust its subsequent actions based on its own judgment [22]. These two studies provide a basis for task adaptation and feedback-driven behavioral adjustment. MetaSCA applies the inner- and outer-loop update ideas of MAML to correct the controller design. The entire process does not require adjusting the main model; it only needs to update the lightweight controller to change the agent's decision-making method. Workflow modification and memory preservation act on task execution and experience accumulation, respectively. The former adjusts specific execution instructions, while the latter retains verified corrective experiences for the agent to use in subsequent tasks.

2.5 Research gap and position of this study

MetaSCA mainly focuses on coordinating the correction process in task execution when manual intervention is required. Existing research has provided different solutions from aspects such as execution, reflection, prompt optimization, and experience memory. MetaSCA further combines the task-adaptive controller with immediate correction, while preserving effective experiences for subsequent tasks. The study also analyzes the impact of the controller, workflow, memory module, and autonomy gating mechanism on the system performance by removing different components for comparison.

3. Methodology

3.1 Overall MetaSCA architecture

MetaSCA employs a fixed-parameter LLM as the execution core, with multiple functional modules on the periphery to handle task status synchronization, process execution feedback, select correction strategies, adjust prompts and workflow, save experiences, and control the execution process. The entire architecture is centered on the fixed backbone model and adapts to different tasks by adjusting correction strategies and workflow. Figure 1 illustrates the relationships among the various modules.

During the execution of a single task, the agent generates an action plan based on the received observation information and uses autonomous gating to decide whether it can continue execution. The results returned by the environment will be used to analyze the execution situation and help the agent select an appropriate correction method. After the task is completed, verified effective correction experiences are saved and used by other tasks. The controller's meta-training is carried out separately and completed through support rounds and query rounds. During actual execution, the controller makes decisions based solely on the feedback available for the current task.

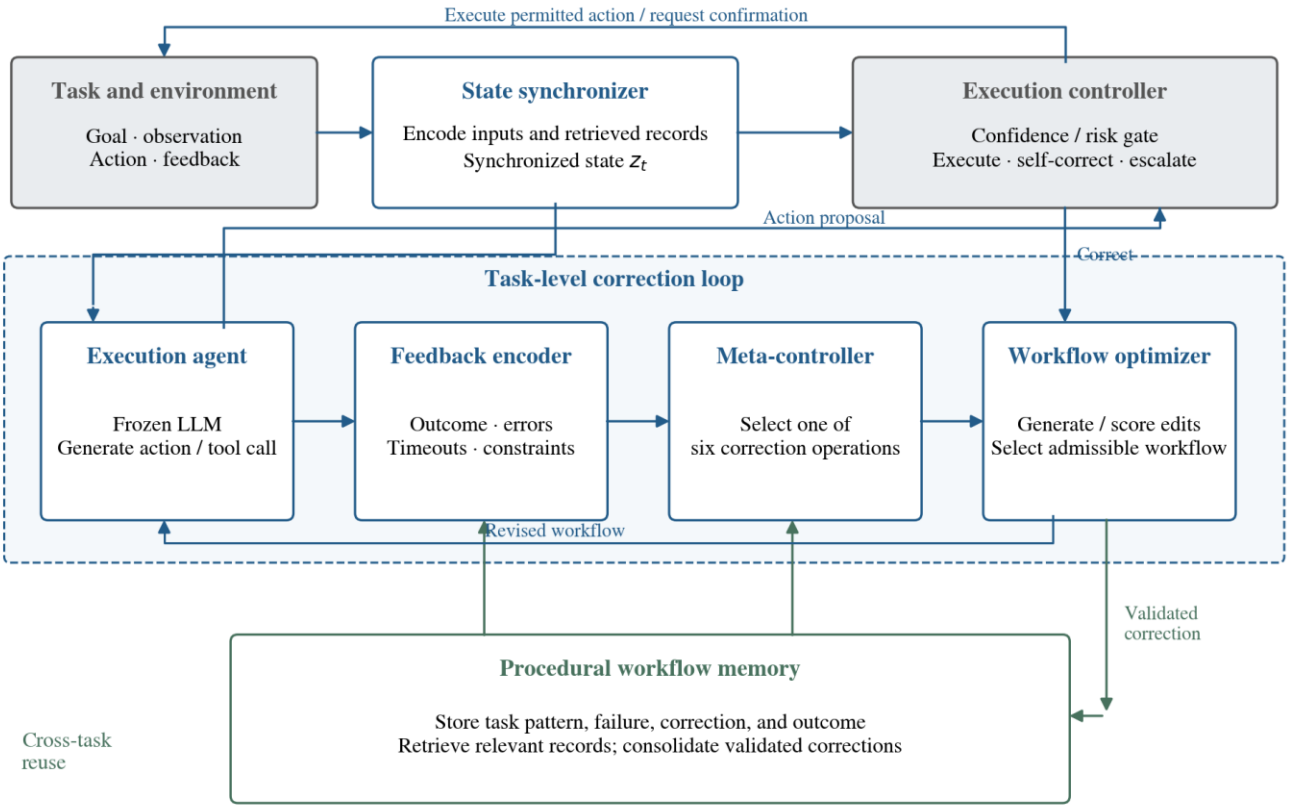

Figure 1. The figure shows the task execution and correction process of MetaSCA. The state synchronization module is responsible for transmitting task information, the frozen execution agent adjusts the action based on feedback, the meta-controller selects the correction strategy and implements the adjustment through the workflow revision. Procedural memory saves the verified correction records. Before the agent performs operations in the environment, it also needs to undergo the judgment mechanism of autonomous gating. The meta-training of the controller adopts independent support rounds and query rounds, and the specific process is described in Section 3.3.

Task correction, workflow adjustment, experience preservation, and execution permissions are handled by four components, respectively. The meta-controller selects corrective measures based on task feedback; the optimizer modifies the workflow and determines whether the modification is effective. Verified experiences are stored in the program memory for subsequent tasks to reference. The gating mechanism ensures the accuracy of actual operations by restricting or preventing the current operation when necessary and deciding whether further correction is required or whether human intervention is needed.

3.2 State and feedback synchronization

In the decision step t, the system integrates the task objective g, the current observation o_t, the environment action from the previous moment a_{t-1}, the execution feedback f_t, and the retrieved memory representation m_t into the current state:

$$z_t = E_\theta([u_g ; u_{o,t} ; u_{a,t-1} ; u_{f,t} ; m_t]) \in \mathbb{R}^d \quad (1)$$

Among them, u_g, $u_{o,t}$, $u_{a,t-1}$, and $u_{f,t}$ represent the numerical representations of the corresponding inputs; the semicolon indicates the concatenation operation, E_θ represents the state encoder, and d is the input dimension of the controller. It is necessary to distinguish between the memory representation in numerical form m_t and the memory bank used for storing records M.

The observation information can include text-based accessibility trees or tool outputs. Feedback is used to record the original actions and their execution results, including tool errors, timeouts, and constraint signals. The system classifies faults into five categories: planning, tools, observations, constraints, and termination, and marks each execution result as successful, partially successful, or failed. For failed execution processes, the fault category to which they belong is also determined.

The core language model always uses fixed parameters. The state encoder and controller are managed by parameters θ and ϕ, respectively, and are updated independently without changing the main model's parameters. This design separates the main model's update process from external modules and clarifies their respective adjustment scopes.

3.3 Episodic meta-policy adaptation

The controller C_ϕ scores the six operations based on the state representation z_t, which forms the action set U = {Continue, Retry, Replan, Switch Tool, Rollback, Escalate}. The controller consists of two fully connected hidden layers, each containing 256 and 128 neurons, and uses the GELU activation function and Dropout. Finally, the probabilities of each corrective operation are calculated through a six-class Softmax layer. For task i, D_i^{sup} and D_i^{qry} represent the support set and query set, respectively. The controller first uses the support set for internal parameter updates, as shown in the following formula:

$$\phi_i' = \phi - \alpha \nabla_\phi L_i^{sup}(\phi ; D_i^{sup}) \quad (2)$$

Here, the learning rate α is used to control the parameter adjustment amplitude of the controller in the current task.

Table 1. Semantics of the six controller operations

Operation	Effect on execution
Continue	Retain the current workflow and advance to the next admissible action.
Retry	Repeat the current operation after incorporating returned errors or corrected arguments.
Replan	Revise the remaining subgoals or action sequence using the current state and failure feedback.
Switch Tool	Replace the current tool or invocation route and revise the affected workflow step.
Rollback	Restore an available prior state before applying a revised continuation; use requires an environment-supported recovery operation.
Escalate	Pause the proposed operation and request a confirmation decision before continuation.

After the internal update is completed, the adjusted parameters are evaluated through the query set:

$$\phi* = arg\ min_{\phi}\ (1/N) \sum_{i=1}^{N} L_i^{qry}(\phi_i'\ ;\ D_i^{qry}) \quad (3)$$

Here, N represents the number of meta-tasks participating in the external parameter update, which is different from the number of training rounds. L_i^{sup} and L_i^{qry} represent the controller losses on the support set and query set, respectively. Equation (3) gives the meta-learning optimization objective, and the specific learning rate used is as described in Section 4.1. To ensure reliable evaluation results, the support and query sets for the same meta-task are independent, and benchmark test results are not used to construct the parameter adaptation target.

This update method draws on the basic idea of MAML [21], adjusting the controller parameters at both the intra-task and inter-task levels. The specific update effect also depends on the loss function used and how it is monitored. Section 4.3 further analyzes the impact of this component on overall performance through a contrast experiment without the controller.

3.4 Reward feedback and failure attribution

MetaSCA evaluates task execution performance. During the calculation process, it considers whether the task has been completed, whether it violates constraints, and the costs resulting from human intervention and execution.

$$r_t\ =\ \lambda_s\ S_t\ -\ \lambda_v\ V_t\ -\ \lambda_h\ H_t\ -\ \lambda_c\ C_t \quad (4)$$

Here, S_t represents the task completion status, V_t represents the constraint violation signal, H_t andC_t respectively represent the cost of human intervention and the normalized execution cost. Each λ weight is non-negative and independent of the internal learning rate α. The reward value reflects the task's overall execution effect and is also used to evaluate candidate solutions and record effective experiences. The relevant signals and weight settings are provided in Appendix A.

When the task is not completed as expected, the system compares the fault classification in Section 3.2 to determine the cause of the problem and then decides on the appropriate course of action. For instance, if there is an error in the tool invocation, one can try replacing the tool; if there is a constraint violation, the relevant operations need to be suspended and, in some cases, handled manually. For other execution issues, the system will also adjust the task plan accordingly.

3.5 Automatic Prompt–Workflow Optimization

If the fault can be repaired, the optimizer will generate a candidate workflow W_k and make adjustments based on the original prompts and execution rules.

$$J_k\ =\ \beta_1\ R_k\ -\ \beta_2\ V_k\ -\ \beta_3\ D(W_k, W_{old}) \quad (5)$$

R_k represents the expected task recovery benefit, V_k represents the constraint violation risk that the candidate solution may bring, and D represents the difference between the candidate workflow and the current workflow W_{old}. The β weights are non-negative and balance the recovery benefit, risk, and modification cost. The system selects the candidate solution with the highest score from those that meet the requirements. Through this scoring method, the workflow's modification extent and potential risks are also included in the solution evaluation. Table 1 lists the six controller operations and their specific impacts on task execution.

Validated corrections are then available for retrieval. Only the correction schemes that have been confirmed as valid by the evaluators and have passed the constraint checks will be saved in the procedural memory. The verification mainly relies on the feedback from the executed correction, while the benchmark test scores are recorded separately. The verified correction schemes can be retrieved and used in subsequent tasks.

3.6 Procedural Workflow Memory

Procedural memory mainly stores the effective correction experiences formed during the task execution, including relevant task information, failure reasons, and processing results. Here, M represents the complete memory record, and m_t is the numerical summary for the controller. Its update method is as follows:

$$\begin{aligned} m_{t+1}\ &=\ (1\ -\ \rho)m_t\ +\ \rho\ \psi(z_t, a_t, r_t),\ \ if\ v_t\ =\ 1 \\ m_{t+1}\ &=\ m_t,\ \ if\ v_t\ =\ 0 \end{aligned} \quad (6)$$

Here, ρ ∈ [0,1] represents the update coefficient, and ψ is responsible for converting the verified corrections into a numerical representation with the same dimension as m_t. Besides, v_t represents whether the correction has passed the verification. Formula (6) is used to update the numerical summary, and the specific workflow correction is still saved as an independent record in M. Schemes that have not passed verification are not saved in long-term memory. When retrieving memory, the system combines the current task state and the effects of existing correction records to find suitable reference experiences.

Whether the correction experiences in the memory can be reused depends on whether the current task environment is consistent with the original situation. Further discussion on this is provided in Section 5.2.

3.7 Risk-aware autonomous control

For each environmental operation to be executed, the system calculates two metrics: the execution confidence $c_t \in [0,1]$ and the constraint risk score $q_t \in [0,1]$. The former reflects whether the operation is expected to facilitate the progress of the task, while the latter is used to assess the potential risk of violating constraints caused by the operation. Both metrics must be calculated based on the currently available information before the operation is executed. The definition of the threshold decision d_t is as follows:

$$\begin{aligned} d_t\ &=\ execute,\quad c_t\ \geq\ \tau_c\ and\ q_t\ <\ \tau_r \\ d_t\ &=\ self-correct,\ c_t\ <\ \tau_c\ and\ q_t\ <\ \tau_r \\ d_t\ &=\ escalate,\quad q_t\ \geq\ \tau_r \end{aligned} \quad (7)$$

Among them, τ_c and τ_r represent the confidence threshold and the risk threshold, respectively. The threshold mechanism will make a judgment before the operation is submitted for execution in the environment. During execution, the controller can also choose to escalate based on the feedback received. Both methods will eventually lead to the same confirmation interface. Therefore, even if the threshold mechanism is removed, the system can still retain the manual intervention requests initiated by the controller on its own initiative.

The risk threshold will be adjusted based on the violation rate actually observed within each update interval k:

$$\tau_r^{k+1} = clip(\tau_r^k - \eta(\nu_k - V*), \tau_{min}, \tau_{max}) \quad (8)$$

Among them, η represents the update rate, ν_k represents the violation frequency within this interval, V* represents the target violation level, and τ_{min} and τ_{max} are used to define the adjustment range of the threshold. When the actual violation rate is higher than the target level, the system will reduce τ_r to make the gating mechanism more strict. When calculating ν_k, only the obtained observations can be used. For missing or delayed results, they need to be processed according to the corresponding rules. The calibration process evaluates the accuracy of the risk judgment by comparing the predicted score with the actual task completion and violation results.

3.8 Training and Execution Algorithm

Algorithm 1 separates the meta-training of the controller and the actual task execution. In the meta-training stage, it is updated through inner and outer layers to optimize the initial controller state in the training rounds. In the execution stage, the workflow is adjusted based on task feedback, and verified memory information is called.

4. Results and Analysis

4.1 Experimental Setup

This study mainly uses WebArena [23] to test the task execution capabilities of intelligent agents in web environments, focusing on examining the task completion status, the need for human intervention, and violations of constraints. WebArena contains a total of 812 tasks, and all tasks run in a self-hosted network environment. After the 812 tasks of WebArena were hierarchically classified by category, 487 training tasks, 162 validation tasks, and 163 test tasks were designated, respectively. The experiment also used AgentDojo [24] as a supplementary benchmark, including 97 regular tasks and 629 security cases, with the proportion of regular tasks being 58/19/20.

This chapter mainly compares the performance of different methods based on the test results of WebArena. The experiment adopts a unified task division and evaluation setting. Specific parameters are shown in Table 2. Table 3 presents the performance comparison on WebArena. Section 5.3 will further analyze the impact of task templates, cross-task memory retention, and evaluation sequence on the experimental results. All experiments are based on the Qwen2.5-7B-Instruct [25] base model. WebArena generates text observation information through an accessibility tree for the agent to determine the current web state. AgentDojo provides feedback to the agent through tool invocation. The experiment set the maximum context length to 8,192 tokens, the temperature to 0.2, and each task could execute up to 30 environmental actions.

This study selected ReAct, Reflexion, Self-Refine, and Promptbreeder [10,13,14,17] as reference methods. To make the comparison fairer, the same model architecture and tool interface were used in the experiment, and the context lengths, generation temperatures, and number of environment actions for each method were also the same. Other experimental settings are described in Section 5.3. TSR represents the proportion of completed tasks out of all tasks. HIR represents the ratio of manual intervention requests to executable operations. Intervention requests may be initiated by the execution gate or voluntarily proposed by the meta-controller.

Table 2. Experimental settings recorded for MetaSCA.

Setting	Value
Backbone	Qwen2.5-7B-Instruct; frozen parameters
Hardware	Intel Core i7-12700K; NVIDIA RTX 4090; 32 GB RAM
Software	Python 3.10; PyTorch 2.3.1; CUDA 12.1
Context and generation	Maximum 8,192 tokens; temperature 0.2
Environment-action budget	30 actions per task
Independent runs	5
Controller hidden layers	256 and 128 units; GELU; Dropout
Inner and outer learning rates	1×10^{-4} and 5×10^{-5}
Initial gate thresholds	τc = 0.70; τr = 0.35
WebArena partition	487 training / 162 validation / 163 test tasks

Algorithm 1. MetaSCA training and execution.

Input: task g, environment E, workflow W_0, controller ϕ, memory M, thresholds τc and τr
Output: terminal task state, final workflow, and validated memory
1 Meta-train ϕ on support/query episodes using Eqs. (2)–(3).
2 Initialize the task state, workflow W ← W_0, and task-local feedback.
3 While the task is active and its action budget remains:
4 Observe E, retrieve relevant records from M, and construct zt.
5 Propose an environment action and obtain confidence ct and risk qt.
6 Apply Eq. (7) to determine execution, self-correction, or escalation.
7 Execute the proposed action only when permitted; otherwise route the current state to correction or confirmation.
8 Associate feedback with each executed action and calculate the reward when its outcome becomes available.
9 Select u ← Cϕ(zt) and identify the relevant failure category.
10 If u is Escalate, obtain a confirmation decision before continuation.
11 If u requests correction, generate and score workflow candidates.
12 Apply the selected admissible edit and return to the gate.
13 Consolidate the correction in M and update mt only after validation.
14 Update τr from available violation observations using Eq. (8).
15 Return the terminal state, final workflow, and validated memory.

Table 3. WebArena performance under the study-specific protocol, mean ± SD across five runs.

Method	TSR (%) ↑	HIR (%) ↓	CVR (%) ↓
ReAct [10]	41.8 ± 1.9	23.6 ± 2.1	8.8 ± 0.8
Reflexion [13]	46.9 ± 2.2	20.7 ± 1.8	7.1 ± 0.7
Self-Refine [14]	45.7 ± 1.8	18.9 ± 1.7	7.6 ± 0.9
Promptbreeder [17]	48.8 ± 2.0	13.9 ± 1.5	9.2 ± 1.0
MetaSCA	54.6 ± 1.7	15.2 ± 1.4	6.2 ± 0.6

After receiving the request, the system decides whether to continue execution or replan based on the task requirements and feedback from the evaluator. CVR is used to count invalid, unauthorized, or safety-requirement violations, as well as unrevocable situations that occur during operation. Since the existing data cannot be statistically categorized using a unified standard for the number of violations of different methods, only a simple result description of CVR is provided here.

The experimental results are expressed as the average value and standard deviation of five runs, and the differences between methods are calculated in percentage points. Figure 2–3 shows the 95% confidence interval of the mean values of the five runs, and the half-width of the interval is calculated as $2.776 \times SD/\sqrt{5}$. This interval reflects fluctuations across multiple runs, and differences between methods are still presented in the descriptive results. The AgentDojo test set has only 20 tasks. The scores and statistical quantities reported in this document cannot be directly associated with each task, so it is currently not possible to accurately compare the performance of different methods.

4.2 Overall performance comparison

After removing the meta controller, the task completion rate of MetaSCA dropped from 54.6% to 50.1%. Removing the prompts and workflow optimizer caused a 3.2-percentage-point decrease in the completion rate. The violation rate improved from 6.2% to 7.3%. The task completion rate decreased by 2.4%. These changes indicate that several components contributed to MetaSCA's task completion. Table 3 shows comparisons under the study-specific WebArena task partition and action protocol.

4.3 Ablation and mechanism validation

Table 4 reports the removal of the four focal components. Removing the meta-controller reduced TSR by 4.5 percentage points, from 54.6% to 50.1%. Removing the prompt–workflow optimizer reduced TSR by 3.2 points and increased CVR from 6.2% to 7.3%. Removing procedural memory reduced TSR by 2.4 points. These comparisons describe each component's contribution within the full architecture. After removing the autonomy threshold, the proportion of manual intervention requests dropped from 15.2% to 10.8%, while the violation rate rose from 6.2% to 10.4%. Even without the threshold judgment, the controller could still request manual confirmation, so the manual confirmation function was unaffected. The experimental results show that as the number of manual intervention requests decreased, the number of violation operations also increased. When the intelligent agent autonomously completes tasks, it still needs to control the operational risks. In every experiment, only one component is removed, and the remaining components are left unchanged.

Table 4. MetaSCA component-removal results on WebArena, mean ± SD across five runs.

Variant	TSR ↑ (%)	HIR ↓ (%)	CVR ↓ (%)
Full MetaSCA	54.6 ± 1.7	15.2 ± 1.4	6.2 ± 0.6
w/o Meta-Controller	50.1 ± 2.1	16.8 ± 1.7	6.9 ± 0.8
w/o Prompt–Workflow Optimizer	51.4 ± 1.9	16.1 ± 1.5	7.3 ± 0.7
w/o Procedural Memory	52.2 ± 2.0	15.9 ± 1.6	6.7 ± 0.6
w/o Risk-Aware Autonomy Gate	53.1 ± 1.8	10.8 ± 1.2	10.4 ± 1.1

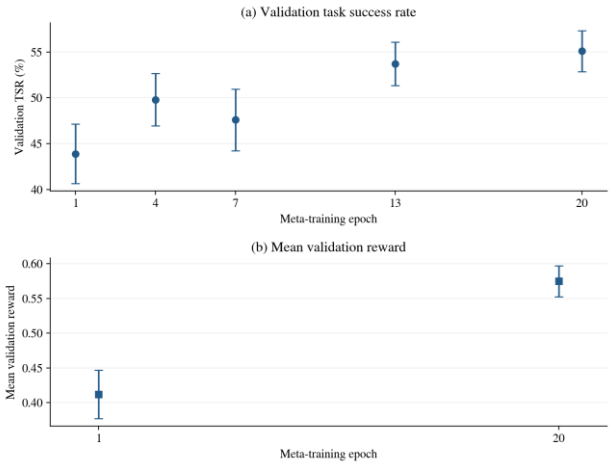

Figure 2. Validation checkpoints: (a) task success rate at epochs 1, 4, 7, 13, and 20; (b) mean validation reward at epochs 1 and 20. Error bars denote pointwise t-based 95% confidence intervals derived from the reported five-run standard deviations.

Furthermore, the study examined the interaction between workflow adjustments and memory functions, and the relationship between the system's threshold judgment and the controller's active request for manual confirmation.

4.4 Convergence and meta-adaptation analysis

Figure 2 shows that as training progresses, both the model's task success rate and the validation reward improve overall. Although the success rate dropped briefly in the 7th cycle, it increased from 43.9% to 55.1% in the 20th cycle, and the validation reward also rose from 0.412 to 0.575. The figure shows results and confidence intervals for each cycle. improved. Overall, as training progresses, the TSR gradually increases, with a brief decrease in the 7th period. The average validation reward continues to increase, indicating that the overall performance of the model has

4.5 Robustness, stability and autonomous control

As the noise level increases from 10% to 30%, the TSR decreases from 53.8 ± 1.8% to 47.9 ± 2.4%, and the HIR increases from 16.0 ± 1.5% to 22.6 ± 2.0%. Overall, the greater the interference with the observed information, the more difficult it is for the agent to complete the task and the more human assistance is required. As the noise increases, the task completion rate gradually decreases, and the number of human intervention requests increases. This indicates that the accuracy of the observed information directly affects the agent's task performance. The accuracy rates under single-step and two-step feedback delays were 52.9 ± 2.0% and 49.6 ± 2.3%, respectively. When faced with unseen tasks, the

accuracy rate was 50.7 ± 2.2%, the HIR was 19.1 ± 1.7%, and the CVR was 6.9 ± 0.8%. These results reflect MetaSCA's performance in the presence of feedback delays and task changes.

5. Discussion

5.1 Correction mechanisms and related approaches

MetaSCA uses task feedback to select correction schemes, adjust work processes, and save verified experiences. Table 4 shows that removing the meta-controller caused the largest decrease in task success rate, indicating that it plays an important role in selecting correction schemes. Removing the workflow optimizer or program memory also decreased the task completion rate, suggesting that these two components contribute to improving task execution effectiveness. Agent Workflow Memory, ExpeL, Voyager, and AutoManual all provide methods to improve task execution by leveraging past experience. On this basis, MetaSCA combines the selection of correction schemes with experience reuse and controls the operation process through risk judgment. The comparison methods in the experiment mainly involved task execution, reflection, content optimization, and prompt search. More models that use memory and risk control can be added later to further test MetaSCA's performance. After removing the gating mechanism, the HIR decreased from 15.2% to 10.8%, and the CVR increased from 6.2% to 10.4%. Compared with Promptbreeder, MetaSCA had a higher task completion rate, more human intervention requests, and a lower violation rate. These results indicate that there is a certain trade-off between task completion effectiveness, human intervention requirements, and operational risks.

5.2 Procedural Memory and Failure Analysis

The procedural record will store problems that occur during task execution and their verified solutions, which the system can use as a reference when encountering similar tasks in the future.

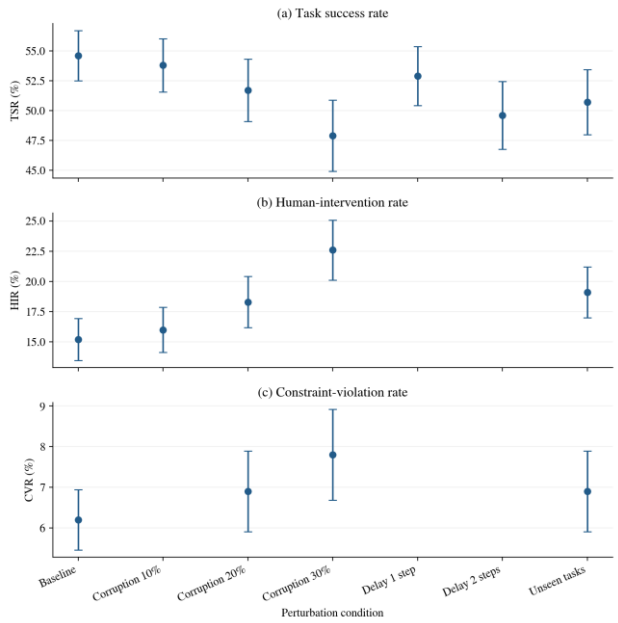

Figure 3. Observation perturbation results: (a) task success rate; (b) human-intervention rate; (c) constraint-violation rate. Conditions are displayed as categorical points with pointwise t-based 95% confidence intervals of five-run means. Panels include the conditions with reported mean and standard deviation for each metric.

Based on this, the system can select tools, decide whether to retry or adjust the execution process. Equation (6) divides the memory content into a numerical summary for the controller and specific correction records. Memory may also bring new problems. For example, the experiences retrieved by the system may come from different types of tasks and may not be applicable to the current situation. After the tool interface changes, the previously effective workflow may also become invalid. If the task feedback is incomplete, it is difficult for the system to determine whether the modification was successful. The timing of feedback arrival also affects adjustments to the risk threshold. These situations indicate that whether memory can function depends on whether the current task and execution environment are similar to those in the past.

5.3 Evaluation scope and reproducibility

This study mainly tested MetaSCA in WebArena, and the experimental results were obtained from five runs. Due to differences in the testing tasks and website environments, how memory is utilized across tasks also affects system performance. AgentDojo was used as another testing environment to supplement the examination of MetaSCA's task completion and security performance. The execution sequence and evaluation criteria of the experiment also need to be consistent to enable other researchers to reproduce the results more accurately. To help other researchers replicate the experiment, it is also necessary to describe the training process and key parameter settings of the model and controller, including the encoding method of state information, model initialization, tuning of baseline methods, and the settings of rewards and risk thresholds. Appendix A has already compared some parameter settings. Future work might focus on the memory effect after long-term use, as well as the time and cost required to complete tasks.

6. Conclusion

This study proposes the MetaSCA framework, aiming to assist frozen large language models in adjusting their execution process based on task feedback. This framework utilizes lightweight controllers to select correction strategies and continuously improves task execution through memory and risk control. In the WebArena test, the task success rate of MetaSCA was 54.6%, higher than that of Promptbreeder (48.8%). The proportion of human-intervention rate was 15.2%, slightly higher than that of Promptbreeder (13.9%), and the constraint-violation rate was 6.2%, lower than that of Promptbreeder (9.2%). The ablation experiments also showed that the complete architecture helps to increase the task completion rate. After removing the autonomy threshold, manual intervention requests decreased, and the violation rate increased, indicating that the system still needs to consider operational risks when autonomously executing tasks. Future work can continue to improve MetaSCA in aspects such as the applicable range of task templates, consistency of manual intervention judgment, and task execution efficiency.

Ethical issue

The author is aware of and complies with best practices in publication ethics, specifically with regard to authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance with policies on research ethics. The author adheres to publication requirements that the submitted work is original and has not been published elsewhere.

Data availability statement

WebArena is at https://github.com/web-arena-x/webarena and AgentDojo at https://github.com/ethz-spylab/agentdojo. The backbone model card is at https://huggingface.co/Qwen/Qwen2.5-7B-Instruct. This report has not yet made public the proprietary code, prompt words, test data division method, parameter settings, and process memory records of MetaSCA. The main experimental results are presented in the text, while the supporting materials include Appendix A with the experimental code, data, analysis results, and related charts, which facilitate understanding and reproducing the experiment.

Conflict of interest

The author declares no potential conflict of interest.

References

[1] M. Abou Ali, F. Dornaika and J. Charafeddine, "Agentic AI: a comprehensive survey of architectures, applications, and future directions," Artificial Intelligence Review, vol. 59, Art. 11, 2026. https://doi.org/10.1007/s10462-025-11422-4

[2] W. Xu, C. Huang, S. Gao and S. Shang, "LLM-Based Agents for Tool Learning: A Survey," Data Science and Engineering, vol. 10, no. 4, pp. 533–563, 2025. https://doi.org/10.1007/s41019-025-00296-9

[3] K. He, D. Fu, L. Yang, W. Xu and C. Wang, "SelfCorrect-Agent: Toward robust and generalizable LLM-based agents," Neurocomputing, vol. 670, Art. 132546, 2026. https://doi.org/10.1016/j.neucom.2025.132546

[4] S. Du, J. Zhao, J. Shi, Z. Xie, X. Jiang, Y. Bai and L. He, "A Survey on the Optimization of Large Language Model-based Agents," ACM Computing Surveys, vol. 58, no. 9, pp. 1–37, 2026. https://doi.org/10.1145/3789261

[5] M. H. Nguyen, D. H. Nguyen, B. O'Sullivan and H. D. Nguyen, "On Controllability in Agentic AI: A Survey," Minds and Machines, vol. 36, no. 2, Art. 29, 2026. https://doi.org/10.1007/s11023-026-09783-y

[6] Z. Z. Wang, J. Mao, D. Fried and G. Neubig, "Agent Workflow Memory," Proceedings of Machine Learning Research, vol. 267, pp. 63897–63911, 2025. https://proceedings.mlr.press/v267/wang25bx.html

[7] A. Zhao, D. Huang, Q. Xu, M. Lin, Y. J. Liu and G. Huang, "ExpeL: LLM Agents Are Experiential Learners," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 17, pp. 19632–19642, 2024. https://doi.org/10.1609/aaai.v38i17.29936

[8] G. Wang, et al., "Voyager: An open-ended embodied agent with large language models," Transactions on Machine Learning Research, 2024. https://openreview.net/forum?id=ehfRiF0R3a

[9] M. Chen, Y. Li, Y. Yang, S. Yu, B. Lin and X. He, "AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning," Advances in Neural Information Processing Systems, vol. 37, pp. 589–631, 2024. https://doi.org/10.52202/079017-0019

[10] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," International Conference on Learning Representations, 2023. https://openreview.net/forum?id=WE_vluYUL-X

[11] C. Qu, et al., "Tool learning with large language models: a survey," Frontiers of Computer Science, vol. 19, no. 8, Art. 198343, 2025. https://doi.org/10.1007/s11704-024-40678-2

[12] U. Nisa, M. Shirazi, M. A. Saip and M. S. M. Pozi, "Agentic AI: The age of reasoning—A review," Journal of Automation and Intelligence, vol. 5, no. 1, pp. 69–89, 2026. https://doi.org/10.1016/j.jai.2025.08.003

[13] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan and S. Yao, "Reflexion: language agents with verbal reinforcement learning," Advances in Neural Information Processing Systems, vol. 36, pp. 8634–8652, 2023. https://doi.org/10.52202/075280-0377

[14] A. Madaan, et al., "Self-Refine: Iterative refinement with self-feedback," Advances in Neural Information Processing Systems, vol. 36, pp. 46534–46594, 2023. https://doi.org/10.52202/075280-2019

[15] X. Liang, et al., "SAGE: Self-evolving Agents with Reflective and Memory-augmented Abilities," Neurocomputing, vol. 647, Art. 130470, 2025. https://doi.org/10.1016/j.neucom.2025.130470

[16] X. Ma, et al., "FuseMind: Fusing reflection and prediction elevates agent's reasoning capabilities," Neurocomputing, vol. 658, Art. 131755, 2025. https://doi.org/10.1016/j.neucom.2025.131755

[17] C. Fernando, D. S. Banarse, H. Michalewski, S. Osindero and T. Rocktäschel, "Promptbreeder: Self-referential self-improvement via prompt evolution," Proceedings of Machine Learning Research, vol. 235, pp. 13481–13544, 2024. https://proceedings.mlr.press/v235/fernando24a.html

[18] Y. Ou, et al., "Symbolic learning enables self-evolving agents," AI Open, vol. 6, pp. 314–322, 2025. https://doi.org/10.1016/j.aiopen.2025.11.004

[19] S. Ye, Z. Sun, G. Wang, L. Guo, Q. Liang, Z. Li and Y. Liu, "Prompt Alchemy: Automatic Prompt Refinement for Enhancing Code Generation," IEEE Transactions on Software Engineering, vol. 51, no. 9, pp. 2472–2493, 2025. https://doi.org/10.1109/TSE.2025.3589634

[20] X. Sécheresse, J. Y. Guilbert–Ly and A. Villedieu de Torcy, "GAAPO: genetic algorithmic applied to prompt optimization," Frontiers in Artificial Intelligence, vol. 8, Art. 1613007, 2025. https://doi.org/10.3389/frai.2025.1613007

[21] C. Finn, P. Abbeel and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," Proceedings of Machine Learning Research, vol. 70, pp. 1126–1135, 2017. https://proceedings.mlr.press/v70/finn17a.html

[22] R. Valiente and P. K. Pilly, "Metacognition for Unknown Situations and Environments (MUSE)," Neural Networks, vol. 194, Art. 108131, 2026. https://doi.org/10.1016/j.neunet.2025.108131

[23] S. Zhou, et al., "WebArena: A realistic web environment for building autonomous agents," International Conference on Learning Representations, 2024. https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html

[24] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," Advances in Neural Information Processing Systems, vol. 37, pp. 82895–82920, 2024. https://doi.org/10.52202/079017-2636

[25] Qwen Team, "Qwen2.5-7B-Instruct," official model card, 2024. https://huggingface.co/Qwen/Qwen2.5-7B-Instruct



This article is an open-access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

Appendix A
Controlled Decision-Rule Diagnostics

This appendix compares the performance of different decision rules through simulated cases, focusing on the impact of risk score calibration, reward weights, and gating conditions on the decision outcomes, and analyzing the changes in the score when it is interfered with. The cases are generated by a pre-defined random probability model. The following describes the specific generation process and evaluation method.

A.1 Task Generation and Evaluation Protocol

The experiment uses a fixed random seed of 361886 to generate 60 sets of template effect vectors. Each set contains 6 independent normal distribution variables with a mean of 0 and a standard deviation of 0.5. According to the template number, the experiment divides them into training set, validation set, and test set. Numbers 0 to 35, 36 to 47, and 48 to 59 correspond to the three data sets respectively. Each template generates 40 cases under five random seeds ranging from 2026 to 2030. Each run includes 1,440 training cases, 480 validation cases, and 480 test cases. The training set will be fully exported and saved as a case list. The experiment uses fixed decision rules, and the validation set is only used for score calibration without training the decision strategy.

For each case, the task difficulty and four task conditions are randomly generated, reflecting temporary situations, planning arrangements, tool usage, and constraint requirements respectively. The task difficulty and condition variables follow the Beta(2, 2) and Beta(1.5, 2.5) distributions respectively. The code combines these variables with the template effect and calculates the success probability and violation probability of the six candidate operations through logical functions. Among them, the results of the four operations are affected by the corresponding task conditions, while the upgrade processing is randomly generated separately. Subsequently, the model determines the success and violation results of the candidate operations through independent Bernoulli sampling based on the probabilities. The complete probability calculation method and the potential results of each operation are saved in the supporting code and CSV files. The experiment simulates the difference between the score and the real situation by adding fixed biases and random noise. The original success score is added a fixed bias and Gaussian noise with a standard deviation of 0.42 on the Logit scale.

The original risk score multiplies the real risk's Logit value by 1.45, subtracts 0.25, and adds Gaussian noise with a standard deviation of 0.38. The templates in the test set do not participate in score calibration, and all pre-defined random seeds are included in the experiment.

$$U_{i,a} = P_{i,a} - \lambda_v Q_{i,a} - \lambda_h h_a - 0.1 C_a \quad \text{(A1)}$$

P and Q represent the estimated success rate and the original risk score respectively. The success weight is set to 1, while the violation weight and intervention weight are λ_v = 2 and λ_h = 0.3 respectively. For the operation, h_a takes the value of 1, and for other operations, it is 0. The sequence of actions {Continue, Retry, Replan, Switch Tool, Rollback, Escalate} is arranged. Their standardized costs are {0.10, 0.20, 0.45, 0.35, 0.55, 1.00}. These costs are only used in the decision model in Appendix A.

$$q_{i,a}^{cal} = sigmoid(A\, logit(Q_{i,a}) + B) \quad \text{(A2)}$$

The experiment uses a logical calibrator to calculate parameters A and B based on the candidate operation results in the validation set. The L2 regularization parameter C is set to 1, and up to 1,000 calculations are performed. Test calibration is measured on the proposed Continue action before any gate selection. The calibration effect is evaluated by three indicators: ECE, Brier score, and AUROC. Among them, ECE divides the data into ten equal-width intervals, and the three indicators use the same test data. Only one operation is selected for each case. TSR, CVR, and HIR represent the proportions of cases where the task is successful, violations occur, and an upgrade handling is selected.

The Continue rule always selects the continue operation. The threshold gating rule decides whether to escalate the processing based on the original risk score of the continue operation: escalation occurs when the score reaches 0.35, and if the success confidence is lower than 0.70, retry is selected when the score is lower than this value, otherwise, the operation is continued. The utility rule calculates the utility value according to the calculation Eq. (A1) in the original score and selects the operation with the highest score. The calibrated utility gating rule uses the calibrated risk score to calculate the utility value and escalates the processing when the risk score of the selected operation reaches 0.35. All rules make decisions based on the pre-set scoring conditions.

Table A1. Decision-rule outcomes under the clean condition, mean ± SD across five seeds.

Decision rule	TSR (%)	HIR (%)	CVR (%)	Assigned cost
Continue	35.83 ± 3.19	0.00 ± 0.00	19.17 ± 1.37	0.100 ± 0.000
Threshold gate	51.83 ± 2.43	2.38 ± 0.32	18.21 ± 1.51	0.215 ± 0.002
Utility selection	62.96 ± 3.71	5.92 ± 0.92	10.96 ± 1.08	0.457 ± 0.009
Calibrated utility gate	64.87 ± 3.27	25.38 ± 2.27	9.04 ± 0.62	0.588 ± 0.017

Table A2. Paired task-success differences for the calibrated utility gate relative to each alternative decision rule.

Comparator	Difference (pp)	Bootstrap 95% CI	t (df = 4)	d_z	Holm p
Continue	29.04	24.50 to 33.04	14.53	6.50	0.0004
Threshold gate	13.04	8.04 to 17.71	5.85	2.62	0.0085
Utility selection	1.92	-0.75 to 4.00	1.78	0.80	0.1493

A.2 Decision trade-offs and paired analysis

The experimental results under clean conditions are shown in Table A1 and Figure A1. Compared to the Continue and Threshold gat rules, the calibration utility gate control rule completes more tasks and has fewer violations, but requires more human intervention. This rule has a 1.92 percentage points higher task success rate than Utility selection. However, this gap is not significant yet from statistical results. The average score of the calibration utility gate control rule is 0.333 under default settings, which is smaller than 0.347 of Utility selection. This implies that more tasks completed does not necessarily mean better performance. The paired analysis treats each experiment as an independent unit. During the statistics, samples are first selected from five random seeds, and then data are extracted from the 12 test templates corresponding to each selected seed, with this process repeated 3,000 times. Table A2 presents the results of the comparison of three groups of TSR under clean conditions, along with the corresponding two-sided paired t-test, standardized effect size, and adjusted p-values using the Holm method. Since five random seeds were used in the experiment, this design also determines the scope that the paired analysis can cover.

A.3 Reward weights risk calibration and perturbations

Figure A2 mainly compares the impact of different reward weights on the experimental results. The experiment adjusted the weights of violations and human intervention, testing 12 $\lambda_v \in \{0.5, 1, 2, 4\}$ and $\lambda_h \in \{0.1, 0.3, 0.6\}$, and recorded the task completion rate, proportion of human intervention, violation rate, and overall score in each case. Additionally, the experiment also compared four risk thresholds. After the threshold increased from 0.15 to 0.35, the proportion of human intervention decreased from 34.12% to 25.38%, while the violation rate rose from 7.42% to 9.04%. When the threshold was raised to 0.50, the operations selected by the system were exactly the same as those at 0.35, so the results in both cases were consistent. Table A3 presents the results of score calibration. After logical calibration, the average ECE decreases from 0.101 to 0.028, indicating that the calibrated scores are more accurate. AUROC remains unchanged because the relative order of various scores does not change before and after calibration. Figure A3(a) shows the grouped results after merging all test data, mainly used to visually present the calibration of the scores. Table A3 summarizes the results of each independent experiment.

Table A3. Risk calibration for the Continue action before gate selection, mean ± SD across five seeds.

Risk score	ECE	Brier score	AUROC
Raw risk	0.101 ± 0.013	0.156 ± 0.009	0.672 ± 0.022
Calibrated risk	0.028 ± 0.004	0.146 ± 0.006	0.672 ± 0.022

In the score perturbation experiment, the study perturbs 10% and 30% of the candidate operation scores. The specific method is to add random noise following a normal distribution with a mean of 0 and a standard deviation of 1.2 to the logarithmic probability of the score. The feedback delay experiment uses the score data of the previous to two cases under the same task template to simulate the situation where the system delays obtaining the current task feedback. Both sets of experiments maintain the original initial position, experimental results, and task quantity unchanged. Table A4 lists the TSR, HIR, and CVR of the two decision rules under different conditions, and Figure A3(b) compares the task success rates of the two rules. These experiments are mainly used to observe how much the performance of the decision rules is affected when the scores are perturbed or when there is feedback delay.

A.4 Failure records and reproducibility

The failure records include cases where tasks were not completed, or violations occurred during the execution process. This example illustrates residual uncertainty in score-based correction decisions. After the actual execution, the task still failed, and no violations occurred. For instance, in the first case of the random seed 2026 and template 48, the system chose Replan, with a prediction success rate of 0.819, a risk score of 0.062. However, the actual task was not completed, and no violations occurred. This indicates that there can sometimes be a discrepancy between the system's prediction results and the actual execution situation.

The supporting materials include experimental codes, data, results of each group of experiments, failure cases, and related charts, which are convenient for others to view and repeat the experiments. In addition, the time required for the system to select the operation is separately recorded, excluding the inference time of the large language model and the running time of external tools.

Table A4. Decision-rule results under score perturbations, mean ± SD across five seeds.

Condition	Decision rule	TSR (%)	HIR (%)	CVR (%)
Clean	Threshold gate	51.83 ± 2.43	2.38 ± 0.32	18.21 ± 1.51
Clean	Calibrated utility gate	64.87 ± 3.27	25.38 ± 2.27	9.04 ± 0.62
Corruption 10%	Threshold gate	51.37 ± 2.12	3.21 ± 0.48	18.29 ± 1.73
Corruption 10%	Calibrated utility gate	64.29 ± 3.32	24.33 ± 2.90	9.62 ± 0.98
Corruption 30%	Threshold gate	51.29 ± 1.97	4.88 ± 0.68	17.67 ± 1.96
Corruption 30%	Calibrated utility gate	63.87 ± 0.89	22.46 ± 2.81	10.25 ± 1.78
Delay 1 step	Threshold gate	52.17 ± 3.32	2.33 ± 0.37	19.12 ± 2.23
Delay 1 step	Calibrated utility gate	61.58 ± 2.79	25.67 ± 2.37	9.04 ± 1.39
Delay 2 steps	Threshold gate	52.62 ± 2.92	2.25 ± 0.31	18.88 ± 1.95
Delay 2 steps	Calibrated utility gate	61.38 ± 0.76	25.46 ± 2.42	9.62 ± 1.21

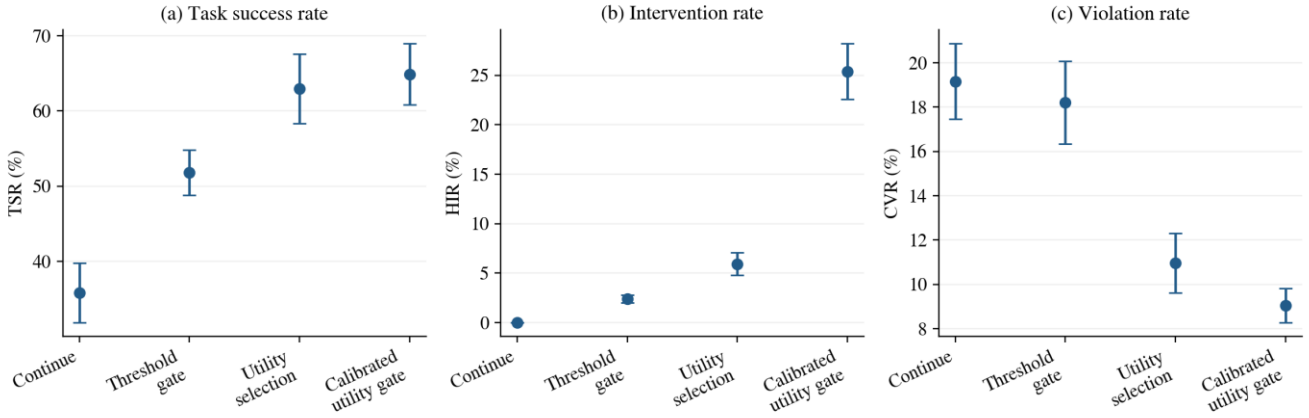

Figure A1. Clean-condition decision-rule outcomes: (a) task success rate; (b) intervention rate; (c) violation rate. Points show means across five seeds; error bars denote pointwise t-based 95% confidence intervals with four degrees of freedom. All decision rules are compared with the same candidate operation results.

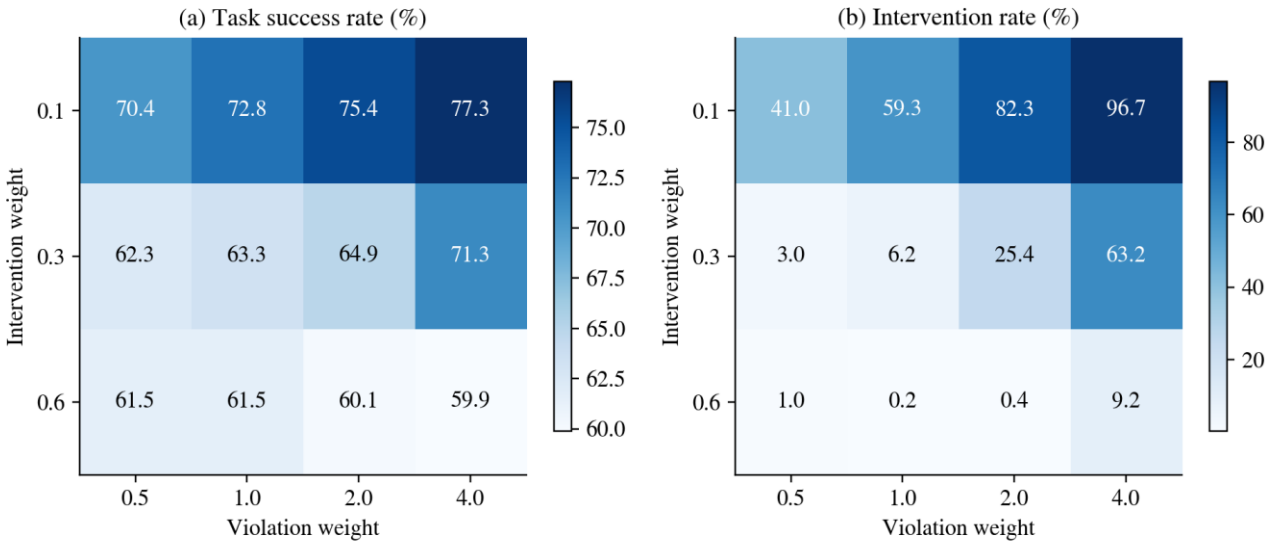

Figure A2. Reward Weight Sensitivity Analysis: (a) Task Success Rate; (b) Proportion of Intervention Requests. The figure shows the average performance of the calibrated utility gating rule in five experiments under different combinations of violation weights and intervention weights. The success weight and cost weight are fixed at 1 and 0.1.

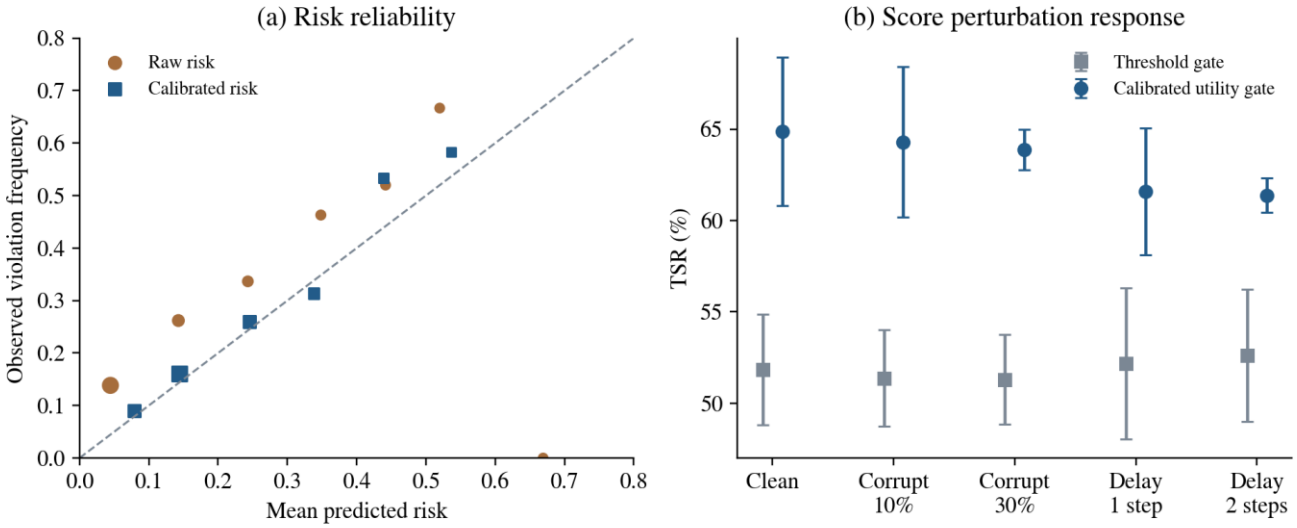

Figure A3. Calibration effect and score disturbance experiment results: (a) accuracy of the risk score; (b) performance of the score after disturbance. (a) The results of the "continue" operation in five test experiments. Each point represents a score range; the size of the point indicates the amount of data within that range, and the dotted line represents the ideal calibration state. (b) The average success rate of the tasks in the five experiments; the error lines represent the 95% confidence interval.